



The Addison Wesley Signature Series

MORE AGILE TESTING

LEARNING JOURNEYS FOR
THE WHOLE TEAM

JANET GREGORY
LISA CRISPIN

A MIKE COHN
Mike Cohn
Signature Book



Forewords by Elisabeth Hendrickson and Johanna Rothman

Elogios por las pruebas más ágiles

"Amo este libro. Ayudará a crear evaluadores realmente excelentes. Eso es bueno, ya que cualquiera que lea esto querrá tener uno en su equipo".

—Liz Keogh, entrenadora ágil, Lunivore Limited

"Este libro cambiará su forma de pensar y trasladará su atención de las pruebas a las pruebas. ¡Sí, no se trata del resultado, sino de la actividad!"

—Kenji Hiranabe, cofundador de Astah y director ejecutivo de Change Vision, Inc.

"En mi opinión, el desarrollo ágil se trata de aprendizaje; esa palabra captura el verdadero espíritu de lo que significa ágil. Cuando tuve la oportunidad de leer su nuevo libro, sólo pude decir: '¡Guau! Janet y Lisa se han sentido orgullosas. Este no es un libro sobre pruebas; Este es un libro sobre el aprendizaje. Sus claras explicaciones van acompañadas de fantásticas historias reales y una impresionante lista de libros, artículos y otros recursos. Aquellos de nosotros a quienes nos gusta aprender, a quienes nos encanta buscar más información, ¡podemos alegrarnos! Sé que siempre estás buscando algo interesante y útil; ¡Puedo garantizar que lo encontrarás aquí!'

—Linda Rising, coautora de Fearless Change: Patterns for Introducing New Ideas

"El primer libro de Janet y Lisa, Agile Testing, trazó algunos principios generales que todavía son importantes hoy en día, pero me dejaron preguntándome: '¿cómo?' En este segundo libro, adaptan esos principios al panorama de desarrollo actual, con aplicaciones móviles, DevOps y basadas en la nube entregadas en ciclos de lanzamiento cada vez más comprimidos. Los lectores obtienen herramientas de prueba específicas para la mente junto con nuevas prácticas y comentarios para acelerar el aprendizaje. Léelo hoy".

—Matt Heusser, director general, Desarrollo de Excelon

"Una guía excelente para el viaje ágil de su equipo, llena de recursos para ayudarlo con todo tipo de desafíos de prueba que pueda encontrar en el camino. Janet y Lisa comparten una gran experiencia con historias personales sobre cómo ayudaron a equipos ágiles a descubrir cómo obtener valor de las pruebas. Realmente me gusta cómo el libro está lleno de técnicas explicadas por profesionales líderes de la industria que han sido pioneros en ellas en sus propias organizaciones".

—Rachel Davies, entrenadora ágil, rebelde y coautora de Agile Coaching

"Permitanme aclararles esto: es difícil lograr una calidad y pruebas ágiles. Tiene matices, se basa en el contexto y no es apto para personas débiles de corazón. Para equilibrarlo de manera efectiva, necesita un asesoramiento pragmático, obtenido con tanto esfuerzo y del mundo real. Este libro lo tiene, no sólo de Janet y Lisa, sino también de cuarenta profesionales ágiles expertos adicionales. Consíguelo y aprende cómo impulsar eficazmente la calidad en tus productos ágiles y en toda tu organización".

—Bob Galen, consultor principal, R Galen Consulting Group y autor de Agile Reflections y Scrum Product Ownership

"Janet y Lisa lo han vuelto a hacer. Han combinado una experiencia de vida pragmática con una amplia narración para ayudar a las personas a llevar sus pruebas ágiles al siguiente nivel".

—Jonathan Rasmusson, autor de Agile Samurai: Cómo los maestros ofrecen un gran software

"En esta secuela de su excelente primer libro, Janet y Lisa han abrazado la madurez de la adopción ágil y la variedad de dominios en los que ahora se aplican enfoques ágiles. En More Agile Testing, han resumido las experiencias de expertos que trabajan en diferentes organizaciones ágiles y las han combinado con sus propios conocimientos en un conjunto de lecciones invaluables para los profesionales ágiles. Estructurado en torno a una variedad de áreas esenciales que los profesionales del software deben considerar, el libro examina lo que hemos aprendido sobre la aplicación ágil, a medida que ha ido creciendo su popularidad, y sobre las pruebas de software en el proceso. Aquí hay algo para todos, no solo para los probadores de software, sino también para personas en cualquier rol o dominio empresarial con interés en ofrecer calidad en un contexto ágil".

—Adam Knight, director de control de calidad, RainStor

"Este libro lo tiene todo: consejos prácticos e historias desde las trincheras. Ya sea que nunca hayas oído hablar de Agile o creas que eres un experto, aquí hay algo que te ayudará. Salta por el libro y prueba algunas cosas; Te prometo que serás un mejor probador y desarrollador".

—Samantha Laing, entrenadora y formadora ágil, Growing Agile

"More Agile Testing es una gran colección de historias e ideas que pueden ayudarle a mejorar no solo la forma en que realiza las pruebas, sino también los productos que construye y la forma en que trabaja. Lo que más me gusta del libro es lo fácil que es probar muchas de las ideas. Si un mensaje está claro es que, independientemente de su contexto y desafíos, hay cosas que puede intentar mejorar. Empiece hoy con algo pequeño y nada será imposible".

—Karen Greaves, entrenadora y entrenadora ágil, Growing Agile

"More Agile Testing es una extensa recopilación de experiencias, historias y ejemplos de profesionales que trabajan con pruebas en entornos ágiles en todo el mundo.

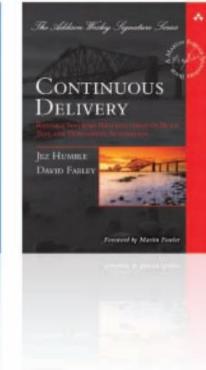
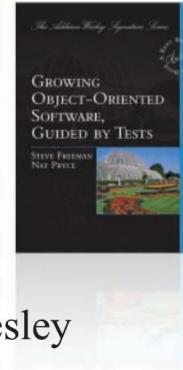
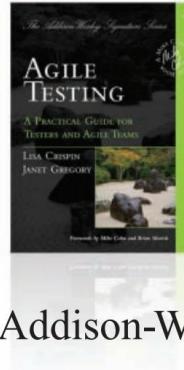
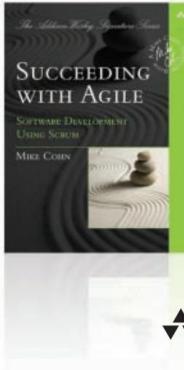
Cubre un amplio espectro, desde desafíos organizacionales y de contratación, técnicas y prácticas de prueba hasta orientación sobre automatización. La diversidad del contenido lo convierte en un excelente libro de cocina para cualquier persona dedicada al desarrollo de software a la que le apasione mejorar su trabajo y quiera producir software de calidad".

—Sigurdur Birgisson, ingeniero de asistencia de calidad, Atlassian

Pruebas más ágiles

La serie exclusiva de Addison-Wesley

Kent Beck, Mike Cohn y Martin Fowler, editores consultores



▼ Addison-Wesley

Visite informit.com/awss para obtener una lista completa de los productos disponibles.

Addison-Wesley Signature Series proporciona a los lectores información práctica y autorizada sobre las últimas tendencias en tecnología moderna para profesionales de la informática.

La serie se basa en una premisa simple: los grandes libros provienen de grandes autores.

Los títulos de la serie son elegidos personalmente por asesores expertos, autores de talla mundial por derecho propio. Estos expertos se enorgullen de poner sus firmas en las portadas, y sus firmas garantizan que estos líderes de opinión hayan trabajado estrechamente con los autores para definir la cobertura temática, el alcance del libro, el contenido crítico y la singularidad general. Las firmas de expertos también simbolizan una promesa a nuestros lectores: estás leyendo un clásico del futuro.



¡Asegúrate de conectarte con nosotros!

informit.com/socialconnect



informIT.com
THE TRUSTED TECHNOLOGY LEARNING SOURCE

Safari
Books Online

Pruebas más ágiles

Viajes de aprendizaje para todo el equipo

Janet Gregorio
Lisa Crispín

◆ Addison-Wesley

Upper Saddle River, Nueva Jersey • Boston • Indianápolis • San Francisco
Nueva York • Toronto • Montreal • Londres • Múnich • París • Madrid

Ciudad del Cabo • Sídney • Tokio • Singapur • Ciudad de México

Muchas de las designaciones utilizadas por fabricantes y vendedores para distinguir sus productos se consideran marcas comerciales.

Cuando esas designaciones aparecen en este libro y el editor tenía conocimiento de un reclamo de marca registrada, las designaciones se imprimieron con letras mayúsculas iniciales o en mayúsculas.

Los autores y el editor han tenido cuidado en la preparación de este libro, pero no ofrecen garantía expresa o implícita de ningún tipo y no asumen responsabilidad por errores u omisiones. No se asume ninguna responsabilidad por daños incidentales o consecuentes relacionados con o que surjan del uso de la información o los programas contenidos en este documento.

Para obtener información sobre la compra de este título en grandes cantidades o para oportunidades de ventas especiales (que pueden incluir versiones electrónicas, diseños de portada personalizados y contenido específico para su negocio, objetivos de capacitación, enfoque de marketing o intereses de marca), comuníquese con nuestro departamento de ventas corporativo. en corpsales@pearsoned.com o (800) 382-3419.

Para consultas sobre ventas gubernamentales, comuníquese con gobiernosales@pearsoned.com.

Si tiene preguntas sobre ventas fuera de los Estados Unidos, comuníquese con international@pearsoned.com.

Visítenos en la Web: informit.com/aw

Datos de catalogación en publicación de la Biblioteca del Congreso

Gregorio, Janet, 1953-

Pruebas más ágiles: viajes de aprendizaje para todo el equipo / Janet Gregory, Lisa Crispin.
páginas cm

Incluye referencias bibliográficas e índice.

ISBN 978-0-321-96705-3 (pbk.: papel alcalino)

1. Software de computadora—Pruebas. 2. Desarrollo ágil de software. I. Crispín, Lisa. II. Título.

QA76.76.T48G74 2015

005.1—dc23

2014027150

Copyright © 2015 Janet Gregory y Lisa Crispin

Ilustraciones de Jennifer Sinclair

Reservados todos los derechos. Impreso en los Estados Unidos de América. Esta publicación está protegida por derechos de autor y se debe obtener permiso del editor antes de cualquier reproducción prohibida, almacenamiento en un sistema de recuperación o transmisión en cualquier forma o por cualquier medio, ya sea electrónico, mecánico, fotocopia, grabación o similar. Para obtener permiso para utilizar material de este trabajo, envíe una solicitud por escrito a Pearson Education, Inc., Departamento de Permisos, 200 Old Tappan Road, Old Tappan, Nueva Jersey 07675, o puede enviar su solicitud por fax al (201) 236-3290. .

ISBN-13: 978-0-321-96705-3

ISBN-10: 0-321-96705-4 Texto

impreso en Estados Unidos en papel reciclado en RR Donnelley en Crawfordsville, Indiana.

Segunda impresión, octubre de 2015.

A mis nietos, Lauren, Brayden y Joe, quienes me hicieron
reír y jugar durante todo el año pasado.

—Janet

A mi familia, a los que todavía están aquí y a los que lamentablemente se han ido, y a mis queridos
amigos que forman parte de mi familia elegida.

—Lisa

Esta página se dejó en blanco intencionalmente.

Contenido

Prólogo de Elisabeth Hendrickson	xvii
Prólogo de Johanna Rothman	xix
Prefacio	xxi
Expresiones de gratitud	xxxx
Sobre los autores	xxxiii
Acerca de los contribuyentes	xxxv

Parte I	Introducción	1
Capítulo 1 Cómo han evolucionado las pruebas ágiles	3	
Resumen	6	
Capítulo 2 La importancia de la cultura organizacional	7	
Invertir tiempo	8	
La importancia de una cultura de aprendizaje	12	
Fomentar una cultura de aprendizaje	13	
Bucle de transparencia y retroalimentación	15	
Educar a la organización	17	
Gestión de probadores	19	
Resumen	20	

Parte II Aprendizaje para mejores pruebas**21**

Capítulo 3 Roles y Competencias	23
Competencias versus roles	24
Conjunto de habilidades en forma de T	28
Especialistas generalizadores	33
Contratar a las personas adecuadas	36
Probadores de incorporación	37
Resumen	39
Capítulo 4 Habilidades de pensamiento para las pruebas	41
Facilitando	42
Resolviendo problemas	43
Dar y recibir comentarios	45
Aprender el dominio empresarial	46
Habilidades de entrenamiento y escucha	48
Pensar diferente	49
organizando	51
Colaborando	52
Resumen	53
Capítulo 5 Conciencia técnica	55
Guiar el desarrollo con ejemplos	55
Habilidades de automatización y codificación	56
Habilidades técnicas generales	59
Entornos de desarrollo	59
Entornos de prueba	60
Sistemas de integración continua y control de código fuente	62
Prueba de atributos de calidad	63
Técnicas de diseño de pruebas	67
Resumen	67
Capítulo 6 Cómo aprender	69
Aprendiendo estilos	69
Recursos de aprendizaje	72
Conferencias, cursos, reuniones y colaboraciones	72
Publicaciones, podcasts y comunidades en línea	75
Tiempo para aprender	77
Ayudar a otros a aprender	79
Resumen	83

Parte III Planificación: para no olvidar el panorama general 85

Capítulo 7 Niveles de precisión para la planificación	87
Diferentes puntos de vista	87
Nivel de lanzamiento del producto	89
Nivel de característica	92
Nivel de historia	96
Nivel de tarea	96
Planificación de pruebas de regresión	97
Visualice lo que está probando	98
Resumen	100
Capítulo 8 Uso de modelos para ayudar a planificar	101
Cuadrantes de pruebas ágiles	101
Planificación de las pruebas del cuadrante 1	105
Planificación de las pruebas del cuadrante 2	105
Planificación de las pruebas del cuadrante 3	106
Planificación de las pruebas del cuadrante 4	107
Desafiando los cuadrantes	108
Utilizar otras influencias para la planificación	113
Planificación de la automatización de pruebas	115
Resumen	116

Parte IV Prueba del valor empresarial 119

Capítulo 9 ¿Estamos construyendo lo correcto?	121
Comience con el "por qué"	121
Herramientas para la participación del cliente	123
Mapeo de impacto	123
Historias Las 7 dimensiones del producto	126
Más herramientas o técnicas para explorar Inversión temprana para construir lo correcto	129
Resumen	134

Capítulo 10 La mentalidad del evaluador en expansión: ¿es esto?	Mi	¿Trabajo?	137
¿De quién es este trabajo?			137
Habilidades de análisis			137
empresarial Habilidades			140
de diseño de UX Habilidades			141
de documentación Tomar			142
la iniciativa Resumen			144
Capítulo 11 Obteniendo ejemplos			145
El poder de usar ejemplos			145
Guiar el desarrollo con ejemplos			148
ATDD			149
BDD			152
SBE			153
Dónde conseguir ejemplos			155
Beneficios de usar ejemplos			157
Posibles peligros del uso de ejemplos			159
Estancarse en los detalles			159
Falta de aceptación			160
Demasiadas pruebas de regresión			161
Aún no se sabe lo suficiente			161
La mecánica del uso de ejemplos para guiar la codificación			162
Resumen			162
Parte V Pruebas de investigación			163
Capítulo 12 Pruebas exploratorias			165
Crear cartas de prueba			168
Generación de ideas para los estatutos de prueba			171
Explorando con personas			171
Explorando con Tours			174
Otras ideas			175
Gestión de cartas de prueba			176
Gestión de pruebas basada en sesiones			176
Gestión de pruebas basada en subprocesos			178
Explorando en grupos			183
Registro de resultados para sesiones de pruebas exploratorias			185
Dónde encajan las pruebas exploratorias en las pruebas ágiles			188
Resumen			190

Capítulo 13 Otros tipos de pruebas	191
Tantas necesidades de pruebas	192
Pruebas de concurrencia	194
Internacionalización y Localización	195
Desafíos de las pruebas de regresión	200
Pruebas de aceptación del usuario	201
Pruebas A/B	203
Pruebas de experiencia de usuario	205
Resumen	207
Parte VI Automatización de pruebas	209
Capítulo 14 Deuda técnica en pruebas	211
Hazlo visible	212
Trabaje en el problema más grande y consiga el todo	217
Equipo involucrado	
Resumen	220
Capítulo 15 Pirámides de la automatización	223
La pirámide original	223
Formas alternativas de la pirámide	224
Los peligros de posponer la automatización de pruebas	227
Usar la pirámide para mostrar diferentes dimensiones	231
Resumen	235
Capítulo 16 Patrones y enfoques de diseño de automatización de pruebas	237
Involucrar a todo el equipo	238
Empezando bien	239
Principios y patrones de diseño	240
Pruebas a través de la API (a nivel de servicio)	241
Prueba a través de la interfaz de usuario	243
Mantenimiento de prueba	248
Resumen	251
Capítulo 17 Selección de soluciones de automatización de pruebas	253
Soluciones para equipos en transición	253
Enfrentando nuevos desafíos de automatización con todo el equipo	258
Lograr un consenso de equipo para soluciones de automatización	260

¿Cuánta automatización es suficiente?	262
Soluciones colaborativas para elegir herramientas para ampliar la automatización a grandes organizaciones	264
Resumen de otras consideraciones sobre automatización	268
	269
Parte VII ¿Cuál es su contexto?	271
Capítulo 18 Pruebas ágiles en la empresa ¿Qué queremos decir con “empresa”?	275
“Escalar” pruebas ágiles	276
Tratar con controles organizacionales	278
Coordinar múltiples equipos	283
Un equipo de prueba del sistema y su entorno	284
Herramientas consistentes	289
Coordinación a través de CI	289
Enfoques de control de versiones	290
Cobertura de prueba	291
Gestión de dependencias	292
Trabajar con terceros	292
Involucrar a los clientes en grandes organizaciones	294
Ventajas de llegar más allá del equipo de entrega	296
Resumen	297
Capítulo 19 Pruebas ágiles en equipos distribuidos ¿Por qué no Colocate?	299
Desafíos comunes	301
Cuestiones	302
culturales	302
Idioma Zonas	304
horarias	305
Dependencias	305
Estrategias	308
de planificación para afrontar la situación	308
Integración de equipos	309
Comunicación y colaboración	311
Colaboración a través de pruebas	312
Pruebas en el extranjero	

Ideas de herramientas para equipos distribuidos	319
Herramientas de comunicación	319
Herramientas de colaboración	319
Resumen	322
Capítulo 20 Pruebas ágiles para sistemas móviles e integrados	325
Similares, pero diferentes	326
Las pruebas son fundamentales	328
Enfoques ágiles	329
Resumen	337
Capítulo 21 Pruebas ágiles en entornos regulados	339
El mito de la “falta de documentación”	339
Ágil y cumplimiento	340
Resumen	346
Capítulo 22 Pruebas ágiles para almacenes de datos y empresas	347
Sistemas de inteligencia	347
¿Qué tienen de especial las pruebas de BI/DW?	348
Uso de datos de principios	351
ágiles: resumen de big data	352
de activos	357
críticos	360
Capítulo 23 Pruebas y DevOps	361
Una breve introducción a DevOps	361
DevOps y calidad	363
Cómo los evaluadores agregan valor de DevOps	371
Resumen	376
Parte VIII Pruebas ágiles en la práctica	379
Capítulo 24 Visualice sus pruebas	381
Comunicar la importancia de las pruebas	381
Visualice para la mejora continua	386
Visibilidad de las pruebas y sus resultados	390
Resumen	392

Capítulo 25 Poniéndolo todo junto	393
Prácticas de fomento de la confianza	394
Ejemplos de uso	394
Prueba exploratoria	395
Prueba de funciones	396
Aprendizaje continuo	397
Sensibilidad al contexto	399
Se realista	401
Crear una visión compartida	402
Resumen	405
Apéndice A Objetos de página en la práctica: ejemplos	407
Un ejemplo con Selenium 2: WebDriver	407
Usando la clase PageFactory	410
Apéndice B Iniciadores de provocación	413
 Glosario	415
Referencias	423
Bibliografía	435
Índice	459

Prefacio

Por Elisabeth Hendrickson

Hace apenas diez años, lo ágil todavía se consideraba radical. Franja. Extraño. El enfoque estándar para la entrega de software implicaba fases: analizar, luego diseñar, luego codificar y luego probar. La integración y las pruebas se produjeron sólo al final del ciclo. El ciclo completo de desarrollo tomó meses o años.

Si nunca ha trabajado en una organización con ciclos largos y fases discretas, la idea puede parecer un poco extraña ahora, pero era el estándar hace una década.

Antes, cuando las fases eran la norma y lo ágil aún era nuevo, la comunidad ágil estaba principalmente centrada en los programadores. Janet y Lisa y algunas otras personas de calidad y pruebas estaban allí. Sin embargo, muchos miembros de la comunidad ágil sintieron que el control de calidad se había vuelto irrelevante. Se equivocaron, por supuesto. El control de calidad cambió, se reformó para adaptarse al nuevo contexto, pero no desapareció.

Fueron necesarias personas como Janet y Lisa para mostrar cómo el control de calidad podría integrarse en equipos ágiles en lugar de pasarse por alto. Su primer libro juntos, Agile Testing, explicó cuidadosamente el enfoque de calidad de todo el equipo. Cubrieron los cambios culturales necesarios para integrar completamente las pruebas con el desarrollo. Explicaron cómo superar las barreras. Es un libro fantástico, y lo recomiendo mucho.

Sin embargo, quedaban preguntas. ¿Cómo podrían adaptarse las prácticas a diversos contextos? ¿Cómo empiezas? ¿Qué deberían aprender los evaluadores para ser más efectivos?

Este libro continúa donde lo dejó Agile Testing y responde esas preguntas y más.

Incluso si eso fuera todo lo que hiciera este libro, sería una excelente secuela.

Pero es más que eso. Dentro de estas páginas encontrará un tema, uno que Janet y Lisa han tejido con tanta destreza a lo largo del texto que tal vez ni siquiera se dé cuenta mientras lo lee. Así que les voy a llamar la atención sobre esto: este es un libro sobre adaptación.

Reflexionar y adaptarse es el único truco sencillo que puede permitir que su organización encuentre el camino hacia la agilidad. Experimente, pruebe algo diferente, extraiga las lecciones aprendidas, repita. Lo siguiente que sabrá es que su organización será ágil y flexible, capaz de adaptarse a las demandas del mercado y cumplir de forma incremental.

Este libro le enseña acerca de la adaptación al mismo tiempo que le enseña acerca de las pruebas ágiles.

La Parte II, “Aprender para mejorar las pruebas”, no trata sólo de cómo se aprende como individuo sino también de cómo construir una cultura de aprendizaje. La Parte VII, “¿Cuál es su contexto?”, no se trata solo de variaciones en metodología ágil adaptadas a diferentes situaciones; También es una guía de campo para varios tipos de adaptaciones.

El mundo está cambiando muy rápidamente. Hace apenas una década, lo ágil era extraño; ahora es la corriente principal. Hace apenas cinco años, tabletas como el iPad ni siquiera estaban en el mercado; ahora están en todas partes. Las prácticas, las herramientas, la tecnología y los mercados están cambiando tan rápido que es difícil mantenerse al día. No basta con aprender una manera de hacer las cosas; necesitas saber cómo descubrir nuevas formas. Necesitas adaptarte.

Este libro es un recurso fantástico para pruebas ágiles. También le ayudará a aprender a adaptarse y a sentirse cómodo con el cambio.

Espero que lo disfrutes tanto como yo.

Prefacio

Por Johanna Rothman

¿Qué hacen los probadores? Proporcionan información sobre el producto bajo prueba, para exponer riesgos para el equipo.

Eso es exactamente lo que Janet Gregory y Lisa Crispin han hecho en su nuevo libro, *More Agile Testing: Learning Journeys for the Whole Team*.

¿Tienes riesgos en tu agilidad? Hay muchas ideas que le ayudarán a comprender el valor de un ritmo sostenible, la creación de una organización de aprendizaje y su papel en las pruebas.

¿No está seguro de cómo realizar pruebas para un producto determinado, en un solo equipo o en un programa? También hay una respuesta para eso.

¿Cómo trabajas con personas en el siguiente cubo, al final del pasillo y en todo el mundo? Janet y Lisa estuvieron allí y lo hicieron. Su enfoque en roles y no en títulos es particularmente útil.

Hay muchas imágenes en este libro, por lo que no tendrá que preguntarse: "¿Qué significan?" Te lo muestran, no sólo te lo dicen.

Pruebas más ágiles: viajes de aprendizaje para todo el equipo es mucho más que un libro sobre pruebas. Es un libro sobre cómo utilizar las pruebas para ayudar a todo su equipo y, por extensión, a su organización, a realizar una transición hacia la metodología ágil de manera saludable.

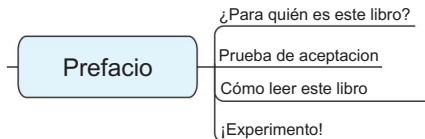
¿No se trata de eso de proporcionar información sobre la organización bajo prueba, exponer los riesgos en la organización?

Si es evaluador o administrador de pruebas, debe leer este libro. Si integra las pruebas en su organización, debe leer este libro.

¿De qué otra manera sabrás qué podrían estar haciendo los evaluadores?

Esta página se dejó en blanco intencionalmente.

Prefacio



Este libro continúa donde lo dejó nuestro primer libro, *Pruebas ágiles: una guía práctica para evaluadores y equipos ágiles*. Evitamos repetir lo que cubrimos en nuestro primer libro, pero brindamos suficiente contexto para que sea independiente si no ha leído *Agile Testing*. Nos referimos al primer libro como *Pruebas ágiles*, cuando pensamos que podría ser útil para el lector explorar conceptos básicos con más detalle.

¿Para quién es este libro ?

Asumimos que usted, el lector, no es un principiante en el mundo de las pruebas ágiles, que tiene algo de experiencia en pruebas ágiles y que ahora está buscando ayuda en áreas más allá de las pruebas ágiles. Si cree que le gustaría recibir una introducción al desarrollo ágil que incluya algunos conceptos básicos de las pruebas ágiles antes de leer este libro, *The Agile Sam-urai* (Rasmussen, 2010) es un excelente lugar para comenzar.

Este libro está dirigido a cualquier persona interesada en probar actividades en un equipo ágil. Según nuestra experiencia, esto incluye no sólo a los evaluadores y gerentes de pruebas, sino también a programadores, propietarios de productos, analistas de negocios, profesionales de DevOps, gerentes de línea... prácticamente todos.

Prueba de aceptacion

Además de compartir lo que hemos aprendido en los últimos años, queríamos que este libro fuera tan útil para nuestros lectores como el primero. Queríamos saber qué necesitaban saber los lectores del primer libro después

Al leerlo, pedimos a los profesionales de la lista de correo de Agile Testing que nos enviaran sus "pruebas de aceptación" para este segundo libro. Destilamos esas respuestas a esta lista de pruebas de aceptación para pruebas más ágiles e hicimos todo lo posible para satisfacerlas mientras escribíamos el libro.

Notarás que hemos utilizado un estilo utilizado en el desarrollo impulsado por el comportamiento. (BDD), del que hablaremos más en el Capítulo 11, "Obtención de ejemplos":

Dada <condición previa>,

Cuando <desencadenante, acción>,

Entonces <el resultado esperado>.

- Dado que soy un evaluador o gerente ágil, cuando contrate nuevos evaluadores sin experiencia ágil, aprenderé cómo ponerlos al día y evitar arrojarlos al abismo sin un chaleco salvavidas.
- Dado que soy miembro de un equipo ágil, cuando termine este libro, espero saber cómo combinar las pruebas exploratorias con las pruebas automatizadas y obtener una idea de la cobertura general de las pruebas, sin recurrir a pruebas pesadas. herramientas.
- Dado que soy un administrador de pruebas ágiles con experiencia, cuando termine este libro, entenderé cómo abordar técnicas de pruebas ágiles con múltiples equipos para permitir que mi exitosa organización ágil crezca.
- Dado que soy un administrador de pruebas ágil con experiencia, cuando termine de leer este libro, debería tener ideas sobre cómo coordinar las actividades de automatización de pruebas entre iteraciones y equipos, con ideas sobre cómo mejorar.
- Dado que soy un administrador ágil con experiencia, cuando leo Al leer este libro, entenderé cómo otros equipos han adaptado prácticas de pruebas ágiles para adaptarlas a su propio contexto y tendré ideas sobre cómo aplicarlas en el mío.
- Dado que soy un miembro de un equipo ágil que está interesado en las pruebas, cuando termine este libro espero tener ejemplos de cómo deberían y no deberían ser las pruebas y cómo puedo diseñarlas de manera efectiva.

- Dado que soy un tester ágil con experiencia, cuando encuentro un tema interesante en este libro sobre el cual me gustaría aprender más, puedo encontrar fácilmente referencias a recursos web u otros libros.
- Dado que soy un entrenador o gerente ágil con experiencia que Al leer el libro, cuando veo un concepto que ayudaría a mi equipo, entonces tengo suficiente información para poder idear una estrategia para lograr que el equipo pruebe un experimento.
- Dado que soy un miembro ágil del equipo que se preocupa por realizar pruebas y mantener a los clientes informados, cuando haya leído este libro, entenderé buenas formas de comunicarme con los miembros del equipo del cliente sobre las actividades de prueba.
- Dado que soy un administrador de pruebas ágiles con experiencia, cuando haya leído este libro, sabré cómo se está llevando a cabo la adopción generalizada de Agile y entenderé el contexto de trabajo de los evaluadores de otras organizaciones cuando soliciten trabajos en mi equipo. (Nota: esta prueba de aceptación no forma parte de esta versión, pero creemos que algunos de los ejemplos e historias del libro ayudarán a lograrlo).

Cómo leer este libro

Aunque hemos organizado este libro de la manera que creemos que fluye mejor, no es necesario comenzar con el Capítulo 1 y continuar. Al igual que con Agile Testing, puede comenzar con los temas que le resulten más útiles. Intentamos cubrir cada tema en detalle sólo una vez, pero debido a que muchos de estos conceptos, prácticas y principios están interrelacionados, encontrará que nos referimos a algunas ideas en más de un capítulo.

Parte I: Introducción

Lea esta parte para comprender dónde comenzaron las pruebas en equipos ágiles y cómo han evolucionado hasta convertirse en la piedra angular del desarrollo ágil y la entrega continua de productos. Parte del desarrollo ágil exitoso es la capacidad de una organización para aprender qué es lo más crítico para el éxito a largo plazo con pruebas ágiles.

- Capítulo 1, “Cómo han evolucionado las pruebas ágiles”
- Capítulo 2, “La importancia de la cultura organizacional”

Parte II: Aprender para realizar mejores pruebas

Tanto la tecnología como el arte de realizar pruebas evolucionan continuamente y las líneas entre las diferentes disciplinas se vuelven más borrosas. Incluso los practicantes experimentados tienen que seguir mejorando sus habilidades. Esta parte incluye ejemplos de lo que los evaluadores y otras disciplinas, como el análisis empresarial y la codificación, necesitan saber para enfrentar desafíos de prueba más difíciles. Explicamos los beneficios de generalizar a los especialistas y enumeramos algunas de las habilidades de pensamiento intangibles y habilidades de prueba técnicas específicas que ayudan a los evaluadores y a los equipos a mejorar. En los siguientes capítulos se tratan diferentes aspectos de qué y cómo aprender:

- Capítulo 3, “Funciones y competencias”
- Capítulo 4, “Habilidades de pensamiento para las pruebas”
- Capítulo 5, “Conciencia técnica”
- Capítulo 6, “Cómo aprender”

Parte III: Planificación, para no olvidar el panorama general

Planificar “lo suficiente” es un acto de equilibrio. Si bien necesitamos trabajar en pequeños incrementos, debemos estar atentos al conjunto de funciones más amplio y a todo el sistema. Esta parte cubre diferentes aspectos de la planificación de pruebas, desde el nivel de lanzamiento hasta el nivel de tarea. También explora diferentes modelos, como los cuadrantes de pruebas ágiles y algunas de las adaptaciones que la gente ha sugerido.

- Capítulo 7, “Niveles de precisión para la planificación”
- Capítulo 8, “Uso de modelos para ayudar a planificar”

Parte IV: Prueba del valor empresarial

Si, como tantos equipos ágiles, entrega un código sólido de manera oportuna y descubre que, después de todo, no es lo que los clientes querían, la información de esta parte le será de ayuda. Cubrimos herramientas y prácticas, en particular aquellas de la profesión de análisis empresarial ágil, para ayudarle a probar ideas y suposiciones con antelación y garantizar que todos sepan qué ofrecer. Nosotros

abordar otras disciplinas superpuestas y mentalidades en expansión. Esta es un área grande, por lo que hay varios capítulos:

- Capítulo 9, “¿Estamos construyendo lo correcto?”
- Capítulo 10, “La mentalidad del evaluador en expansión: ¿es este mi trabajo?”
- Capítulo 11, “Obtención de ejemplos”

Parte V: Pruebas de investigación

Los programadores han entregado algo de código para probar. ¿Por dónde empiezas? Si usted o su equipo carecen de experiencia con pruebas exploratorias, encontrarán ayuda aquí. Describimos varias técnicas de prueba exploratorias, como el uso de personas y recorridos para ayudar a generar ideas de cartas de prueba, así como la gestión de cartas con gestión de pruebas basada en sesiones y gestión de pruebas basada en subprocesos.

Junto con todas esas diferentes formas de realizar pruebas exploratorias, analizamos otras formas de verificar que el código entregado satisfaga una amplia gama de necesidades comerciales y de usuarios. Esta parte cubre formas de mitigar riesgos y generar información útil en varios tipos diferentes de pruebas que presentan desafíos para los equipos ágiles. Los capítulos sobre pruebas de investigación son

- Capítulo 12, “Pruebas exploratorias”
- Capítulo 13, “Otros tipos de pruebas”

Parte VI: Automatización de pruebas

Vemos que cada vez más equipos encuentran formas de tener éxito con la automatización de pruebas. Sin embargo, para muchos equipos, las pruebas automatizadas producen fallas esporádicas que resultan costosas de investigar. El tiempo (costo) invertido en cada falla puede ser mayor de lo que vale la prueba. Existen muchos inconvenientes en la automatización de pruebas. En esta parte damos ejemplos de formas de hacer visible la deuda técnica en las pruebas. Analizamos diferentes formas de utilizar la pirámide de pruebas ágiles de forma eficaz para ayudarle a pensar en cómo planificar su automatización. Hemos introducido algunos modelos piramidales alternativos

abordar la automatización desde diferentes perspectivas. Aprenderá formas de diseñar pruebas automatizadas para una confiabilidad óptima y facilidad de mantenimiento. Esta parte también incluye ejemplos de ampliación de la automatización de pruebas en una gran empresa.

Los capítulos de la Parte VI son

- Capítulo 14, “Deuda técnica en las pruebas”
- Capítulo 15, “Pirámides de la automatización”
- Capítulo 16, “Patrones y enfoques de diseño de automatización de pruebas”
- Capítulo 17, “Selección de soluciones de automatización de pruebas”

Parte VII: ¿Cuál es su contexto?

Su enfoque hacia las pruebas ágiles dependerá naturalmente de su contexto.

¿Trabaja con sistemas empresariales grandes? Tal vez tenga la tarea reciente de probar aplicaciones móviles o software integrado. Quizás su equipo tenga el desafío de encontrar buenas formas de probar datos que ayuden a las empresas a tomar decisiones. ¿Se ha preguntado qué tan ágil puede funcionar al probar software regulado? Finalmente, analizamos las sinergias entre las pruebas y el movimiento DevOps. Los capítulos de esta parte cubren una variedad de áreas, por lo que hemos incluido una serie de historias de personas que actualmente trabajan en esas situaciones. Es posible que algunos de estos capítulos no se apliquen a su entorno de trabajo actual, pero mañana, ¿quién sabe?

- Capítulo 18, “Pruebas ágiles en la empresa”
- Capítulo 19, “Pruebas ágiles en equipos distribuidos”
- Capítulo 20, “Pruebas ágiles para sistemas móviles e integrados”
- Capítulo 21, “Pruebas ágiles en entornos regulados”
- Capítulo 22, “Pruebas ágiles para almacenes de datos y empresas Sistemas de Inteligencia”
- Capítulo 23, “Pruebas y DevOps”

Parte VIII: Pruebas ágiles en la práctica

Concluimos el libro con una mirada a cómo los equipos pueden visualizar la calidad y las pruebas, y un resumen de prácticas de pruebas ágiles que le darán a su equipo

confianza al tomar decisiones de liberación. Crear una visión compartida para su equipo es fundamental para el éxito y compartimos un modelo para ayudar a llevar las actividades de prueba a todo el equipo. Si se siente un poco abrumado en este momento y no está seguro de por dónde empezar, lea estos capítulos primero:

- Capítulo 24, “Visualice sus pruebas”
- Capítulo 25, “Juntando todo”

El libro también incluye dos apéndices: Apéndice A, “Objetos de página en la práctica: ejemplos” y Apéndice B, “Iniciadores de provocación”.

Otros elementos

Dado que los equipos utilizan una variedad tan amplia de prácticas y enfoques ágiles, hemos intentado mantener nuestra terminología lo más genérica posible. Para asegurarnos de que tengamos un lenguaje común con usted, hemos incluido un glosario de términos que utilizamos.

Encontrará íconos en los márgenes de todo el libro donde nos gustaría llamar su atención sobre una práctica específica. Encontrará los seis íconos en el Capítulo 1, “Cómo han evolucionado las pruebas ágiles” y en el Capítulo 25, “Juntando todo”. Un ejemplo del ícono de aprendizaje se puede ver al lado del siguiente párrafo.



Esperamos que desee aprender más sobre algunas de las prácticas, técnicas y herramientas que cubrimos. Consulte la bibliografía para obtener referencias a libros, sitios web, artículos y blogs. Lo hemos ordenado por partes para que puedas encontrar más información fácilmente cuando leas. Las fuentes que se mencionan directamente en el libro se enumeran alfabéticamente en la lista de referencias para facilitar su búsqueda.

La descripción general del mapa mental de Agile Testing se incluye en el sitio web del libro, www.agiletester.com, para que pueda tener una idea de lo que se cubrió allí si aún no lo ha leído.

¡Experimento!

Linda Rising nos animó hace años a intentar pequeños experimentos, evaluar los resultados y seguir iterando para solucionar los problemas y alcanzar las metas. Si lee algo en este libro que le parece útil para usted o su equipo, pruébelo una o dos iteraciones. Utilice sus retrospectivas para ver si está ayudando y modifíquelas según sea necesario. Si no funciona, aprendiste algo y puedes probar algo diferente.

Esperamos que encuentre muchos experimentos para probar en estas páginas.

Expresiones de gratitud

Este libro ha sido un esfuerzo de grupo. Conozca a todos los maravillosos practicantes que compartieron sus historias en las barras laterales de “Acerca de los colaboradores”. Muchas son historias de éxito, algunas describen lecciones aprendidas de la manera más difícil, pero sabemos que todas beneficiarán a usted, el lector.

Estamos muy agradecidos a Jennifer Sinclair por sus maravillosas ilustraciones. A ella se le ocurrieron ideas tan creativas para ayudarnos a transmitir algunos conceptos importantes.

Hicimos referencia a las ideas de muchas otras personas que tomaron ideas de Agile Testing, las adaptaron para satisfacer sus necesidades y estaban dispuestas a compartir las con el mundo. Gracias.

Nuestros incansables críticos nos ayudaron a darle forma al libro y a cubrir los temas adecuados. Estamos especialmente agradecidos a Mike Talks, Bernice Niel Ruhland y Sherry Heinze, quienes repasaron cada capítulo, en algunos casos varias veces. Gracias a Augusto Evangelisti, Gojko Adzic, Adam Knight, Steve Rogalsky, Aldo Rall, Sharon Robson, James Lyndsay, JeanAnn Harrison, Ken Rubin, Geoff Meyer, Adam Yuret y Mike Cohn por sus valiosos comentarios. Cada uno de nuestros colaboradores de historias también ayudó a revisar los capítulos que incluían sus historias.

Un agradecimiento especial a nuestros revisores técnicos, cuyos comentarios sobre nuestro borrador final fueron de gran ayuda: Tom Poppendieck, Liz Keogh, Markus Gärtner y George Dinwiddie.

Gracias, Christopher Guzikowski, por hacer posible este libro en primer lugar, y a Olivia Basegio, por responder mil preguntas y mantenernos organizados. Estamos agradecidos con nuestro desarrollo.

editor, Chris Zahn, Kesel Wilson, nuestro editor de producción, y a Barbara Wood por realizar la edición final. Fue maravilloso volver a trabajar con el equipo de Addison-Wesley.

Gracias a una nueva graduada de inglés, Bea Paiko, que hizo una edición preliminar que nos ayudó a escribir de forma un poco más limpia. Gracias, Mike Cohn, por permitirnos ser parte de un gran grupo de autores ágiles. Gracias a Ellen Gottesdiener y Mary Gorman por compartir con nosotros algunos de sus consejos sobre el proceso de escritura de libros; Esos nos ayudaron a organizar el libro más fácilmente.

Ambos somos afortunados de haber trabajado junto a tantas personas increíbles a lo largo de los años que nos enseñaron mucho sobre cómo ofrecer software valioso. Son demasiado numerosos para nombrarlos aquí, pero nos referimos a algunos en el texto y la bibliografía. Tenemos suerte de ser parte de una generosa comunidad de software global.

Finalmente, un agradecimiento a nuestra maravillosa y solidaria familia y amigos.

Agradecimiento personal de Janet:

Gracias a mi esposo, Jack, por todos los contratos revisados, las cenas preparadas y los recados realizados, y por permitirme trabajar hasta altas horas de la noche. Sé que prácticamente te ignoré de nuevo durante el tiempo que me llevó escribir este libro. Tu aliento me mantuvo adelante.

Lisa, nos complementamos en nuestros estilos de escritura y creo que eso es lo que nos convierte en un gran equipo. Gracias por brindarnos un excelente lugar para revisar nuestro primer borrador y la oportunidad de conocer a sus burros.

Y, por último, quiero reconocer el poder de la capacidad inalámbrica y de Internet. Mientras escribía este libro, viajé al norte, a Helsinki, Finlandia, y acampé en Grande Prairie, Canadá. Estaba al sur de Johannesburgo, en Sudáfrica, y acampé en Botswana y Zimbabwe, escribiendo mientras observaba leones y elefantes. También estuve en Australia, aunque no probé la conexión inalámbrica en el interior de allí. Incluso llegué a alcanzar los 3.000 metros (~10.000 pies) en Perú. Sólo había unos pocos lugares donde no podía conectarme en absoluto. Este escrito fue verdaderamente un esfuerzo de equipo distribuido.

Agradecimiento personal

de Lisa: Gracias a mi marido, Bob Downing, sin cuyo apoyo nunca podría escribir ni presentar nada. Nunca imaginó que algún día estaría afuera limpiando el corral de un burro mientras yo trabajaba como esclavo en un teclado. Nos ha mantenido a mí y a todas nuestras mascotas bien alimentadas y amadas. ¡Sigues siendo la rodilla de la abeja, querida!

Gracias, Janet, por mantenernos encaminados y hacer tanto trabajo pesado para organizarnos, escribir y crear tantas imágenes geniales. Trabajar con usted siempre es un privilegio, una experiencia de aprendizaje y mucha diversión. ¡Y también agradezco al esposo de Janet, Jack, por su ayuda con la letra pequeña y por permitirle a Janet compartir toda esta diversión y arduo trabajo conmigo!

Si los lectores aprenden una fracción de lo que yo aprendí mientras escribía este libro, ¡lo consideraré un éxito!



Esta página se dejó en blanco intencionalmente.

Sobre los autores

Janet Gregory es entrenadora de pruebas ágiles y consultora de procesos en DragonFire Inc. Es coautora, junto con Lisa Crispin, de Agile Testing: A Practical Guide for Testers and Agile Teams (Addison-Wesley, 2009) y More Agile Testing: Learning Journeys for the Whole Team. (Addison-Wesley, 2015). También colabora con 97 cosas que todo programador debería saber. Janet se especializa en mostrar a equipos ágiles cómo los evaluadores pueden agregar valor en áreas más allá de criticar el producto, por ejemplo, guiar el desarrollo con pruebas orientadas al negocio. Janet trabaja con equipos para la transición al desarrollo ágil e imparte cursos y tutoriales de pruebas ágiles en todo el mundo. Contribuye con artículos para publicaciones como Better Software, Software Test & Performance Magazine y Agile Journal y le gusta compartir sus experiencias en conferencias y reuniones de grupos de usuarios en todo el mundo. Para obtener más información sobre el trabajo de Janet y su blog, visite www.janetgregory.ca. También puedes seguirla en Twitter: @janetgregoryca.

Lisa Crispin es coautora con Janet Gregory de Agile Testing: A Practical Guide for Testers and Agile Teams (Addison-Wesley, 2009) y More Agile Testing: Learning Journeys for the Whole Team (Addison-Wesley, 2015); también es coautora de Tip House of Extreme Testing (Addison-Wesley, 2002) y colaboradora de Experiences of Test Automation de Dorothy Graham y Mark Fewster (Addison-Wesley, 2011) y Beautiful Testing (O'Reilly, 2009). Lisa fue honrada por sus compañeros que la votaron como la persona profesional de pruebas ágiles más influyente en Agile Testing Days 2012. Lisa disfruta trabajar como tester con un increíble equipo ágil. Comparte sus experiencias escribiendo, presentando, enseñando y participando en comunidades de pruebas ágiles en todo el mundo. Para obtener más información sobre el trabajo de Lisa, visite www.lisacrispin.com y siga a @lisacrispin en Twitter.

Esta página se dejó en blanco intencionalmente.

Acerca de los contribuyentes

Gojko Adzic es un consultor de entrega de software estratégico que trabaja con equipos ambiciosos para mejorar la calidad de sus productos y procesos de software. Se especializa en mejora de la calidad ágil y eficiente, en particular pruebas ágiles, especificación por ejemplo y desarrollo impulsado por el comportamiento. Gojko es autor de Especificación por ejemplo (Adzic, 2011), ganador del premio Jolt 2012; Mapeo de Impacto (Adzic, 2012); Reducir la brecha de comunicación (Adzic, 2009); un blog galardonado; y otros libros relacionados con pruebas y agilidad. En 2011, sus pares lo votaron como el profesional de pruebas ágiles más influyente.

A Matt Barcomb le apasiona cultivar organizaciones sosteniblemente adaptativas, le gusta estar al aire libre, le encantan los juegos de palabras y le encanta guiar a las empresas hacia culturas autoorganizadas más gratificantes y productivas. Matt ha hecho esto en sus funciones como ejecutivo de desarrollo de productos, consultor de diseño organizacional, entrenador ágil, gerente de equipos de desarrollo y programador. Él cree que la evolución de las empresas hacia sistemas humanísticos centrados en el cliente es el mayor desafío que enfrentan las empresas en la actualidad. Como tal, ha dedicado una cantidad excesiva de su tiempo y energía a encontrar formas de ayudar a las organizaciones a convertirse en mejores lugares para trabajar.

Susan Bligh ha trabajado en la industria de TI durante diecisiete años y siente entusiasmo por los procesos comerciales y la excelencia operativa mediante el uso de la tecnología. Actualmente es analista de negocios líder en una empresa de petróleo y gas en Calgary, Alberta, Canadá. Susan ha liderado esfuerzos de analistas de negocios para proyectos a gran escala que afectan a muchas disciplinas y en amplias geografías. Anteriormente trabajó en desarrollo de software, capacitación y gestión de clientes, así como en administración de bases de datos.

Tiene una licenciatura en informática con especialización en administración de la Universidad de Calgary.

Paul Carvalho se dedica a ayudar a los equipos de desarrollo de software a ofrecer altos niveles de calidad con confianza. Inspira a equipos colaborativos, ágiles y llenos de pruebas con un enfoque holístico de la calidad. Paul ha dedicado más de veinte años a aprender y aplicar enfoques, modelos, métodos, técnicas y herramientas de prueba para iluminar a los tomadores de decisiones.

Transmite ese conocimiento a personas y organizaciones a través de coaching, consultoría, capacitación, redacción y conferencias a nivel internacional.

A Paul le apasiona comprender los ecosistemas humanos para ofrecer excelentes productos que satisfagan y deleiten a los clientes, lo que considera que encaja naturalmente con la comunidad ágil. Conéctese con él a través de STAQS.com.

Augusto Evangelisti es un profesional del desarrollo de software, bloguero y futbolista con un gran interés en las personas, la calidad del software y las prácticas ágiles y lean. Le gusta cocinar, comer, aprender y ayudar a equipos ágiles a superar las expectativas de los clientes mientras se divierten.

David Evans es un consultor, entrenador y formador ágil con más de veinticinco años de experiencia en TI. Líder intelectual en el campo de la calidad ágil, ha brindado capacitación y consultoría a clientes en el Reino Unido, Estados Unidos, Irlanda, Suecia, Alemania, Francia, Australia, Israel, Sudáfrica y Singapur. David, orador habitual en eventos y conferencias en toda Europa, fue elegido Mejor Orador Principal en Agile Testing Days 2013. También ha publicado varios artículos en revistas internacionales de TI. Actualmente vive y trabaja en el Reino Unido, donde es socio, junto con Gojko Adzic, de Neuri Consulting LLP. Puede comunicarse con él en david.evans@neuri.co.uk por correo electrónico y [@DavidEvans66](https://twitter.com/#!/DavidEvans66) por correo electrónico.

Gorjeo.

Kareem Fazal es ingeniero senior de desarrollo de software de plataforma en Dell Enterprise Solutions Group. Tiene más de siete años de experiencia en la industria del firmware trabajando en automatización y desarrollo de productos. Se unió a Dell en 2010 como líder de pruebas y luego pasó a la organización de desarrollo de firmware para liderar las estrategias de automatización y el desarrollo de productos.

Benjamin Frempong, ingeniero de pruebas senior del Dell Enterprise Solutions Group, tiene más de diez años de experiencia liderando programas de control de calidad de hardware y software en las organizaciones de clientes y empresas de Dell. Actualmente se centra en ayudar a los equipos a implementar estrategias de automatización de pruebas eficientes y sostenibles.

Chris George ha sido probador de software y formulador de preguntas desde 1996, trabajando para una variedad de empresas del Reino Unido creando herramientas para el desarrollo de bases de datos, generación de informes de datos y transmisión de contenido digital. Durante ese tiempo, ha explorado, investigado, innovado, inventado, planificado, automatizado, estresado, informado, cargado, codificado y estimado en equipos de software tanto tradicionales (en cascada) como ágiles. También presenta en conferencias de software sobre temas de pruebas y escribe un blog, www.mostly-testing.co.uk.

Mary Gorman, líder en análisis y requisitos comerciales, es vicepresidenta de calidad y entrega en EBG Consulting. Mary entrena equipos de productos, facilita talleres de descubrimiento y capacita a las partes interesadas en prácticas colaborativas esenciales para definir productos de alto valor. Habla y escribe para las comunidades ágiles, de análisis empresarial y de gestión de proyectos. Mary, una profesional certificada en análisis de negocios, ayudó a desarrollar la Guía del IIBA para el conjunto de conocimientos sobre análisis de negocios y el examen de certificación. También formó parte del grupo de trabajo que creó la definición de funciones de Profesional en Análisis de Negocios de PMI. Mary es coautora de Discover to Deliver (Gottesdiener y Gorman, 2012).

Ellen Gottesdiener, fundadora y directora de EBG Consulting, ayuda a las personas a descubrir y ofrecer los productos de software adecuados en el momento adecuado. Ellen es una líder reconocida internacionalmente en prácticas ágiles de gestión de proyectos y productos, visión y hoja de ruta de productos, análisis y requisitos comerciales, retrospectivas y colaboración. Como facilitadora, asesora y formadora experta, Ellen trabaja con clientes de todo el mundo y habla con frecuencia en una amplia gama de conferencias de la industria. Es coautora de Discover to Deliver (Gottesdiener y Gorman, 2012) y autora de otros dos libros aclamados: Requisitos por colaboración (Gottesdiener, 2002) y The Software Requisitos Memory Jogger. (Dios siervo, 2005).

xxxviii Acerca de los contribuyentes

Jon Hagar es un consultor independiente que trabaja en pruebas de integridad, verificación y validación de productos de software en Grand Software Testing. Jon publica regularmente, incluido un libro sobre pruebas de software integrado/móvil: *Software Test Attacks to Break Mobile and Embedded Devices*. (Agar, 2013). Sus intereses incluyen lo ágil, lo móvil, lo integrado, el control de calidad, el desarrollo de habilidades y el aprendizaje permanente.

Parimala Hariprasad pasó su juventud estudiando a las personas y la filosofía. Cuando se puso a trabajar, pudo utilizar esos conocimientos para crear evaluadores increíbles. Ha trabajado como tester durante más de diez años en ámbitos como la gestión de relaciones con los clientes, la seguridad, el comercio electrónico y la atención sanitaria. Su especialidad es entrenar y crear grandes equipos, equipos que finalmente la despidieron porque ya no la necesitaban. Ha experimentado la transición de la web a los dispositivos móviles y enfatiza la necesidad del pensamiento de diseño en las pruebas. Con frecuencia despotrica en su blog, Curious Tester (<http://curioustester.blogspot.com>). Ella tuitea en @CuriousTester y se la puede encontrar en LinkedIn en <http://in.linkedin.com/in/parimalahariprasad>.

JeanAnn Harrison ha estado en el campo de pruebas de software y control de calidad durante más de quince años, incluidos siete años trabajando dentro de un entorno regulatorio y ocho años realizando pruebas de software móvil. Su nicho son las pruebas de integración de sistemas con un enfoque en entornos de sistemas de múltiples niveles que involucran cliente/servidor, aplicaciones web y aplicaciones de software independientes. JeanAnn es oradora habitual en muchas conferencias sobre pruebas de software y otros eventos, y es facilitadora de Weekend Testing Americas. Siempre busca inspirarse en compañeros probadores de toda la comunidad de pruebas de software y continúa combinando sus experiencias prácticas con la interacción en foros de prueba y calidad del software, asistiendo a clases de capacitación y permaneciendo activa en los sitios de redes sociales.

Mike Heinrich ha trabajado como evaluador durante más de una década en logística, banca, telecomunicaciones, viajes y servicios públicos. A lo largo de su carrera, Mike se ha centrado en las pruebas de integración y datos. Su pasión por los datos y por ofrecer valor al cliente le ha brindado la oportunidad de realizar presentaciones ante varias organizaciones norteamericanas sobre pruebas y almacenamiento de datos ágiles. En su tiempo libre, Mike disfruta viajar por el mundo, jugar voleibol y entrenar baloncesto.

Sherry Heinze es estratega de pruebas, evaluadora, analista de control de calidad y capacitadora con una amplia experiencia en análisis, diseño, pruebas, capacitación, implementación, documentación y soporte al usuario. Durante los últimos 17 años, Sherry se ha centrado en las pruebas desde el análisis y el diseño en adelante, a veces en equipos multifuncionales, a veces con equipos de evaluadores, a veces sola. Sherry tiene una amplia experiencia trabajando en diversas metodologías tanto con usuarios como con personal técnico para identificar y probar requisitos, diseñar, crear, probar, implementar y respaldar sistemas.

Matthew Heusser ha pasado su vida adulta desarrollando, probando y gestionando proyectos de software. En el camino, Matt se desempeñó como editor colaborador de la revista Software Test & Quality Assurance , organizó el taller patrocinado por Agile Alliance sobre deuda técnica y formó parte de la junta directiva de la Association for Software Testing. Quizás mejor conocido por sus escritos, Matt fue el editor principal de Cómo reducir el costo de las pruebas de software (Heusser, 2011) y actualmente se desempeña como editor gerente de Stickyminds.com. Como consultor gerente de Excelon Development, Matt administra las cuentas clave de la empresa al mismo tiempo que realiza consultoría y redacción. Puede leer más sobre Matt en el sitio web de Excelon, www.xndev.com, o seguirlo en Twitter: @heusser.

Michael Hüttermann, un campeón de Java, es un ingeniero de entrega independiente y experto en DevOps, entrega continua y gestión de control de fuente/gestión del ciclo de vida de las aplicaciones. Es autor de Agile ALM (Hüttermann, 2011a) y DevOps for Developers (Hütter-mann, 2012). Para obtener más información, consulte <http://huettermann.net>.

Griffin Jones, un evaluador, formador y asesor ágil, brinda consultoría sobre pruebas de software basadas en el contexto y cumplimiento normativo a empresas de industrias reguladas y no reguladas. Recientemente, fue director de calidad y cumplimiento normativo en iCardiac Technologies, que brinda servicios de laboratorio básicos para la industria farmacéutica para evaluar la seguridad cardíaca de posibles nuevos medicamentos. Griffin era responsable de todos los asuntos relacionados con la calidad y el cumplimiento normativo de la FDA, incluida la presentación de los resultados de verificación y validación (pruebas) a auditores regulatorios externos. Es anfitrión del Taller sobre pruebas de software reguladas (WREST) y miembro de ASQ, AST, ISST y RAPS.

Stephan Kämper estudió física, escribió su tesis sobre holografía y luego se unió al grupo de oceanografía de la Universidad de Bremen. En 2001 se inició en el desarrollo de software uniéndose al equipo de pruebas de un sistema de base de datos orientado a objetos. Nunca abandonó las pruebas de software y se especializó en pruebas de software automatizadas y métodos ágiles. Trabajó en temas tan diversos como sistemas de navegación de precisión, plataformas de pago, sistemas de atención médica, telecomunicaciones y redes sociales.

Trabajar en estos diferentes campos le ayudó a reconocer patrones comunes, que encontró útiles en las pruebas de software. Sus idiomas son (en orden alfabético) inglés, alemán y Ruby. Síguelo en Twitter en @S_2K y visita su sitio web: www.seasidetesting.com.

Trish Khoo ha trabajado en ingeniería y gestión de pruebas para empresas como Google, Campaign Monitor y Microsoft. Mantiene un blog en www.trishkhoo.com y un podcast en testcast.net, le gusta hablar en conferencias y escribe artículos para publicaciones técnicas. Cuando no está haciendo todo eso, está ocupada viajando por el mundo, dibujando robots o tal vez simplemente durmiendo hasta el mediodía. Trish obtuvo una licenciatura en tecnología de la información de la Universidad de Queensland, donde se graduó con honores.

Adam Knight ha estado probando software de análisis y almacenamiento de datos durante diez años, siete de los cuales los pasó trabajando en un equipo ágil. Adam es un entusiasta exponente de los enfoques de pruebas exploratorias respaldados por el uso exigente de la automatización. Es un gran creyente en la creación de equipos polivalentes basados en conjuntos de habilidades individuales ricos y únicos. En su empleador actual, RainStor, Adam ha supervisado las pruebas y el soporte técnico de un sistema de almacenamiento de datos a gran escala desde su lanzamiento inicial hasta su adopción exitosa en algunas de las empresas de servicios financieros y de telecomunicaciones más grandes del mundo. Escribe en www.a-sisyphean-task.com.

Cory Maksymchuk es un desarrollador de software apasionado por los procesos ágiles y el desarrollo de software eficiente. Ha pasado la mayor parte de los últimos 12 años trabajando en la pila de Java como parte de grandes iniciativas de desarrollo de software. Su verdadera pasión en la vida es encontrar soluciones elegantes a problemas difíciles y realmente le entusiasma ver cómo las grandes ideas cobran vida.

Drew McKinney es diseñador de experiencias de usuario en Pivotal Tracker y Pivotal Labs. Antes de Pivotal, Drew dirigió Bloomingsoft, una consultoría de desarrollo y diseño móvil. En el pasado, Drew ha trabajado con empresas como Disney Animation Studios, Audi USA, Cook Medical y Deloitte Consulting. Es un miembro activo de la comunidad de diseño y ha hablado sobre diseño en numerosos eventos tecnológicos de Indiana y Colorado.

Geoff Meyer, arquitecto de pruebas de Dell Enterprise Solutions Group, tiene más de veintiocho años de experiencia en la industria del software como desarrollador, gerente, analista de negocios y arquitecto de pruebas. Desde 2010, un enfoque secundario de Geoff ha sido fomentar las prácticas de prueba y desarrollo de software basadas en ágiles de más de ochocientos ingenieros de desarrollo, pruebas y experiencia de usuario en cuatro centros de diseño globales. Geoff es un miembro activo y colaborador de la comunidad Agile Austin.

Jeff "Cheezy" Morgan, director de tecnología y cofundador de LeanDog, ha estado impartiendo clases y entrenando equipos sobre técnicas ágiles y lean desde principios de 2004. La mayor parte de su trabajo se ha centrado en las prácticas de ingeniería utilizadas por desarrolladores y evaluadores. Durante los últimos años, ha experimentado un gran éxito y reconocimiento por su trabajo centrado en ayudar a los equipos a adoptar el desarrollo basado en pruebas de aceptación utilizando Cucumber. Es autor de varias gemas Ruby populares utilizadas por los probadores de software y es autor del libro Cucumber & Cheese—A Taller de probadores (Morgan, 2013).

Claire Moss se convirtió en la primera graduada empresarial en matemáticas discretas del Instituto de Tecnología de Georgia en 2003 e inmediatamente se lanzó a las pruebas de software. Ha seguido esta vocación desde entonces, trabajando con equipos de productos ágiles como profesora de pruebas, asesora de pruebas unitarias y de integración, evaluadora exploratoria y automatizadora de pruebas. Aunque siempre volverá a hacer álbumes de recortes, su pasatiempo dominante en los últimos años ha sido escribir, hablar y ser nerd sobre las pruebas. Claire siempre ha tenido pasión por escribir y continúa usando sus poderes malignos para hacer el bien en su trabajo y en su blog en <http://aclairefaction.com>.

Aldo Rall comenzó a realizar pruebas como programador junior al comienzo de la burbuja Y2K. Desde entonces, trabajando en Sudáfrica y el Reino Unido, adquirió experiencia práctica en pruebas en una gran cantidad de títulos, tareas,

y proyectos. Su mayor pasión radica en la dimensión de las "personas" y cómo eso se traduce en productos, equipos y evaluadores exitosos. A través de esta experiencia, disfruta de oportunidades para desarrollar, crecer y hacer madurar las pruebas, los evaluadores y los equipos.

Sharon Robson es la líder de práctica de pruebas de software para Software Education. Sharon, una tester apasionada y una formadora nata, ofrece y desarrolla cursos en todos los niveles de pruebas de software, desde introductorio hasta avanzado. Sharon también se enfoca en lo ágil y dedica una cantidad significativa de su tiempo a trabajar con equipos (capacitación, entrenamiento y tutoría) para ayudarlos en sus transiciones. Actualmente, Sharon investiga y escribe sobre enfoques de pruebas ágiles en diversos ámbitos empresariales. Presenta en conferencias locales e internacionales y contribuye a la comunidad ágil y de pruebas a través de blogs, tweets, participación en conferencias y tutoría.

Steve Rogalsky, reconociendo que la cultura, la gestión y los procesos de desarrollo de software pueden ser frustrantes e inhibidores, ha invertido significativamente en encontrar formas de superar y contrarrestar esos efectos. Ha descubierto que valorar la simplicidad, el respeto por las personas, la mejora continua y los ciclos cortos de retroalimentación son herramientas poderosas para abordar estas deficiencias. Dado que el desarrollo de software no se hace cargo de esas frustraciones, también ha estado traduciendo lo que ha aprendido a otras áreas de la organización, la vida familiar, los grupos comunitarios y el coaching. Habla regularmente en conferencias en Canadá y Estados Unidos, ha aparecido en InfoQ, cofundó Winnipeg Agile User Group y trabaja en Protegra. Puede leer más sobre lo que aprendió en <http://WinnipegAgilist.Blogspot.com>.

Bernice Niel Ruhland, con más de veinte años de experiencia profesional que abarca una variedad de disciplinas técnicas, actualmente se desempeña como directora de programas de gestión de calidad para ValueCentric LLC. Aplicando sus competencias en programación, pruebas, evaluación e implementación de software, Bernice dirige el departamento de pruebas de software de la empresa. Como fuerza impulsora detrás de los programas de calidad para toda la empresa de ValueCentric, se basa en prácticas en teorías y metodologías ágiles y basadas en el contexto para guiar los esfuerzos fundamentales. Cuando no está trabajando, mantiene un exitoso blog, www.TheTestersEdge.com, una colección de sus observaciones relacionadas con una variedad de temas técnicos que incluyen pruebas de software, liderazgo y desarrollo profesional.

Huib Schoots es tester, consultor y amante de la gente. Comparte su pasión por las pruebas a través del asesoramiento, la capacitación y las presentaciones sobre una variedad de temas de prueba. Curioso y apasionado, es un evaluador ágil y contextual que intenta leer todo lo publicado sobre pruebas de software. También es miembro de TestNet, AST e ISST; cinturón negro en la Escuela de Pruebas de Software Miagi-Do; y coautor de un libro sobre el futuro de las pruebas de software. Huib mantiene un blog en www.magnificant.com y tuitea como @huibschoots.

Paul Shannon y Chris O'Dell se unieron al equipo de 7digital en 2010 y 2011 respectivamente, ambos comenzando en el equipo responsable de la API de 7digital. Trabajaron para mejorar la calidad de las pruebas en la API y Chris ahora lidera ese equipo, concentrándose en mejorar la plataforma para la entrega continua, la resiliencia y el escalamiento. Paul trabaja en todos los equipos del equipo de tecnología de 7digital que están orientados a la mejora continua y prácticas de desarrollo de software impulsadas por la calidad.

El equipo sigue un enfoque de prueba primero con un flujo de trabajo visible y altamente colaborativo, y a todos les encanta la tecnología y las pruebas.

Jennifer Sinclair ha sido artista, instructora de arte y educadora desde 1995. Durante ese tiempo, ha vivido en Canadá, Japón y Estados Unidos y ha trabajado para mejorar la exploración del arte para niños y adultos de todas las edades y habilidades. Ha diseñado e ilustrado imágenes para el Consejo de Educación Infantil de la Asociación de Maestros de Alberta y el Consejo de Educación de Alberta en Canadá. Actualmente está trabajando en el desarrollo de lecciones de arte que se integren fácilmente en las materias básicas de la educación primaria. Como ama de casa, artista independiente e instructora de arte voluntaria, le apasiona seguir desarrollando sus habilidades y conocimientos y compartirlos con la mayor cantidad de personas posible. Puede comunicarse con ella en jvaagesinclair@live.com.

Toby Sinclair se unió al negocio de pruebas de software como graduado universitario en 2007 y no ha mirado atrás. Ha trabajado para varios software.

consultorías de pruebas en el Reino Unido y actualmente está trabajando con JP Morgan para mejorar sus capacidades de prueba para respaldar la transición a la metodología ágil. Toby es un miembro activo de la comunidad de pruebas y se le puede encontrar en Twitter: [@TobyTheTester](https://twitter.com/TobyTheTester).

Tony Sweets es un veterano con 20 años de experiencia en la industria del software y actualmente trabaja como arquitecto de tecnología de la información. Durante los últimos 13 años

Ha estado trabajando en aplicaciones web empresariales Java en el sector financiero. Tony posee una amplia gama de habilidades, pero le gusta trabajar principalmente en aplicaciones Java y las herramientas que mejoran el proceso de desarrollo. Tony tiene una licenciatura en informática de la Universidad de Wyoming.

Mike Talks tenía 26 años cuando "probó" la TI por primera vez como carrera. Antes de eso, había sido profesor, investigador científico y analista de datos, y a sus padres les preocupaba que nunca "consiguiera un trabajo adecuado". Aunque originalmente era programador, fue en las pruebas donde floreció. Originalmente trabajó en proyectos militares en cascada largos y con muchos requisitos en el Reino Unido, pero desde que se mudó a Nueva Zelanda se encontró trabajando cada vez más en proyectos con compañías como Assurity, Kiwibank y Datacom, donde la entrega oportuna es un factor clave.

Eveliina Vuolli se desempeña actualmente como gerente de desarrollo operativo en Nokia Solutions and Networks. Ha trabajado con el equipo de desarrollo de I+D del sistema de gestión de redes durante 15 años, desempeñando diferentes tipos de roles en la organización global y multinacional: propietaria del proceso de integración y verificación, directora de proyectos y formadora en diversas áreas, y también coach. Además, ha estado involucrada en la transformación ágil en su propia área de producto.

Pete Walen se dedica al desarrollo de software desde hace más de veinticinco años. Ha trabajado en una variedad de roles, incluidos desarrollador, analista de negocios y gerente de proyectos. Es un contratista de consultoría independiente que trabaja con equipos de prueba durante períodos prolongados, entrenándolos y trabajando para mejorar sus técnicas y prácticas de prueba. Pete se describe a sí mismo como un "antropólogo y probador de software", lo que abarca el examen de cómo el software y las personas se relacionan e interactúan. Ha trabajado en una variedad de talleres utilizando una variedad de metodologías de desarrollo y ha adoptado una actitud de "hacer lo que tenga sentido" para la organización y el proyecto.

Mary Walshe ayuda a los equipos a ofrecer soluciones exitosas a los problemas comerciales y desempeña un papel importante en la lucha por una cultura kaizen en estos equipos. Mary fue la fuerza impulsora detrás de la introducción del desarrollo basado en pruebas de aceptación en su departamento. Ella ha estado

trabajando en la industria durante cuatro años y actualmente trabaja en un equipo en Paddy Power como tester ágil. Su equipo está utilizando kanban para ayudarles a medir sus experimentos y mejorar continuamente.

En su tiempo libre, Mary corre carreras de aventuras, anda en bicicleta de montaña y recientemente encontró un nuevo amor por el esquí.

Christin Wiedemann, tras finalizar su doctorado. Licenciado en física por la Universidad de Estocolmo en 2007, comenzó a trabajar como desarrollador de software. Christin pronto descubrió que las pruebas le parecían más desafiantes y creativas, y se unió a la empresa de pruebas AddQ Consulting. Allí, trabajó como evaluadora, líder de pruebas y formadora, impartiendo cursos sobre pruebas ágiles, diseño de pruebas y pruebas exploratorias. A finales de 2011, Christin se mudó a Vancouver, Canadá, y se unió a Professional Quality Assurance. En sus funciones de evaluadora, líder de pruebas, formadora y oradora, Christin utiliza su experiencia científica y sus habilidades pedagógicas para desarrollar continuamente sus propias habilidades y las de los demás.

Lynn Winterboer, con experiencia comprobada en una variedad de proyectos de datos y prácticas ágiles, enseña y entrena a equipos de inteligencia empresarial/almacenamiento de datos sobre cómo aplicar eficazmente principios y prácticas ágiles a su trabajo. Durante más de quince años, Lynn ha desempeñado numerosos puestos dentro del ámbito de análisis, inteligencia empresarial y almacenamiento de datos. Ella comprende muy bien el conjunto único de desafíos que enfrentan los equipos en esta área que desean beneficiarse del estilo incremental del desarrollo ágil; Lynn aprovecha su experiencia y formación para ayudar a ofrecer soluciones prácticas a sus clientes y estudiantes.

Puede comunicarse con Lynn en www.LynnWinterboer.com.

Cirilo Wortel es un evaluador y formador independiente de los Países Bajos. En 2006, Cirilo se involucró por primera vez en el desarrollo de software ágil. Ha trabajado con varias empresas empresariales, entrenando y ayudando a implementar la automatización de pruebas durante su adopción ágil. Cirilo fue coanfitrión, junto con Janet Gregory, de una clase magistral sobre pruebas ágiles durante varios años en los Países Bajos. Ha contribuido a la comunidad fundando la Federación de Probadores Ágiles, el grupo de usuarios de pruebas ágiles más grande de los Países Bajos, y es un orador frecuente en conferencias internacionales.

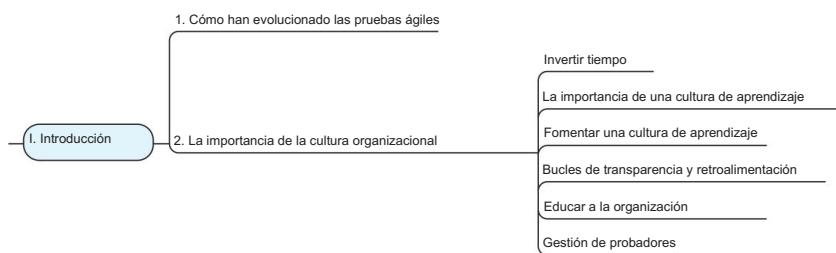
Con varios colegas de Xebia, Cirilo desarrolló Xebium, una herramienta de automatización para aplicaciones web.

Alexei Zheglov se dedica a descubrir y practicar nuevos métodos para gestionar y liderar la mejora del trabajo moderno, complejo e intensivo en conocimientos. Llegó a esto después de una larga carrera en ingeniería de software, durante la cual aprendió a ver y resolver muchos problemas en la entrega de software. Alexei presenta sus hallazgos con frecuencia en conferencias en Canadá y en el extranjero. Es reconocido como un Profesional de Coaching Kanban y un Entrenador Kanban Acreditado. Alexei vive en Waterloo, Ontario, Canadá. Su blog se puede encontrar en <http://connected-knowledge.com>.

Parte I

Introducción

En esta primera parte, repasamos una breve historia de cómo han cambiado las pruebas ágiles desde que trabajamos por primera vez en equipos ágiles. Introducimos algunos de los nuevos conceptos y entramos en más detalles sobre la importancia de una cultura organizacional de aprendizaje.



- Capítulo 1, “Cómo han evolucionado las pruebas ágiles”
- Capítulo 2, “La importancia de la cultura organizacional”

Esta página se dejó en blanco intencionalmente.

Capítulo 1

Cómo las pruebas ágiles Ha evolucionado

Cada uno de nosotros comenzó nuestra carrera "ágil" como probador solitario en un equipo de Programación Extrema (XP). En ese momento, ninguna de las publicaciones sobre XP mencionaba siquiera tener probadores en el equipo. Había dos roles: programador y cliente. Los clientes especificarían todas las pruebas de aceptación, los programadores las automatizarían y listo, ya está. Cuando asistíamos a conferencias de XP, normalmente éramos los únicos asistentes que se identificaban como evaluadores, aunque sabíamos que los evaluadores tenían mucho valor que agregar. Experimentamos, discutimos las pruebas con los pioneros de XP e intercambiamos ideas entre nosotros y con otros miembros de nuestros respectivos equipos.



Impulsando el desarrollo con ejemplos

El desarrollo ágil evolucionó y las pruebas ágiles evolucionaron junto con él. Los equipos comenzaron a desarrollar bibliotecas de prueba y marcos para la automatización de pruebas por encima del nivel de unidad. Cuando escribimos Agile Testing, muchos profesionales ágiles reconocían las contribuciones de evaluadores experimentados como Elisabeth Hendrickson y Michael Bolton. Profesionales como Brian Marick y Joshua Kerievsky fueron pioneros en la idea de utilizar ejemplos y pruebas de historias para guiar el desarrollo.

Ward Cunningham creó Fit (Marco para pruebas integradas), una herramienta para ayudar a definir esos ejemplos, y Dan North introdujo el desarrollo impulsado por el comportamiento (BDD) (North, 2006), que allanó el camino para nuevas herramientas populares. Los equipos ágiles habían comenzado a comprender el valor de las pruebas exploratorias. Las pruebas en equipos ágiles habían ido más allá de lo funcional para abarcar muchos tipos de pruebas, como lo ilustra la matriz de pruebas ágiles de Brian Marick (Marick, 2003), que adaptamos a los cuadrantes de pruebas ágiles (también nos referimos a ellos como los cuadrantes).

Por supuesto, todavía había desafíos que obstaculizaban el éxito de las pruebas ágiles. Los evaluadores envidiábamos todas las ayudas para las pruebas unitarias que estaban integradas en los entornos de desarrollo integrados (IDE) de los programadores. Queríamos esa misma facilidad para los evaluadores que especifican pruebas. Encontramos enormes beneficios al aplicar el “Poder de Tres” o “Tres Amigos”, como lo llama George Dinwiddie (Dinwiddie, 2010), donde un cliente, un programador y un evaluador colaboran cada vez que surgen preguntas sobre cómo funciona un característica debe comportarse. Sin embargo, todavía nos resultó difícil capturar muchas dimensiones de los requisitos del cliente, como el diseño, la usabilidad, los datos y otros atributos de calidad. Estos son algunos de los desafíos que abordamos en este libro.

Profesionales de varias disciplinas han estado llenando vacíos en la práctica de las pruebas ágiles. Tenemos la suerte de poder compartir las historias de otros profesionales sobre cómo han tenido éxito con las pruebas ágiles en diferentes contextos.

Una idea clave se ha convertido en parte de nuestro pensamiento cotidiano: las pruebas en equipos ágiles son una actividad, no una fase. Aprendimos este concepto de Elisabeth Hendrickson (Hendrickson, 2006) y lo aplicamos a lo largo del libro para enfatizar que las pruebas son algo que todas las disciplinas deben considerar al desarrollar software.



aprendiendo

A medida que hemos aprendido continuamente más y mejores técnicas para realizar pruebas en forma ágil, hemos descubierto cómo los especialistas en generalización con profundidad y amplitud en sus conjuntos de habilidades ayudan a los equipos a enfrentar desafíos de prueba difíciles. Los profesionales han creado prácticas y patrones que ayudan a los miembros del equipo en diferentes disciplinas a colaborar y aprender “lo suficiente” de las especialidades unos de otros.

Profesionales como los miembros del grupo Agile Alliance Functional Test Tools (AA-FTT) han abierto el camino hacia mejores herramientas para las pruebas. Hoy en día, escribir código de prueba está a la par que escribir código de producción. Hemos aprendido mejores formas de identificar qué marcos, bibliotecas de prueba y controladores se adaptan bien a las necesidades de un equipo específico.

Los analistas de negocios han aportado sus habilidades especializadas para descubrir los requisitos de los clientes al desarrollo ágil. Las pruebas y el análisis empresarial comparten habilidades complementarias que ayudan a los equipos a ofrecer el valor empresarial adecuado. De manera similar, los expertos en experiencia de usuario (UX) nos han demostrado

formas buenas y sencillas de obtener comentarios de los clientes a medida que se diseñan nuevas funciones. Los profesionales de DevOps han combinado habilidades de desarrollo, operaciones y pruebas para mejorar la calidad en nuevas dimensiones, como la entrega y la implementación, y para ayudar a acortar los ciclos de lanzamiento para reducir el riesgo y el impacto para los clientes.



Exploratorio
pruebas

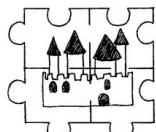
Seguimos conociendo muchos más equipos ágiles que comprenden el valor de las pruebas exploratorias que los que vimos hace unos años. Aunque nuestro primer libro cubre las pruebas exploratorias, está claro que los equipos pueden aprovechar su poder de muchas más maneras. Somos afortunados de que las personas que practican pruebas exploratorias en un contexto ágil compartan su experiencia.

Hemos visto enfoques nuevos y creativos para la planificación y colaboración de pruebas.

Hoy en día, los equipos encuentran más formas de visualizar la calidad. Los Cuadrantes y la pirámide de automatización de pruebas se han adaptado y ampliado para representar más dimensiones.



Contexto
sensibilidad



Característica
pruebas

Hoy en día, más equipos se enfrentan a la prueba de aplicaciones móviles y software integrado en una gama cada vez mayor de dispositivos y plataformas. Grandes y complejos conjuntos de datos con nueva tecnología para almacenar, gestionar, analizar, buscar y visualizar los datos definen una nueva categoría: Big Data. Las pruebas deben mantenerse al día.

El desarrollo ágil surgió para su uso en equipos pequeños y ubicados. Ahora vemos que las grandes organizaciones, algunas de las cuales comenzaron como pequeñas empresas ágiles, así como equipos distribuidos, están utilizando un enfoque ágil para el desarrollo.

En organizaciones con soluciones de software para toda la empresa, las pruebas se topan con diferentes obstáculos, como restricciones organizativas pesadas.

Al mismo tiempo, las organizaciones están encontrando nuevas formas de desarrollar productos mínimos viables (MVP), utilizando lanzamientos iterativos con ciclos de retroalimentación rápidos y aprendizaje validado.

Si bien es posible que algunas empresas hayan estado realizando pruebas en producción en 2008, cuando escribimos nuestro primer libro, no empezamos a oír hablar de esa técnica hasta más tarde, ja menos que cuente el lanzamiento sin pruebas y esperando lo mejor! Hoy en día, publicar actualizaciones para un pequeño porcentaje de usuarios de producción mientras se monitorean los archivos de registro en busca de errores y otras técnicas para obtener comentarios rápidos de los usuarios de producción, puede ser una estrategia válida y necesaria en el contexto apropiado.



Acuerdo
Es real

Todos estos cambios e innovaciones nos impulsaron a compartir lo que nosotros y otros profesionales hemos aprendido. Agile se trata de mejorar continuamente, y es posible que muchos de ustedes ya lleven varios años en una adopción ágil, cambiando su proceso a medida que aprenden. Los problemas relacionados con las pruebas que enfrenta ahora probablemente sean muy diferentes de los que eran durante su transición inicial a la metodología ágil. Esperamos que las experiencias compartidas en este libro le brinden algunas ideas para realizar experimentos mientras visualiza la calidad de su producto y luego inspecciona y adapta su proceso.

Resumen

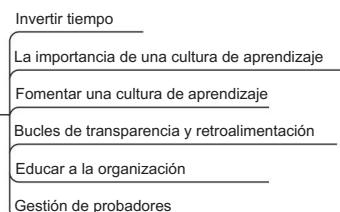
A medida que evoluciona el desarrollo de software ágil, inspeccionamos y adaptamos continuamente las pruebas ágiles para mantener el ritmo. En este capítulo, analizamos algunas formas en que las pruebas ágiles han ido cambiando.

- Hemos reconocido que las pruebas son una actividad crucial para éxito del producto y es de interés para todos los miembros del equipo del producto, desde las partes interesadas hasta las operaciones y el soporte.
- Los equipos ágiles reconocen cada vez más las formas en que los profesionales de otras disciplinas, como el análisis empresarial, el diseño de la experiencia del usuario y DevOps, han ampliado nuestra capacidad para incorporar la calidad que nuestros clientes desean.
- Las herramientas para pruebas ágiles continúan avanzando a pasos agigantados, permitiendo a los equipos ágiles implementar la infraestructura necesaria para respaldar el aprendizaje y la retroalimentación rápida.
- Los equipos ágiles están aprendiendo el valor de las pruebas exploratorias y otras prácticas que nos ayudan a proporcionar información esencial a los clientes y equipos de desarrolladores.
- Las pruebas ágiles deben mantenerse al día con la tecnología que cambia rápidamente y los nuevos contextos para el desarrollo ágil. Exploraremos estas ideas a lo largo de este libro.

Capítulo 2

La importancia de Cultura organizacional

2. La importancia de la cultura organizacional



Hemos notado un tema común en las organizaciones que están en transición hacia métodos ágiles. Los equipos de desarrollo pueden adoptar valores, principios y prácticas ágiles. Sin embargo, el lado comercial de la empresa puede tardar en comprender los beneficios de la metodología ágil y, a menudo, puede tardar más en adaptarse al cambio. Para una transición exitosa hacia la metodología ágil, es importante que tanto el desarrollo como el negocio estén sincronizados y comprendan la realidad de cada uno.

Cuando trabaja en una empresa cuyos líderes entienden que centrarse en la calidad es una de las mejores maneras de ofrecer valor comercial a los clientes con frecuencia, es probable que su equipo (y su empresa) tengan éxito. La frecuencia y consistencia de la entrega mejoran con el tiempo. Por el contrario, si una empresa se centra primero en la velocidad, a menudo se sacrifica la calidad. La deuda técnica en forma de código frágil y difícil de cambiar y ciclos de retroalimentación más lentos disminuye la capacidad del equipo para lograr resultados consistentes.

Si los equipos se sienten presionados a cumplir plazos poco realistas con un alcance definido, ya no tienen el control de su propia carga de trabajo. A medida que se acerca la fecha límite, los desarrolladores sienten que tienen menos opciones y toman decisiones para cumplir con la fecha límite en lugar de satisfacer al cliente (Rogal-sky, 2012). Es probable que empiecen a tomar atajos y a tomar decisiones.



Figura 2-1 Demasiadas funciones para la infraestructura

sin pensar en posibles efectos dominó, y pueden caer en una espiral mortal de deuda técnica cada vez mayor.



Los equipos deben poder articular los peligros de un ritmo insostenible para que los líderes de la empresa sean conscientes de los riesgos y las consecuencias. A veces, un equipo necesita reducir el ritmo para abordar la deuda técnica, o tal vez para aprender nuevos conceptos o aplicar nuevas ideas. Janet utiliza una imagen (Figura 2-1) en sus presentaciones para mostrar cómo agregar más y más funciones sin considerar la infraestructura puede ser realmente perjudicial. La imagen muestra cuatro autos con superpoderes alineados en una carretera, sin ir a ninguna parte porque el tráfico es muy denso. ¿Por qué agregar más funciones cuando las que ya tiene no funcionan según lo previsto? Cuando acumula demasiada deuda técnica, su equipo sufre un estado similar de estancamiento.

Invertir tiempo

Se requiere creatividad para ofrecer a los clientes soluciones de software útiles, y las personas necesitan tiempo para pensar y utilizar su ingenio. También debemos seguir aprendiendo nuevas habilidades y herramientas técnicas para poder realizar pequeños experimentos que puedan ayudar a abordar sus problemas.

En un artículo titulado “Ideas lentas” (Gawande, 2013), Atul Gawande exploró por qué algunas innovaciones, como la anestesia quirúrgica, se popularizan rápidamente, mientras que otras, como los antisépticos, tardan muchos años o es posible que nunca consigan arraigarse. Su observación fue que el cambio sólo puede ocurrir con el aprendizaje, la práctica y un toque personal. En los programas que estudió, las personas aprendían mejor cuando realizaban la nueva actividad ellos mismos y la describían con sus propias palabras, con la orientación de un formador. Puede parecer más barato o más rápido que un formador demuestre la actividad o que la gente vea un vídeo, pero el enfoque práctico produce un cambio real y sostenible.



En la profesión del software (como en muchas otras), la gente parece olvidar que se necesita tiempo para aprender y practicar una nueva habilidad. También suelen descuidar la presencia de un formador que guíe a los alumnos.

Dedique tiempo a lo correcto

David Evans , un Entrenador ágil de calidad, comparte la idea de metáfora él explicar los usos dedicar el tiempo a lo correcto. en

A veces encuentro resistencia de la dirección a mis sugerencias de inversión. tiempo para escribir pruebas de aceptación en colaboración antes de implementarlas cada historia. El argumento habitual es que si el equipo dedica más tiempo en las pruebas de aceptación, ralentizará el ritmo de desarrollo. I Admito que sí, en un sentido muy limitado, se ralentizará, pero sólo de la misma manera que detenerse para los pasajeros ralentiza al público autobús. Tratar de optimizar la velocidad del autobús no tiene sentido, ya que no tiene nada que ver con el propósito o el éxito del transporte público. Imagine a un conductor de autobús sugiriendo que podría mejorar su desempeño. métricas si no se detuviera para pasajeros de edad avanzada, como parecen ser el más lento para abordar. De hecho, si no recogiera ningún pasajero, Sería mucho más rápido y no tendría que parar en todas esas paradas de autobús. en su sinuoso camino. En cambio, podría tomar la autopista y estar ¡De vuelta al depósito en un abrir y cerrar de ojos!

Esta es la misma lógica errónea que dice que trabajar en pruebas ralentizará desacelerar el desarrollo. Desarrollo basado en pruebas: Por Ejemplo , kent Beck tiene una bonita afirmación (Beck, 2002): "El código que no se prueba no trabajo, ésta parece ser la suposición segura". Si dedicas tiempo a crear Las pruebas de aceptación antes de la codificación ralentizan cualquier cosa, solo están ralentizando el ritmo al que estamos creando código que no funciona.

Las empresas manufactureras que adoptan valores lean nos han enseñado que respetar la capacidad de las personas para aportar ideas y trabajar continuamente para mejorar se traduce en productos innovadores que aman a los clientes. Cuando las organizaciones brindan a los equipos tiempo y apoyo para aprender y experimentar, esos equipos pueden hacer su mejor trabajo. En la Parte II, "Aprender para mejorar las pruebas", exploraremos algunas de las prácticas que creemos que merecen una inversión en aprendizaje.

Gojko Adzic (Adzic, 2013) ha señalado que se obtienen grandes resultados cuando

- La gente sabe por qué hace su trabajo.
- Las organizaciones se centran en generar resultados e impactos en lugar de que características
- Los equipos deciden qué hacer a continuación basándose en resultados inmediatos y directos. retroalimentación sobre el uso de su trabajo
- A todos les importa

Algunas empresas han intentado crear holgura y darles a los empleados tiempo para aprender, creando tiempo de holgura para trabajar en proyectos personales, ya sea como un porcentaje del tiempo de trabajo o semanas o días dedicados a "trucar". Lisa ha trabajado en equipos que utilizan esta práctica y obtuvieron buenos resultados al mantener la deuda técnica baja y el rendimiento constante. Sin embargo, es importante comprender los principios detrás de la utilización y la capacidad. De lo contrario, los equipos podrían tener que intentar exprimir aún más trabajo entre plazos estrictos.

Capacidad de uso

Alexéi Zheglov , a consultor en lean y kanban para organizaciones de trabajo del conocimiento, explicó la teoría detrás Utilización de la capacidad y cómo superar los ~~límites~~ del 20%. escape. Hemos incluido un parte de su barra lateral aquí; el com-el sitio web del libro. completo uno puede ser encontrado en

La razón principal para el 20% de tiempo es mantener la utilización promedio de la capacidad en 80% en lugar de 100% para generar tiempo de inactividad. podemos pensar de una organización de desarrollo de software como un sistema que convierte funciones solicitudes en funciones desarrolladas. Entonces podemos modelar su comportamiento. utilizando la teoría de colas.

Teoría

Si las solicitudes llegan más rápido de lo que el sistema puede atenderlas, se ponen en cola. Cuando las llegadas son más lentas, el tamaño de la cola disminuye. Debido a que los procesos de llegada y servicio son aleatorios, el tamaño de la cola cambia aleatoriamente con el tiempo (Figura 2-2).

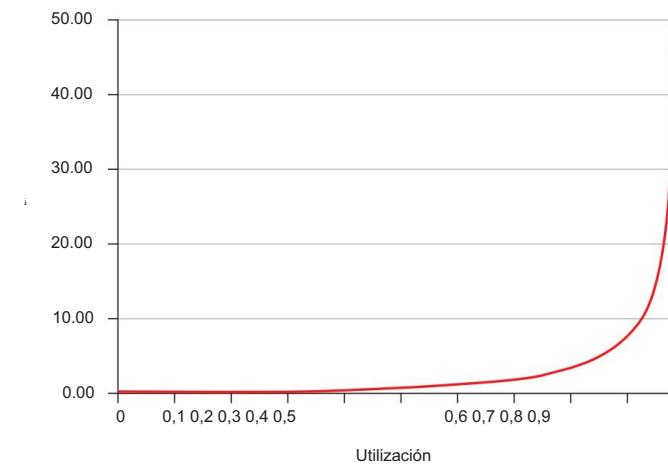


Figura 2-2 Curva J sobre utilización

El tamaño medio de la cola sigue siendo bajo mientras que la utilización es de hasta 0,8, luego aumenta bruscamente y llega al infinito. Podemos entender esto intuitivamente pensando en la CPU de nuestra computadora: cuando su utilización es baja, responde instantáneamente a nuestras entradas, pero cuando una tarea en segundo plano eleva su utilización cerca del 100%, la computadora se vuelve frustrantemente lenta para responder a cada clic.

Práctica

Es esencialmente la holgura lo que mantiene al sistema receptivo. Varias conclusiones prácticas sobre lo que no se debe hacer son

- Contabilidad de costos (el tiempo de los ingenieros cuesta X, pero es posible que la empresa no pueda permitírselo). El beneficio económico proviene de la reducción del costo del retraso. •

Establecer un sistema de propuestas para que las personas presenten proyectos para realizar en su “20% de tiempo”.

- Seguimiento del 20% del tiempo mediante el llenado de hojas de tiempo.

- Utilizar la innovación como motivador durante el 20% del tiempo. Mientras que nuevo Los productos han salido de un 20% de proyectos, no eran los punto. Si su empresa no puede innovar durante sus horas centrales, ¡eso es un problema!
- Aprovechar el 20% del tiempo para fomentar la creatividad. diciendo que lo harás Libera tu creatividad con un 20 % de tiempo y plantea la pregunta de por qué. ya no eres lo suficientemente creativo durante tus horas principales.
- Destinar el 20% del tiempo a un viernes de cada semana.

Ésas son todas las cosas que no se deben hacer; ¿Dónde están los dos?

Bien, ¿qué tal si lo hacemos bien? Respondamos con la mejor pregunta. hemos escuchado al discutir este tema con los profesionales: "Si el 20% de su capacidad debe llenarse con elementos que no están en cola, entonces acaba de reducir su capacidad al 80% y el 80% es su nuevo 100%.

¿Bien?"

Sí, "80 son los nuevos 100" destaca el principal problema de los intentos imponer el 20% del tiempo sin entender la teoría. Quieres escapar de la trampa de la utilización, no permanecer en la trampa y asignar tiempo diferentemente.

La utilización depende de dos procesos: proceso de llegada (demanda) y proceso de servicio (capacidad). Realmente no puedes elegir tu utilización. Él es lo que es porque los procesos son los que son. Sin embargo, puedes trabajar en los procesos mejorando el software de tu empresa. capacidad de entrega y configuración de la demanda. A medida que avanzas, surgirá holgura.

La bibliografía de la Parte I, "Introducción", incluye recursos donde puede aprender más sobre la gestión de la capacidad y el rendimiento. Una de las cosas más importantes que hay que recordar es que la sobreutilización afecta el flujo y un ritmo insostenible en realidad reduce el rendimiento.

La importancia de una cultura de aprendizaje

Si puedes "fallar rápido" lo suficientemente rápido como para que el fracaso no sea demasiado costoso, puedes aprender de esos errores para mejorar. También puede utilizar ciclos de retroalimentación rápida para perfeccionar sus pruebas y cambiar algo que funcionó "bien" en algo que funcione muy bien. Si la cultura de su organización es tal que no se toleran los errores, la innovación es poco probable.

Hacer el mismo proceso, incluso si no funciona, se siente seguro, por lo que no hay razón para c

La historia de Lisa

He tenido la suerte de pasar la mayor parte de mi carrera en equipos que trabajaron en un ritmo sostenible y valorando la mejora continua. Cuando me he encontrado con ambientes donde se desaconsejan las preguntas y los experimentos, Se siente como si me hubiera estrellado contra una pared de ladrillos. En un equipo con el que trabajé durante un Al poco tiempo un ScrumMaster me dijo que yo era un "impedimento para el equipo" porque seguí planteando problemas en el stand-up. Estaba bajo presión de la gerencia para que el equipo cumpliera. No quería verse frenado por alguien señalando que no había habido un entorno de prueba durante tres semanas. Había mucha gente estupenda en el equipo, pero no eran permitido hacer su mejor trabajo. Mis intentos de ser un agente de cambio en este La organización fracasó, así que, como dicen, cambié de organización.

En su charla sobre el poder de las retrospectivas (Rising, 2009), Linda Rising señaló que a menudo, cuando se enfrentan a prácticas como la programación en pareja o las retrospectivas, algunos directivos tienen esta reacción: "No tenemos tiempo para pensar". Si creemos que podemos mejorar, lo haremos: Quizás no todo de una vez, o de la noche a la mañana, pero podemos encontrar mejores formas de trabajar, paso a paso.

Fomentar una cultura de aprendizaje



Reúna a su equipo para ver qué habilidades podrían faltar y qué personas están interesadas en aprender. Permítales administrar su carga de trabajo para que puedan dedicar tiempo a experimentos, lecturas, cursos de capacitación y practicar nuevas habilidades. Los miembros del equipo deben sentirse capacitados para investigar una nueva técnica o probar una nueva herramienta. Por ejemplo, se les podría permitir dedicar algo de tiempo diario o semanal a trabajar en lo que quieran, ya sea aprender habilidades de facilitación o encontrar una nueva solución para automatizar las pruebas de regresión. Las semanas de piratería en las que los miembros del equipo trabajan en proyectos experimentales o aprenden una nueva tecnología son un enfoque cada vez más popular. Los miembros del equipo comparten lo que han aprendido, lo que ayuda a fomentar una cultura de aprendizaje.

Gojko Adzic nos dijo que ha tenido éxito trabajando con la gerencia y preguntándoles si esperan que los equipos mejoren su proceso con el tiempo o si creen que el proceso ya es el mejor posible. Habiendo establecido que el proceso realmente podría mejorarse, trabaja con ellos para desarrollar un presupuesto, no necesariamente en términos de dinero, sino en términos de

a tiempo. Por ejemplo, los miembros del equipo deberían dedicar el 10% de su tiempo a mejorar activamente su proceso. Cada equipo debe decidir cómo utilizarlo, pero como parte de la negociación, la dirección puede retirar este presupuesto durante un breve período de tiempo cuando sea necesario. Si el presupuesto se retira constantemente, es una señal de advertencia de que se requiere otra conversación seria.

Una forma en que una organización puede fomentar una cultura de aprendizaje es estableciendo comunidades de práctica (algunas organizaciones las llaman gremios) y permitiendo que estos grupos se tomen el tiempo para discutir y compartir ideas durante el tiempo de trabajo. Una y otra vez hemos oido hablar de estas comunidades discutiendo libros o compartiendo su trabajo con otras personas de ideas afines.

Gremios de calidad

Augusto Evangelista , a ingeniero de pruebas en Arrozal Poder, acciones cómo crearon los evaluadoresen su lugar de trabajo a gremio de calidad.

En mi actual lugar de trabajo contamos con un gremio de calidad compuesto por un 20% probadores, 20% facilitadores kanban y 60% desarrolladores. Necesitamos alguien que nos diga qué hacer, qué problemas debemos abordar, ¿Y en qué innovación deberíamos trabajar? Créeme, no. Empezamos con cinco personas, y después de un par de meses teníamos 20 personas con Ideas increíbles para mejorar la calidad y mejorar nuestra vida. Nosotros reunirse cada dos semanas y tener siempre una lista de temas de los que hablar.

Intenté el mismo ejercicio como gerente y terminé con cinco evaluadores. que tenían que estar allí, sentados y esperando que yo les dijera qué hacer. Es muy diferente cuando la gente tiene el poder de hacer sus propias nuestras propias vidas mejor.

Somos fanáticos de las sesiones de Lean Coffee, que alientan a los participantes a plantear nuevas ideas o inquietudes y hablar abiertamente sobre ellas. Llevar a cabo una sesión estilo Lean Coffee abierta a todos en una organización puede ayudar a aclarar malentendidos. Consulte los enlaces de Lean Coffee en la bibliografía de la Parte I para comenzar.

Algo tan simple como emparejarse con otro miembro del equipo, remoto o compartido, puede ser una oportunidad de aprendizaje o capacitación, especialmente si está tratando de aprender una nueva habilidad que otra persona ya domina.

Cambiar una cultura establecida es difícil. Si la agilidad es nueva para su equipo, ayude a los miembros del equipo a adaptarse dando pequeños pasos. Ayúdelos a adoptar nuevos hábitos y asegúrese de que cada miembro del equipo se sienta valorado y seguro.

Aprenda a adaptarse al cambio

Bernice Niel Ruhland, a director de gestión de calidad, comparte sus experiencias de adopción a reunión diaria de pie.

Muchos equipos se llaman a sí mismos "ágiles" pero no siguen los principios detrás de la carta del Manifiesto Ágil. Son "ágiles" de espíritu, pero no necesariamente en prácticas comunes a los equipos autogestionados. Entender las prácticas y adopte lo que funcione para su equipo. Empieza con algo pequeño y deja que tus ideas evolucionen de forma natural. Los fracasos pueden conducir a nueva información para entender dónde y cómo aplicar diferentes enfoques y técnicas.

Por ejemplo, si desea iniciar una reunión diaria, programe 15 minutos a la misma hora y en el mismo lugar todos los días. Comenzar con las preguntas que suelen utilizar los equipos ágiles: ¿Qué hiciste ayer; Qué vas a hacer hoy; ¿Y hay algún obstáculo en tu forma? Evalúe cómo funcionan esas preguntas, tal vez para su proyecto son necesarias diferentes preguntas.

Las reuniones breves son una excelente manera de mantener a todos en movimiento. la dirección correcta y promover la comunicación cara a cara. Permanecer Se centró en el resultado positivo de la reunión diaria en lugar de defender el formato. Facilite las reuniones para que sean breves, de modo que los participantes sepan que se valora su tiempo, y modifique el formato si es necesario.

En el Capítulo 6, "Cómo aprender", veremos más formas en que los equipos pueden aprender y practicar las habilidades que necesitan para tener éxito con las pruebas en equipos ágiles.

Bucles de transparencia y retroalimentación

Los ciclos cortos de retroalimentación son un aspecto importante no solo de las pruebas sino también del ciclo completo de entrega ágil de software. Se necesita transparencia en la toma de decisiones para tener éxito a largo plazo en el suministro de software de alta calidad.

Si es un gerente directo en una organización de software, o aspira a serlo, aprenda buenas formas de generar ciclos de retroalimentación y transparencia en su equipo.

cultura. Leer libros orientados al desarrollo ágil, como Management 3.0 de Jurgen Appelo (Appelo, 2011). Los artículos y publicaciones de blogs de líderes en la cultura organizacional ágil, como Johanna Rothman y Esther Derby, son excelentes fuentes de ideas. Consulte la bibliografía de la Parte I para obtener más recursos.

Cuando nuestro trabajo es transparente para la organización, se reduce la necesidad de control. La visibilidad de lo que están haciendo los equipos de desarrollo ayuda a generar confianza con las partes interesadas del negocio y los ejecutivos de la empresa. La confianza se construye con el tiempo con paciencia, práctica y coherencia (consulte la Figura 2-3).

Dominar los valores y principios ágiles requiere un gran cambio cultural para la organización. Para citar a Johanna Rothman, “Agile es para personas que quieren y pueden gestionar el cambio cultural que requiere” (Rothman, 2012a). Nuestro verdadero objetivo no es implementar la metodología ágil; es entregar los productos que nuestros clientes desean. Spotify, un servicio de música digital, es citado a menudo como una empresa que adoptó con éxito una cultura ágil y que ha crecido, se ha adaptado y ha hecho transparente todo el proceso. Henrik Kniberg ha compartido las experiencias de Spotify en un video animado que vale la pena ver (Kniberg y Spotify Labs, 2013).

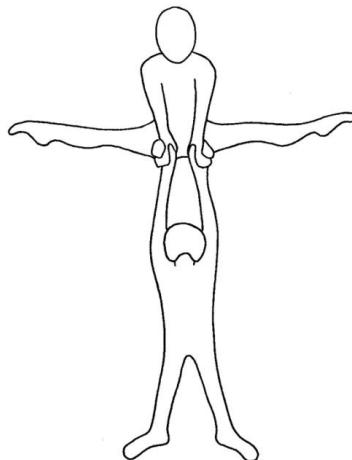


Figura 2-3 La confianza requiere práctica, paciencia y constancia.

Celebrar los éxitos, tanto grandes como pequeños, es importante. Concientizar a las personas sobre los éxitos de los demás para que quieran ser parte de una cultura que celebra los logros de las personas y los equipos.

Educar a la organización

Para que la cultura de una empresa cambie, los altos ejecutivos deben estar dispuestos a cambiarla. Esto significa que deben comprender lo que significa el cambio para ellos.



Antes de intentar educar a los ejecutivos de su empresa, infórmese usted mismo sobre sus funciones y responsabilidades. Por ejemplo, el director financiero de la empresa proyecta el desempeño financiero e intenta predecir el retorno de posibles inversiones. Esa persona sabe mucho sobre todos los aspectos del negocio y probablemente esté interesada en comprender cómo avanzan los proyectos de software y qué podría causar retrasos.

Encuentre formas de cuantificar y articular las ventajas de los cambios que desea, así como el costo de los problemas continuos, como la acumulación de deuda técnica. Muestre a los ejecutivos cómo los problemas impactan los resultados de la empresa.

Cuando los líderes empresariales trabajan con los equipos de desarrollo para establecer objetivos impulsados por el negocio, los equipos comprenden lo que es importante. Luego pueden hacer sugerencias para priorizar y recortar alcances innecesarios. Hablamos más sobre la idea del mapeo de impacto para ayudar con esto en el Capítulo 9, “¿Estamos construyendo lo correcto?”

Las partes interesadas pueden priorizar los resultados, de modo que el equipo pueda ofrecer incrementos valiosos dentro del plazo necesario y utilizar ciclos de retroalimentación rápidos para seguir mejorando el producto y brindando retroalimentación visible sobre el progreso de los proyectos y el desarrollo.

Genere confianza entregando valor con regularidad, incluso si es una cantidad pequeña. Como advierte Bob Martin (Martin, 2011), no diga “lo intentaré” cuando le den plazos poco realistas. En su lugar, ayude a los gerentes a comprender lo que su equipo puede ofrecer de manera realista, cuantifique el costo de los atajos que pueda tener que tomar y ayúdelos a reducir el alcance para que puedan obtener sus funciones de máxima prioridad.



Según nuestra experiencia, los ejecutivos de negocios se sienten incómodos con la incertidumbre inherente al desarrollo de software. Como desarrolladores de software, es posible que podamos predecir de manera confiable las fechas de entrega de funciones que son simples o que hemos hecho antes y entendemos bien. Sin embargo, las partes interesadas tienden a querer capacidades nuevas y brillantes que la competencia no tiene. Si la empresa quiere algo que sea impredecible, explíquenle las complejidades. Consideraremos un enfoque como el de Opciones Reales (Matts y Maassen, 2007). Chris Matts ha adaptado esta idea del mundo financiero en el que las opciones se mantienen abiertas hasta el último momento responsable en el que se requiere una decisión. Pagar para ampliar las opciones genera tiempo para conocer soluciones alternativas, lo que permite a los equipos tomar mejores decisiones. Por ejemplo, podemos elegir tecnología que facilite los cambios o acordar un rango de fechas en lugar de una fecha límite (Keogh, 2012b).

El marco Cynefin, desarrollado por Dave Snowden, proporciona una manera de evaluar si una nueva característica será sencilla de implementar o si tiene incógnitas que deben explorarse antes de que pueda surgir una solución. Consulte los enlaces en la bibliografía de la Parte I para obtener más información sobre formas de "abrazar la incertidumbre" (Keogh, 2013b).

Ya sea que utilice o no un enfoque o marco definido para gestionar el riesgo y la incertidumbre, los equipos pueden ayudar a los tomadores de decisiones de la empresa a comprender por qué a menudo no podemos predecir el progreso de manera confiable y cómo las pruebas pueden ayudarlos a mantenerlos informados.

La historia de Janet

Una organización con la que trabajé encontró una forma muy creativa de lograr que los ejecutivos comprendieran muy rápidamente algunos de los problemas de los equipos. Había múltiples equipos, todos trabajando en el mismo producto, por lo que tenían tareas transversales preocupaciones. Todas las reuniones diarias se llevaron a cabo a las 9:15 o 9:30 am en A las 10:00 am, realizaron un Scrum-of-Scrums en una sala de conferencias donde todos Los ScrumMasters hablaron sobre esas preocupaciones transversales. A las 10:15, uno ScrumMaster se quedó atrás, y cualquier ejecutivo que tuviera un interés personal en El producto llegó a esa habitación. Discutieron cualquier tema nuevo y luego revisó las cuestiones pendientes que fueron asignadas, fechadas y enumeradas en un pizarrón. Si se resolvió un problema, se borró. Cualquier problema nuevo que debían tratarse fuera de los 15 minutos asignados, se añadieron, asignado y fechado.

El libro *Fearless Change* (Manns and Rising, 2005) explica buenas formas de introducir nuevas ideas. Para obtener orientación al reunirse con ejecutivos, consulte el patrón “Susurrar al oído del general” (págs. 248 y 249).

Siempre que interactúes con personas que no formen parte del equipo de entrega, aprovecha la oportunidad para informarles sobre cómo estás trabajando.

Pregúntales qué podrían necesitar saber para hacer su trabajo y esté dispuesto a expresar lo que usted podría necesitar de ellos.

Gestión de probadores



En Agile Testing hablamos bastante sobre los gestores de pruebas y cuál podría ser su nuevo rol. Sin embargo, todavía recibimos muchas preguntas al respecto y vemos conversaciones frecuentes sobre este tema en diferentes foros. No existe una única respuesta correcta; mucho depende del contexto. Por ejemplo, si tiene una organización pequeña con solo un par de equipos, es posible que no necesite ningún gerente de línea separado.

Sabemos de equipos donde todos los miembros reportan a un gerente de entrega o desarrollo, y funciona muy bien si el gerente tiene una buena apreciación de lo que hacen los evaluadores. La historia de Augusto Evangelisti sobre los gremios, descrita anteriormente en este capítulo, ilustra este enfoque. En su empresa, los evaluadores senior asesoran a los nuevos empleados y todos trabajan para mejorar su sistema. Actualmente cuentan con seis equipos y Augusto cree firmemente que a medida que crezcan, su cultura les permitirá mantener esta forma de trabajar.

Algunas organizaciones pueden necesitar una estructura de informes secundaria o alguien específicamente para ayudar al aprendizaje entre equipos. Adam Knight compartió su experiencia con nosotros. Para él, el papel de un director de pruebas va más allá del ámbito del trabajo diario del equipo ágil.

Gran parte de su trabajo implica mirar más allá de las actividades de iteración y más hacia las necesidades culturales y estratégicas de la operación de prueba. Cree que es necesario tener a alguien que represente las pruebas al mismo nivel que el director de desarrollo, para garantizar un enfoque equilibrado que considere todas las disciplinas por igual. A menudo, en organizaciones más grandes, las disciplinas son más especializadas, por lo que esto puede tener sentido. Adam también trabaja para garantizar que la organización tenga el equilibrio adecuado de habilidades en toda la operación de prueba y que los evaluadores de los equipos ágiles trabajen en colaboración y no dupliquen esfuerzos.

Un director de pruebas también puede ser una especie de entrenador para las comunidades de práctica dentro de una organización. El director de pruebas no está ahí para prescribir, sino que puede facilitar sesiones de aprendizaje en las que las partes interesadas puedan discutir nuevas ideas. Como dijimos, no existe un único camino correcto. Comprenda su contexto y lo que necesita, para que pueda practicar la agilidad y obtener el mayor beneficio para su organización.

Comuníquese con las comunidades globales de desarrollo y prueba de software para obtener ayuda para realizar cambios culturales que respalden la construcción de calidad, la entrega de valor con frecuencia y el trabajo a un ritmo sostenible. Mencionaremos algunas comunidades en línea útiles en el Capítulo 6, “Cómo aprender”.

Resumen

Este capítulo trataba sobre la importancia de cambiar y adaptar la cultura organizacional. Destacamos algunas ideas que necesitan nutrirse a nivel de equipo y organización:

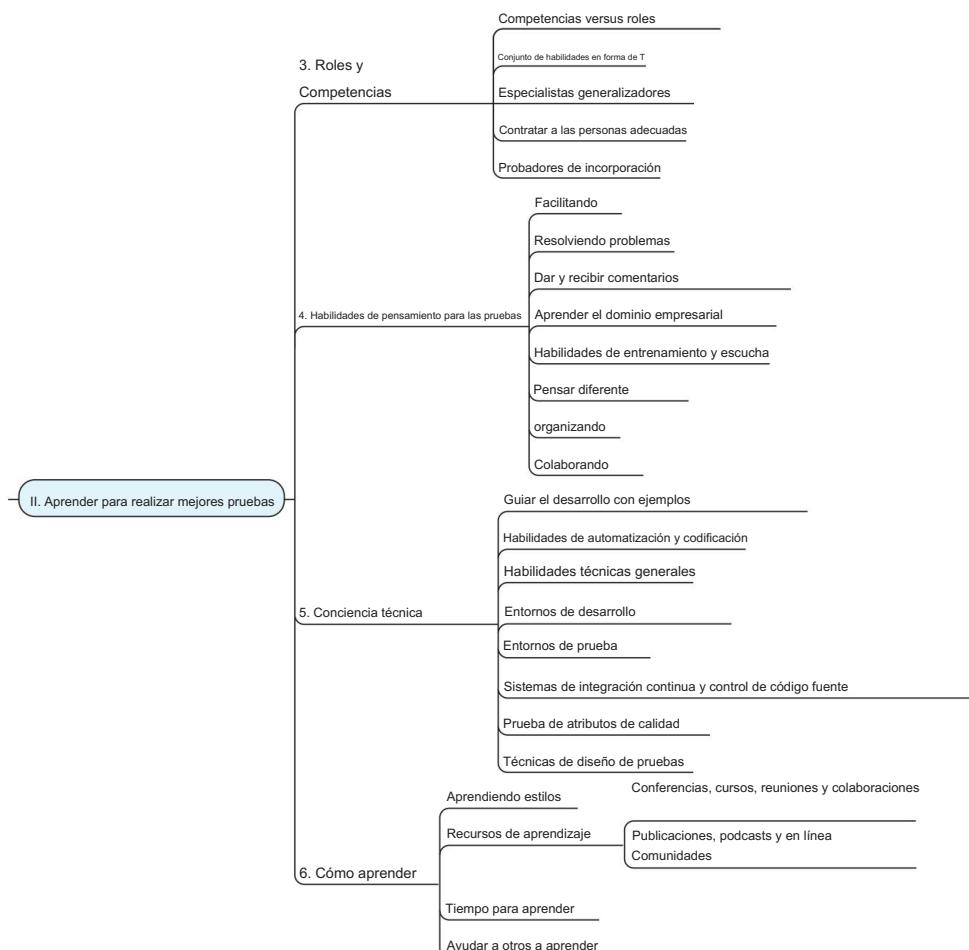
- Centrarse en la calidad, lo que conduce a un ritmo sostenible a largo plazo y a una entrega consistente en el desarrollo de software.
- Libere su carga de trabajo mejorando la entrega de software capacidad y alcance de gestión, para que su empresa pueda responder a las oportunidades.
- Fomentar una cultura de aprendizaje en la que los equipos identifiquen impedimentos para la calidad y tengan tiempo para realizar pequeños experimentos para mejorar las prácticas y los procesos de prueba.
- Educar a las partes interesadas sobre cómo las buenas prácticas compensan el negocio.
- Haga que su proceso sea visible para ayudar a generar confianza entre los ejecutivos y los equipos de entrega.
- Recuerde celebrar los éxitos, grandes y pequeños.

Parte II

Aprender para realizar mejores pruebas

A medida que pasan los años, nos encontramos con más y más equipos en los que personas de todos los roles, no solo los evaluadores, participan en actividades de prueba. Al mismo tiempo, hemos descubierto que, como evaluadores, necesitamos ampliar y profundizar nuestra gama de habilidades para ayudar a nuestros equipos a ofrecer el valor que nuestros clientes requieren.

En la Parte II, exploraremos los roles y competencias del equipo relacionados con las pruebas y la calidad, junto con los tipos de habilidades necesarias para crear software de alta calidad, tanto desde una perspectiva "pensativa" como desde una perspectiva "técnica". El último capítulo de esta parte enfatiza la importancia del aprendizaje para individuos y equipos.

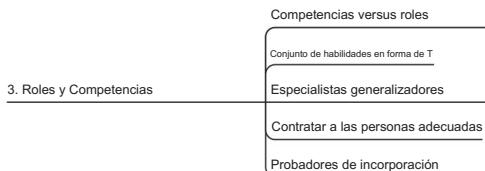


■ Capítulo 3, “Funciones y competencias” ■

Capítulo 4, “Habilidades de pensamiento para realizar pruebas” ■ Capítulo 5, “Conciencia técnica” ■ Capítulo 6, “Cómo aprender”

Capítulo 3

Roles y competencias



La gama de desafíos de prueba que cualquier equipo puede enfrentar parece ampliarse día a día. Es posible que su equipo esté trabajando en un producto basado en la web y, de repente, su empresa decida que también necesita una aplicación que funcione en dispositivos móviles. Los equipos adoptan nuevas tecnologías, prácticas de desarrollo, herramientas y marcos mientras, al mismo tiempo, cambian las prioridades de los clientes.

De ello se deduce que las actividades de prueba también tendrán que cambiar.

Una de las formas en que los equipos manejan estos cambios sin estar constantemente en el caos es gestionando qué y cómo aprenden.

Equipos autogestionados

Bernice Niel Ruhland , director de gestión de calidad, un comparte su opinión sobre los equipos autogestionados.

A menudo escuchamos sobre los beneficios y el éxito de los equipos autogestionados. Para algunas personas el cambio puede resultar liberador, mientras que a otras puede resultarles un desafío. Proporcionar un proyecto a un equipo y permitirles determinar cómo gestionar las tareas laborales puede resultar abrumador. Me he encontrado con equipos autogestionados que no pueden avanzar sin un líder asignado. Algunas personas parecen que se les asignen tareas; otros no se sienten cómodos diciéndoles a otras personas qué hacer; y a veces una o dos personas terminan con la mayor parte del trabajo.

Esto no significa que estos equipos no puedan hacer la transición a ser autogestionados; solo necesitan ayuda. Me resulta útil celebrar una reunión inicial del equipo para discutir el proyecto, responder preguntas y ayudar al equipo a establecer reglas básicas, con un gerente, entrenador o facilitador presente.

Como facilitador, puedo trabajar con ellos para definir hitos pero permitirles determinar cómo trabajarán juntos. Si tiene un puesto de liderazgo o gerencial, intente delegar la mayor parte de la responsabilidad al equipo.

Si tienen dificultades con la autoorganización, haga que un entrenador asista periódicamente a una reunión para ayudar a guiarlos cuando sea necesario. Sin embargo, tan pronto como sea razonable, devuelva la reunión al equipo.

Sesiones donde cada miembro del equipo discute sus habilidades y

La posible contribución al proyecto también ha resultado útil para mis equipos. A menudo las personas tienen habilidades ocultas de las que sus pares no son conscientes. Por ejemplo, un evaluador puede tener experiencia con un lenguaje de programación que podría usarse para automatizar algún trabajo repetitivo.

Haga avanzar la conversación más allá de las habilidades técnicas para incluir habilidades como gestión del tiempo, resolución de conflictos y relaciones internas y externas al equipo que podrían ser beneficiosas. Si es útil, cree una matriz de habilidades para que el equipo comience. Con el tiempo, cambie el enfoque de las habilidades documentadas a la construcción de relaciones dentro del equipo.

Generar confianza entre los miembros del equipo es aún más importante para un equipo autogestionado. Algunos equipos comparten intereses comunes y crean relaciones personales. Hay muchos tipos diferentes de actividades de formación de equipos; solo asegúrese de utilizar técnicas que el equipo respete.

Durante las retrospectivas, pregunte qué funciona para reforzar el buen comportamiento y analice qué es lo que los miembros del equipo encuentran difícil para ayudar a realizar cambios. Asegúrese de celebrar los éxitos y ayudar a facilitar el cambio cuando sea necesario.

Competencias versus roles

Hemos visto un movimiento positivo hacia enfatizar las competencias en un equipo en lugar de roles o títulos. A medida que los equipos hacen ese cambio, vemos menos excusas de "no es mi trabajo" y más "¿cómo puedo ayudar?". conversaciones. Los miembros del equipo seguirán teniendo competencias básicas en algunas áreas más que en otras, pero es posible que no se identifiquen tan fuertemente con un rol en particular. Por ejemplo, decir "Soy un evaluador" en realidad significa: "Realizo principalmente actividades de prueba porque esa es mi principal pasión y fortaleza. Puedo brindar liderazgo y orientación a otros, y también puedo ayudar en otras áreas".

¿Son importantes los títulos?

Un día en la conferencia, en una **Pete Walen**, de **Michigan, EE.UU.** era entrega ~~de su~~ tarjeta de presentación, Janet aceptó felizmente y así lo pidió. que embargo, su título le llamó la atención, él al respecto. Aquí está su respuesta.

Mi tarjeta de presentación tiene tres cosas diferentes, además de mi nombre e información de contacto.

El elemento obvio es "Probador de software". Eso es lo que hago, de una forma u otra. Pruebo software y trabajo con personas que también prueban software y les ayudo a realizar un mejor trabajo de prueba.

Después de mucha consideración, se añadió "Antropólogo del software". Finalmente se agregó en CAST (Conferencia de la Asociación para Pruebas de Software) en 2011. Michael Bolton dio una charla sobre desarrollo de software, y pruebas en particular, como ciencia social. Esto desencadenó una reflexión en mi mente y puso en marcha otras cosas. Entre ellas, fueron cruciales las preguntas sobre la interacción entre aplicaciones de software y cómo las personas interactúan con el mismo software. Al evaluar cómo se comporta el software, estas interacciones son importantes y forman una parte integral de lo que hace que las pruebas sean pruebas.

Esto nos lleva al tercer y más importante elemento: "El que hace la pregunta".

Parece que el control de calidad es un término que a menudo se mezcla con pruebas y lo ha sido desde que llevo en el desarrollo de software. Esto me ha molestado desde hace algún tiempo. En 2009 me encontré conversando con Michael Bolton, Fiona Charles, Lynn McKee, Nancy Kelln y algunos otros. En el curso de la conversación, la idea de que "los evaluadores hagan preguntas que conduzcan a información sobre cómo se comporta o se espera que se comporte el software" siguió flotando. Hacer preguntas nos lleva a información sobre el software, cómo interactuamos con él y, en última instancia, a más preguntas; al menos, más preguntas hasta que todas las preguntas que nos interesan a nosotros y a las partes interesadas hayan sido formuladas y respondidas.

Lo que nos lleva de nuevo al "Probador de software".

Nos gusta el término autor de preguntas, que también puede resultar útil a la hora de planificar sesiones. Consulte la Parte IV, "Prueba del valor empresarial", para obtener más información sobre este tema.

Los equipos en los que Lisa ha trabajado han desdibujado las líneas entre roles.

Algunos programadores son probadores experimentados. A veces es un probador quien

presenta una solución simple a un complicado problema de diseño de código.

Además, las personas con una amplia gama de competencias desempeñan más de un rol. Por ejemplo, el equipo actual de Lisa tiene codificadores que también se encargan de la administración del sistema y tienen tareas operativas.



En los últimos años, términos como DevOps han ganado un uso más amplio, lo que indica la interdependencia del desarrollo de software con la infraestructura y las operaciones. Aunque los términos pueden volverse populares recientemente, realizar el trabajo que normalmente realiza alguien en un rol especializado ha sido un sello distintivo de los equipos ágiles. (Exploraremos la colaboración mutuamente beneficiosa entre DevOps y los evaluadores en el Capítulo 23, "Pruebas y DevOps").

Olvídese de los desarrolladores en la prueba; Necesitamos probadores en desarrollo

Trish Khoo , la Ingeniera de pruebas originaria de Australia, comparte sus experiencias sobre que sucede cuando todo el equipopienso en realizar pruebas.

El año pasado comencé a trabajar en un pequeño equipo con desarrolladores que realmente valoran las pruebas y las tratan como una parte integral del ciclo de desarrollo. Nunca olvidaré una de nuestras primeras reuniones de planificación cuando estábamos discutiendo una función que estábamos a punto de crear y un desarrollador frunció el ceño y dijo: "Sí, pero ¿cómo vamos a probarla?". Como resultado, cambiaron todo el diseño.

Creo que estuve a punto de desmayarme del shock, ya que nunca había escuchado eso en mi carrera. La parte clave de esto fue que no era el equipo el que me preguntaba a mí, el evaluador, cómo iba a probarlo. Fue una pregunta que se hizo a todo el equipo: "¿Cómo vamos a probarlo como equipo? ¿Cómo podemos construir esto para que tengamos confianza en que funcionará mientras lo construimos?"

A medida que construímos funciones, los desarrolladores siempre escribían pruebas (desde pruebas unitarias hasta pruebas a nivel de navegador) y pedían a otro desarrollador que hiciera algunas pruebas manuales antes de que me las pasaran para que las probara en un entorno totalmente integrado. Cuando me lo entregaron de esta manera, rara vez encontré errores debidos a un descuido. La mayoría de los errores que encontré se debieron a escenarios de usuario o escenarios del sistema en los que no se había pensado antes.

Entonces podrías pensar, como tester, ¿realmente tenía mucho que hacer en un equipo como este? Mi activo más valioso en este proceso seguía siendo la forma en que pensaba sobre el producto desde el punto de vista de las pruebas, desde el punto de vista del usuario. Lo que descubrí fue que mi experiencia en pruebas se volvió menos valiosa al final del ciclo y mucho más valiosa al principio.

Cuanto más esfuerzo puse en probar el producto conceptualmente al inicio del proceso, menos esfuerzo tuve que poner en probar manualmente el producto al final porque, como resultado, surgirían menos errores.

Sólo quiero dividir esa última parte en una cita grande con una fuente elegante porque creo que es bastante importante. Cuanto más Mientras menos esfuerzo puse en probar el producto conceptualmente al comienzo del proceso, tanto menor esfuerzo pongo en probar el producto al final.

Pero la parte clave de esto fue que sabía que el equipo de desarrollo estaba probando el producto de manera efectiva al pensar en las pruebas, escribir pruebas y probarlas manualmente a medida que se desarrollaba el producto.

Podría confiar en que si hubiéramos pensado en un escenario durante la planificación, se desarrollaría y probaría de manera efectiva con pruebas de regresión automatizadas al final del proceso.

Este año he tenido muchas discusiones sobre el papel del evaluador. Dejemos eso de lado por ahora y comencemos a pensar en el papel de un desarrollador de software. Un desarrollador de software debe poder crear un producto con la confianza de que hará lo que se espera que haga. Saber cómo hacerlo a un nivel básico debería ser fundamental para el papel de un buen desarrollador de software. Por esa razón, necesitamos más pruebas en el desarrollo de software. Y debe ser realizado por las personas que crean el producto.

Tener un especialista en pruebas en el equipo es un activo valioso, pero la responsabilidad de las pruebas no debe limitarse a una sola persona.

Es posible que también tenga un especialista en bases de datos en su equipo, pero eso no significa que sea la única persona que trabaja con bases de datos. Lo mismo ocurre con las pruebas. El especialista puede ayudar con los problemas de prueba realmente difíciles, sabiendo que el resto del equipo es capaz de abordar los problemas de prueba simples.

Entonces se trata de ciclos de retroalimentación más cortos, mayor confianza y lanzamientos de calidad más rápidos para todos. ¿A quién no le encanta eso?

Exploramos la interacción entre los equipos de clientes y desarrolladores en Agile Testing. Desde entonces, muchos equipos ágiles han incorporado especialistas con diferentes competencias. Por ejemplo, ahora es más común encontrar analistas de negocios en un equipo ágil, así como evaluadores que realizan muchas actividades de análisis de negocios. Los límites entre los roles siguen difuminándose. Al mismo tiempo, crear un equipo multifuncional no significa prescindir de especialidades. Según nuestra experiencia, los equipos a menudo se sienten bloqueados porque les falta un conjunto de habilidades en particular.

A veces, la solución es simple: contratar a alguien con esas habilidades o encontrar formas de capacitar a los miembros existentes del equipo.

La historia de Lisa

En mi último trabajo, mi equipo se frustró porque comenzamos una nueva iteración con varias historias que no estaban bien definidas por los expertos en negocios de la empresa matriz. Como resultado, íbamos constantemente al propietario del producto para aclarar los requisitos o para mostrarle lo que habíamos desarrollado, sólo para que nos dijeran que estaba mal.

Los evaluadores sugerimos contratar a un analista de negocios (BA) que pudiera trabajar con el propietario del producto para ayudar a los clientes a articular mejor lo que necesitaban.

Nuestro propietario de producto no estaba familiarizado con la nueva empresa matriz y, aunque se reunió con las partes interesadas, no sabía qué preguntas correctas debía hacer para llegar al núcleo de las funciones que querían. Un BA capacitado sabría cómo colaborar con los clientes y hacer las preguntas correctas.

Desafortunadamente, el gerente no estaba dispuesto a contratar un licenciado en Letras, por lo que formamos una comunidad de práctica de análisis empresarial. Los evaluadores, el propietario del producto, el ScrumMaster, el gerente de desarrollo y los programadores interesados se reunieron e identificaron formas de desarrollar nuestras habilidades de análisis. Leímos libros y artículos, asistimos a talleres de conferencias y participamos en seminarios web para adquirir competencia en análisis empresarial. Nos reunimos a intervalos regulares para compartir información y la documentamos en la wiki del equipo.

Quizás hubiera sido mejor contratar a un experto, pero nuestros esfuerzos dieron sus frutos. El propietario del producto aprendió técnicas para utilizar cuando se reunía con las partes interesadas de la empresa matriz, por lo que comprendió mejor las características deseadas. A veces todavía nos faltaba contexto sobre los problemas comerciales que los clientes intentaban resolver, pero ya no pasábamos tanto tiempo yendo y viendo con el propietario del producto para obtener información sobre las historias que estábamos desarrollando.

Conjunto de habilidades en forma de T

En Agile Testing, describimos diez principios para los evaluadores ágiles, que enfatizaban la actitud y la mentalidad por encima de las habilidades técnicas específicas. Como repaso rápido, son:

- Proporcionar retroalimentación continua.
- Entregar valor al cliente.
- Permitir la comunicación cara a cara.
- Tenga coraje.
- Manténgalo simple.
- Practicar la mejora continua.

- Responder al cambio.
- Autoorganizarse.
- Centrarse en las personas.
- Disfrute.

Sin embargo, todavía escuchamos la pregunta: "¿Los evaluadores de equipos ágiles necesitan ser programadores?"

Nuestra respuesta es que los evaluadores necesitan habilidades en forma de T (ver Figura 3-1), un término definido por primera vez por David Guest (Guest, 1991). Para trabajar eficazmente en cualquier equipo, necesitamos conjuntos de habilidades amplios y profundos. Un amplio conocimiento en áreas distintas a nuestra propia especialidad nos permite colaborar en disciplinas con expertos en otras funciones. El conocimiento profundo y la amplia práctica en un solo campo garantizan que aportemos algo esencial al equipo. Rob Lambert tiene una buena publicación en su blog sobre este tema (Lambert, 2012).

La parte superior de la "T" de los evaluadores generalmente incluye habilidades técnicas como una comprensión básica de la arquitectura de su sistema, conocimiento de conceptos generales de programación y principios de diseño, capacidad para realizar consultas básicas de bases de datos y competencia con herramientas tales como entornos de desarrollo integrados (IDE) y paneles de control de integración continua (CI).

Los miembros del equipo que desempeñan otros roles necesitan conceptos básicos de prueba, entre otras habilidades en su barra T. El conocimiento superficial del dominio puede ser adecuado en

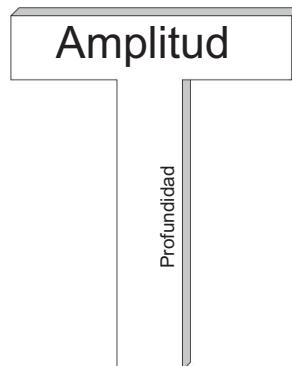


Figura 3-1 Conjunto de habilidades en forma de T

algunas situaciones. En otros, la capacidad de ofrecer valor empresarial requiere que algunos miembros del equipo, tal vez evaluadores o analistas empresariales, necesiten conocer el negocio en profundidad. Reúna a todo el equipo para hablar sobre las habilidades en forma de T y cómo llenar los vacíos. Y no olvide esos diez principios para evaluadores ágiles. La actitud realmente lo es todo.

Equipo en forma cuadrada

Adán Caballero , a Director de Control de calidad y soporte en el Reino Unido, acciones su historia sobre su experiencia con Miembros del equipo en forma de T para hacer una equipo de forma cuadrada.

Mi equipo y yo llevamos siete años trabajando en probar un sistema de almacenamiento de Big Data. El sistema es un sistema de almacenamiento de datos a gran escala combinado con un motor de consultas SQL que se ejecuta en varios sistemas operativos, principalmente Linux. Uno de los principales problemas que he encontrado al probar un producto de esta naturaleza es la variedad de habilidades relevantes que se requieren para realizar todas las tareas necesarias para probar el sistema. Los evaluadores de nuestra empresa necesitaban la capacidad no solo de probar el producto desde la perspectiva de las distintas partes interesadas, sino también de construir y mantener los diversos entornos y arquitecturas de prueba que se necesitaban. Algunas de las habilidades que necesitábamos eran

- Conocimiento de sistemas operativos en Linux/UNIX para crear y mantener Mantener entornos de prueba y monitorear esos entornos para evaluar el impacto del software sobre ellos.
- Virtualización y conocimiento de la nube para ampliar las pruebas a entornos que podrían soportar los entornos de múltiples máquinas agrupadas
- Conocimiento de secuencias de comandos para desarrollar y mantener continuamente los numerosos arneses necesarios para probar el producto a través de las herramientas de línea de comandos del servidor.
- Conocimiento de programación para desarrollar y mantener los arneses necesarios para las pruebas funcionales y de escalabilidad de las interfaces API del cliente cuando diferían de nuestros lenguajes de programación de productos principales C y C++.
- Conocimiento de SQL y bases de datos para comprender el dominio de implementación y probar el extenso motor SQL con una variedad de consultas realistas.
- Habilidades de pruebas exploratorias para identificar y ejercitarse la variedad de estados y combinaciones de operaciones que pueden afectar la capa de almacenamiento de datos.

Muy rápidamente se hizo evidente que era poco probable que encontráramos tal riqueza de habilidades en un solo individuo. En lugar de eso, adopté el enfoque de tratar de poblar el equipo con una cantidad de evaluadores que, en combinación, poseyeran la variedad de habilidades que permitirían al equipo abordar los múltiples desafíos que enfrentaba. Cada individuo necesitaría tener un conjunto general de habilidades para comprender el producto y el enfoque general de prueba.

A medida que incorporamos nuevos miembros al equipo, nos aseguramos de que cada uno poseyera un conjunto profundo de habilidades, que incluían las habilidades prioritarias que el equipo había identificado. Estas habilidades eran complementarias a las de los demás miembros del equipo y a las del equipo en su conjunto.

Cuando leí por primera vez sobre el concepto del individuo "en forma de T", resonó con lo que estábamos haciendo. La idea de un amplio conjunto de habilidades generales combinadas con un profundo núcleo de habilidades especializadas era exactamente la forma del individuo que habíamos intentado reclutar. Entonces, por ejemplo, ya tuve mucha suerte de trabajar con un evaluador que poseía sólidos conocimientos de bases de datos y lenguaje SQL combinados con conocimientos de dominio de una función anterior como administrador de bases de datos (DBA).

Empleamos a una persona con sólidas habilidades de programación para mantener el armés central, respaldado con excelentes habilidades de pruebas exploratorias.

Otro tenía grandes capacidades de sistema operativo para crear entornos de prueba, clústeres virtuales y pruebas de Hadoop y de la nube, combinados con conocimientos de pruebas de rendimiento y remoto. La Figura 3-2 muestra cómo cada evaluador tiene diferente profundidad de habilidades con la misma amplitud.

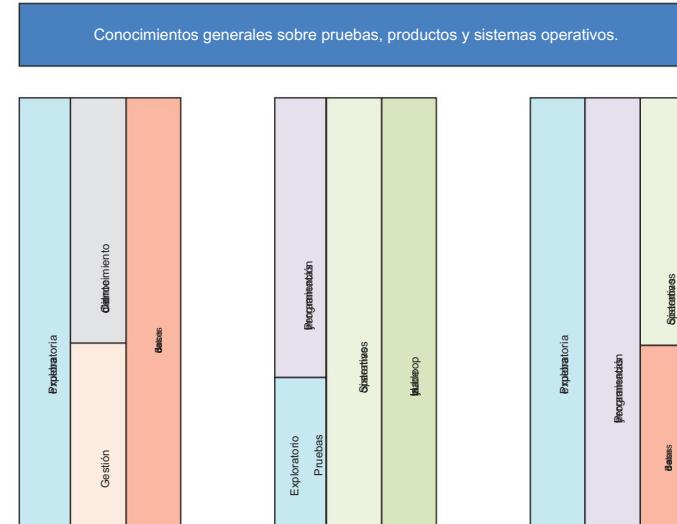


Figura 3-2 Tres probadores con diferentes puntos fuertes

Sentí que al concepto en forma de T le faltaba una de las razones principales por las que adoptamos el enfoque que adoptamos. El concepto de individuo en forma de T se limita exactamente a eso: un individuo. Lo que sentí fue que el verdadero poder del evaluador en forma de T fue cuando esos individuos combinaron sus habilidades en un equipo de la manera que lo habíamos hecho nosotros, un concepto que denominé "equipo en forma cuadrada", como se muestra en la Figura 3.-3.

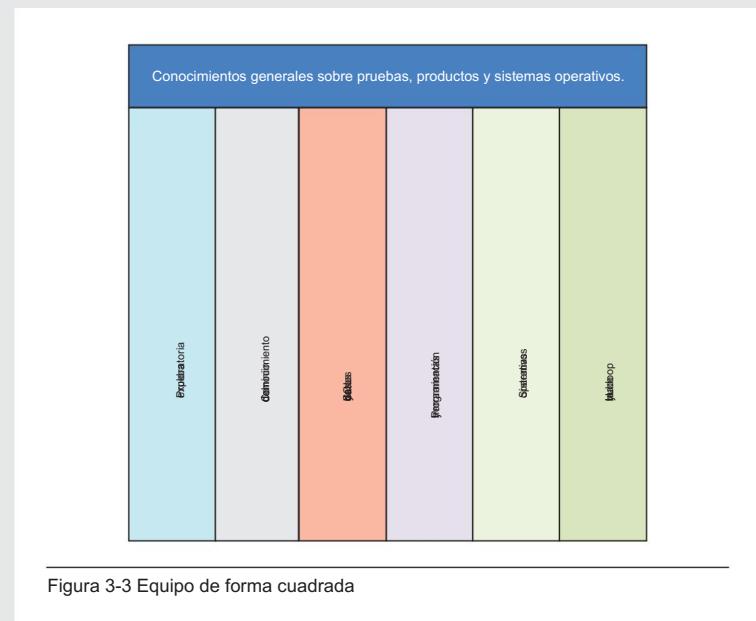


Figura 3-3 Equipo de forma cuadrada

Cada evaluador que hemos contratado ha aportado habilidades únicas al equipo. Algunas de estas habilidades sabíamos que eran una prioridad cuando estábamos en el proceso de contratación; otros eran menos evidentes en ese momento, pero seguían siendo válidos. Por ejemplo, un evaluador que contratamos era una opción menos obvia, ya que tenía experiencia en consultoría trabajando en proyectos de implementación. Sin embargo, esto significó que sus habilidades en informes y análisis de requisitos lo convirtieron en una excelente incorporación para comprender las preocupaciones de las partes interesadas y definir los criterios de aceptación en consecuencia. También utilicé sus excepcionales informes de prueba como ejemplo para el resto del equipo. Si hubiésemos observado a esa persona de forma aislada, es posible que no se la hubiera considerado para desempeñar un papel de prueba en el producto. Al adoptar un enfoque holístico para formar el equipo, pudimos integrar bien a los nuevos miembros y todos nos beneficiaríamos de la perspectiva única que tenía cada uno.

Al igual que Adam, creemos que es importante tener en cuenta las habilidades del equipo además de las individuales. Juntos, todo el equipo puede resolver casi cualquier cosa. No tenga miedo de mirar a su alrededor y utilizar el conocimiento que otras personas aportan.

Especialistas generalizadores

El desarrollo ágil atrae a especialistas generalizadores. Este término se ha utilizado para referirse a personas con un profundo nivel de conocimiento en al menos un dominio y una comprensión de al menos otro. Hmm, suena mucho a habilidades en forma de T. A veces el término se utiliza para describir a una persona que es buena en todo, pero no es eso lo que queremos decir. Corremos el riesgo de diluir nuestras fortalezas, y en lugar de hacer algunas cosas bien, hacemos muchas cosas con mediocridad. Sin embargo, para colaborar bien con miembros del equipo que desempeñan otras funciones, necesitamos conocer algunos conceptos básicos de su especialidad.

Convertirse en un especialista generalizador

Matt Barcomb, un especialista en diseño organizacional, comparte su ideas sobre cómo un especialista generalizador evoluciona.

Una cosa es saber qué es un especialista generalizador, pero ¿cómo se puede llegar a serlo? La mayoría de las personas dedican mucho tiempo a ser muy buenas en su especialidad, pero no entienden cómo generalizar de manera efectiva sin simplemente convertirse en especialistas en otra área.

Qué Especialista en generalización no es

He estado en varios lugares en los últimos años donde la gerencia se aferraba al término con demasiado entusiasmo. La creencia tentadora pero equivocada era que los programadores, evaluadores, diseñadores, analistas, redactores técnicos, ingenieros de lanzamiento y casi todos los demás (excepto los gerentes) eventualmente sabrían cómo hacer el trabajo de los demás y podrían ser intercambiados, indistintamente, por cualquier otro. otra persona del equipo. ¡Imagínese lo simples que serían las hojas de cálculo de equilibrio de recursos y los planes de proyecto!

Esa situación no es un equipo de especialistas generalizadores; sería un equipo de generalistas generalizadores. También es pura ficción, ya que sería imposible que todas las personas involucradas en el lanzamiento de software empresarial conocieran la función de cada otra persona con la profundidad que sería necesaria para entregarlo de manera efectiva.

Entonces, ¿qué es un especialista generalizador?

Ser un generalista en un equipo multifuncional significa ser capaz de apreciar, comunicarse y colaborar de manera efectiva con otros miembros del equipo y roles con diferentes especialidades. No significa que la persona pueda hacer el mismo trabajo que diferentes especialistas con la misma habilidad o entusiasmo.

Por ejemplo, un programador con un profundo conocimiento del código no es intercambiable con un tester y viceversa. Sin embargo, ambos podrían colaborar, como pareja, en las tareas de cualquiera de las personas. Idealmente, la diferencia es obvia y el ejemplo podría aplicarse fácilmente a cualquier otro rol del equipo, como probador, diseñador, analista, operaciones, redactor técnico o incluso gerente.

Cómo Convertirse en un Especialista en generalización

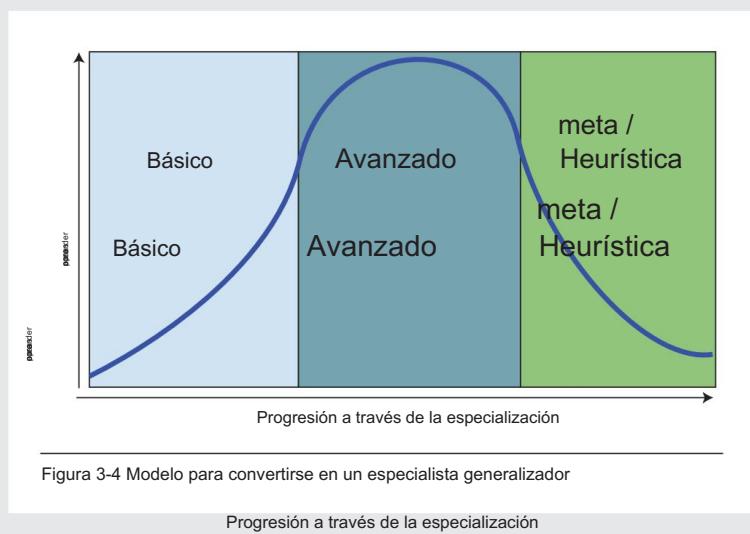
Existen muchos enfoques para convertirse en un especialista generalizador.

No existe una única forma correcta de aprender y, en última instancia, convertirse en un especialista generalizador se trata de aprender.

Considero que tener forma de T es un problema de adquisición de habilidades. Hay muchos modelos para la adquisición de habilidades, pero con el fin de convertirme en un especialista generalizador utilizo tres categorías: (1) Básico, (2) Avanzado y (3) Meta, combinados con una cantidad asociada de "cosas para aprender".

La cantidad de cosas que aprender varía según la categoría; Básico tiene una cantidad pequeña, Avanzado tiene la cantidad más significativa y finalmente Meta tiene una cantidad menor, pero más que Básico.

Esto podría parecerse al modelo que se muestra en la Figura 3-4.



En realidad, este modelo muestra cómo convertirse en un especialista. Sin embargo, también ofrece sugerencias sobre cómo convertirse en un generalista. Según el modelo, todo principiante debe aprender los conceptos básicos de una habilidad o especialidad (como pruebas o programación). Esto es igualmente cierto para los generalistas o los aspirantes a especialistas.

Entonces, el primer paso para convertirse en generalista es aprender los conceptos básicos de la especialidad. Para un evaluador de un equipo multifuncional, esto podría significar la conciencia técnica a la que Lisa y Janet se refieren en el Capítulo 5, "Conciencia técnica". Por ejemplo, los evaluadores pueden aprender más sobre el proceso de implementación o los entornos de destino; podrían aprender cómo consultar una base de datos, o sobre la sintaxis, las estructuras y las herramientas que utilizan los programadores para desarrollar el producto.

Luego, el especialista pasa años (a veces la mayor parte de su carrera) aprendiendo la materia avanzada de su especialidad e incluso más tiempo para obtener el metaconocimiento sobre la especialidad. Obviamente, hacer que todos los miembros de un equipo multifuncional dediquen tanto tiempo a aprender una especialidad no es la forma más eficaz de convertirse en un especialista generalizador. Entonces la pregunta es: ¿Cómo puede un individuo continuar creciendo como generalista pero saltarse los años necesarios para progresar desde Avanzado hasta Meta?

La respuesta a la pregunta radica en comprender la naturaleza de las Metahabilidades derivadas. El metaconocimiento de una especialidad es ese conocimiento intuitivo y tácito que uno desarrolla después de años de aplicar conocimientos avanzados de una especialidad en muchas situaciones diferentes: casi un sexto sentido dentro de un dominio. Un especialista podría describir algo como "simplemente no sentirse bien" o la "forma" o el "olor" de algo como el código, el diseño o la arquitectura de un producto. La cuestión es que hay algo que el especialista está intuyendo, alguna señal que debe buscar.

Estos signos se pueden enseñar a los no especialistas como heurísticas o reglas generales y son la siguiente forma en que un generalista puede crecer. Un generalista puede colaborar con un especialista y ofrecerle ayuda valiosa para la detección de señales de posibles problemas. Puede que la heurística no siempre sea correcta, pero está bien; así es como funcionan las reglas generales. Entonces, si bien puede llevar años desarrollar saber cómo aplicar correctamente el metaconocimiento en una situación determinada, a los generalistas se les puede enseñar las señales heurísticas que deben buscar.

Algunos ejemplos de esto para un evaluador que generaliza hacia la programación podrían ser las reglas del código limpio (Martin, 2009), el principio de DRY (no te repitas), la longitud o el recuento de clases, variables, argumentos de función o también muchas declaraciones if en un método. El generalista no necesitaría saber cómo solucionar estas cosas y, a veces, puede haber buenas razones por las que se está infringiendo una regla. El valor agregado es ayudar al especialista con la detección de señales, no con la corrección de problemas.

La razón por la que esto es valioso es porque los individuos tienden a involucrarse en un nivel más bajo o de una manera más enfocada y analítica al ejecutar una tarea o implementar algo. Aplicar metaconocimiento o buscar heurísticas requiere una visión de nivel superior o una forma más abstracta de pensar sobre la tarea en cuestión. Es difícil para una sola persona realizar una tarea simultáneamente de forma analítica y abstracta.

Entonces, cuando los generalistas aprenden los conceptos básicos y, lo que es más importante, las heurísticas de una especialidad, pueden comunicarse más eficazmente con esos especialistas. Este es un primer paso muy importante para tener un equipo multifuncional eficaz. Pasar de la comunicación a una verdadera colaboración es clave para un equipo multifuncional más maduro.

El concepto de especialistas generalizadores se aplica a todos los roles del equipo, no solo a los evaluadores. Cuando los programadores aprenden los conceptos básicos de las pruebas, pueden comunicarse más eficazmente con los evaluadores y aprender mejores formas de prevenir defectos y mejorar la calidad a medida que desarrollan el software. Según nuestra experiencia, asociarse con un evaluador es la forma más fácil para que los programadores desarrollen sus conocimientos sobre pruebas.

Los equipos de Lisa se beneficiaron de la documentación de listas de verificación e información de pruebas especializada en la wiki del equipo. Si bien la información contenida en un wiki no sustituye a la conversación cara a cara, puede servir como estimulante de la memoria.

En el Capítulo 6, "Cómo aprender", exploraremos formas en que los evaluadores pueden aprender los conceptos básicos de otras especialidades, como programación y diseño de bases de datos, para que puedan comunicarse más fácilmente con compañeros de equipo que desempeñan otras funciones.

Contratar a las personas adecuadas

Lisa fue influenciada al principio de su carrera ágil por el artículo de Alistair Cockburn "Caracterización de las personas como componentes no lineales de primer orden en el desarrollo de software" (Cockburn, 1999). Después de estudiar docenas de proyectos de software a lo largo de 20 años, descubrió que el factor común de éxito en el desarrollo de software era que "unas pocas personas buenas intervinieron en momentos clave". Tener buena gente hizo que los proyectos tuvieran éxito, no el lenguaje de programación, las herramientas o la metodología utilizadas.



Según nuestra experiencia, es fundamental conseguir personas con la actitud adecuada para el equipo. Tómate el tiempo que necesites para encontrar evaluadores que encajen bien. Si contrata personas que expresen curiosidad, quieran aprender y no tengan reparos en salir de su zona de confort, pueden recibir capacitación para sus necesidades específicas. Aunque la colocación tiene grandes ventajas sobre los equipos distribuidos, considere ampliar su búsqueda geográfica. Los evaluadores que trabajan de forma remota pueden ser eficaces si el equipo es disciplinado en el uso de buenas prácticas de comunicación y aprovecha la tecnología actual.

Hiring Geeks That Fit de Johanna Rothman (Rothman, 2012b) ofrece más ayuda para encontrar y contratar evaluadores con adaptación cultural y las habilidades que su equipo necesita. Cada individuo aporta sus fortalezas y su perspectiva única, y es importante observar todos los aspectos del valor que aporta cada individuo. La diversidad dentro de un equipo es esencial para obtener diferentes perspectivas. A veces, la persona que no encaja bien al principio, pero le apasiona ofrecer un producto de calidad al cliente, aún puede desempeñar un papel valioso.

Probadores de incorporación

Incluso los evaluadores ágiles experimentados necesitan ayuda para encontrar su lugar en un nuevo equipo. Nos resulta útil establecer expectativas. ¿Qué puede esperar saber esa persona al final del primer día? ¿La primera semana? ¿El primer mes? Coloque esta información en una página wiki o en algún otro lugar de fácil acceso y mantenimiento.

Ayude al nuevo empleado a conocer a los miembros de los equipos de desarrollo y de clientes. Proporcione una lista de "quién sabe qué" y presente a la nueva persona a cada una de esas personas. Programe tiempo para que la nueva persona se reúna con expertos en negocios y descubra qué hacen en sus trabajos.

Tome medidas para garantizar que el nuevo evaluador tenga suficiente tiempo para aprender y no se sienta presionado a "producir" de inmediato. Ayude al nuevo empleado a sentirse seguro al pedir ayuda y hacer preguntas en cualquier momento.

Según nuestra experiencia, el emparejamiento es la mejor manera de poner al día a nuevas personas. Empareje al nuevo evaluador con otros evaluadores, con programadores, con BA, con DevOps y con expertos en negocios. Es útil asignar un amigo o mentor como primera línea de apoyo para el nuevo empleado; los dos pueden actuar

como par predeterminado. Hay una ventaja al emparejarse con la nueva persona que proporciona un nuevo par de ojos.

La historia de Lisa

Me estaba emparejando con un evaluador altamente experimentado que era nuevo en nuestro dominio de servicios financieros. Los resultados de una de nuestras pruebas arrojaron unos pocos centavos de diferencia con el resultado esperado. Había visto esto muchas veces a lo largo de los años y lo descarté por una diferencia de redondeo. Pero la discrepancia molestó a nuestro nuevo evaluador. Se acercó a hablar con uno de los programadores al respecto.

Resultó ser un error real que todos habíamos estado ignorando durante años.

Se redactaron pruebas, se solucionó el problema y ya no vimos las discrepancias. Los centavos se suman, así que me siento mal al pensar que algunos de nuestros clientes pueden haber sido ligeramente defraudados. No hay nada mejor que un par de ojos nuevos.

Lección aprendida: ¡no descartes nada que observe un nuevo evaluador!

Sea creativo al planificar la capacitación del nuevo evaluador de su equipo. Programe una sesión con alguien que pueda explicarle la arquitectura del sistema. Programe tiempo para que un programador o administrador del sistema guíe al nuevo evaluador a través del proceso de CI. Proporcionar formación sobre los conjuntos de pruebas automatizados y la documentación viva que estos proporcionan. Las discusiones individuales añaden un valor tremendo a la práctica habitual de leer toda la documentación porque el nuevo empleado tiene la oportunidad de hacer preguntas. Los nuevos miembros del equipo pueden sentirse abrumados por la complejidad de intentar aprender todo a la vez. Intente dividir el entrenamiento en partes más pequeñas. Quizás después de una sesión de capacitación, deles una carta para explorar esa área. Hay más información sobre los estatutos y las pruebas exploratorias en el Capítulo 12, "Pruebas exploratorias".

Mike Talks nos dijo que intenta dedicar de 15 a 30 minutos todos los días durante las dos primeras semanas a ponerse al día con los nuevos empleados. Después de eso, lo reduce a semanal y luego mensual. Pasarán muchos meses antes de que estén al día, así que no intente apresurar el proceso. Todos se benefician de un enfoque reflexivo al incorporar un nuevo evaluador.

Resumen

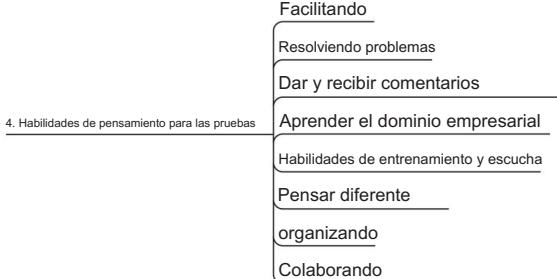
Los equipos que logran crear software de alta calidad incluyen personas en una variedad de roles y con una amplia gama de competencias. Busque formas de conseguir los que necesita en su equipo.

- Equipos cuyos miembros tienen una variedad de conjuntos de habilidades en forma de T, como Continuar con las pruebas en entornos que cambian rápidamente.
- Todos los miembros del equipo necesitan conocimientos básicos amplios sobre pruebas ágiles, que les permitan colaborar bien para mejorar la calidad, pero cada uno puede contribuir con una habilidad profunda y especializada diferente.
- Los evaluadores se comunican mejor con programadores, analistas de negocios, propietarios de productos, gerentes, profesionales de DevOps y otros miembros del equipo cuando conocen los conceptos básicos de esas otras especialidades.
- Contrate miembros del equipo apasionados por la calidad y el aprendizaje, cuyas habilidades en forma de T se complementen entre sí.
- Establecer expectativas realistas para los nuevos empleados y generar visibilidad y comentario.

Esta página se dejó en blanco intencionalmente.

Capítulo 4

Habilidades de pensamiento para exámenes



El Oxford English Dictionary define las habilidades sociales de esta manera: "atributos personales que permiten a alguien interactuar de manera efectiva y armoniosa con otras personas". "Suaves" parece implicar que estas habilidades son más fáciles de aprender o menos importantes que las habilidades "duras", como las competencias técnicas. Hay mucho debate sobre cuál es el mejor término para describir estas habilidades.

corbatas. A algunos les gusta llamarlos habilidades interpersonales. Preferimos el término habilidades de pensamiento porque, además de nuestras relaciones con las personas, estas habilidades se aplican a otras áreas como la resolución de problemas, la comprensión del ámbito empresarial, el uso del estilo de pensamiento correcto para una actividad de prueba determinada y la organización de nuestro tiempo.

Las habilidades de pensamiento no son tangibles en el sentido de que podamos decir: "Lo he aprendido; Puedo practicarlo perfectamente ahora". Habilidades como la comunicación, la colaboración, la facilitación, la resolución de problemas y la priorización pueden ser las más difíciles de dominar, pero son las más cruciales para el éxito en las pruebas ágiles.

En muchas organizaciones, cuando los evaluadores forman parte de un equipo separado, tienden a hablar sólo con otros evaluadores, posiblemente con programadores, pero rara vez con alguien del equipo del cliente. Sin embargo, en equipos ágiles, los evaluadores y otros miembros del equipo trabajan en estrecha colaboración con las partes interesadas del negocio y los propietarios y gerentes de productos para obtener requisitos y descubrir aspectos ocultos.

suposiciones. Los programadores, analistas y miembros del equipo en otras funciones también contribuyen a generar requisitos, lo que ayuda a abordar el impacto de los problemas técnicos y las dependencias. Esta es la razón por la que conceptos como el pensamiento sistémico (cómo llegamos aquí y qué cambios impactan otras partes del sistema) son tan críticos.

Los evaluadores y otros miembros del equipo involucrados en pruebas y actividades centradas en la calidad pueden aplicar habilidades interpersonales y de liderazgo para ayudar a los equipos de desarrollo y de clientes a mejorar su producto y proceso de software.

Facilitando

Actividades como los talleres de especificación (Adzic, 2009) funcionan mejor cuando alguien con experiencia en facilitación ayuda a guiar la discusión. Lo ideal sería un facilitador (consulte la Figura 4-1) que no esté involucrado en capturar los requisitos, pero cualquier miembro del equipo que tenga habilidades de pensamiento clave, como cómo lograr que las personas en diferentes roles trabajen juntas y se mantengan enfocadas en un objetivo común, puede intensificarse si es necesario. Los facilitadores de los talleres de especificación ayudan a las partes interesadas a establecer objetivos comerciales y ayudan tanto a los miembros del equipo de desarrollo como al del cliente a definir de manera colaborativa el alcance que alcanzará esos objetivos.

Habilidades similares le ayudarán a facilitar sesiones informales de lluvia de ideas, asegurando que cada participante se sienta libre de expresar ideas sin ser

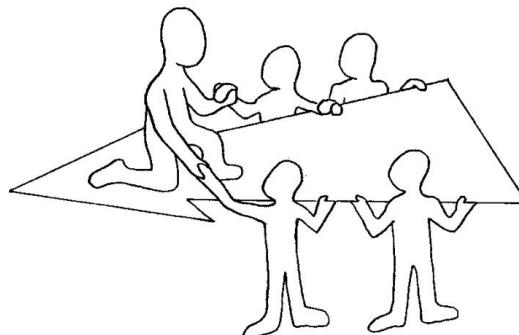


Figura 4-1 La facilitación ayuda a los equipos a lograr un entendimiento común.

criticado. Trabajar para adquirir estas habilidades le ayudará a encontrar formas creativas para que su equipo resuelva los problemas. Consulte la bibliografía de la Parte II, “Aprender para realizar mejores pruebas”, para obtener más libros recomendados que le ayudarán a perfeccionar sus habilidades de facilitación para recopilar requisitos y colaborar de forma eficaz.

Resolviendo problemas

Cuando hablamos de habilidades no técnicas, no queremos decir que sean fáciles de dominar. Por ejemplo, muchos de nosotros trabajamos continuamente para mejorar nuestras habilidades para resolver problemas; Algunos programas universitarios de informática o TI pueden enseñar habilidades como ésta, pero a menudo no están en el plan de estudio. Por lo general, la gente necesita adquirirlos en el trabajo.

La historia de Janet

Recuerdo la primera vez que escribí mi examen para Gerente de Calidad Certificado de ASQ, Reprobé la parte escrita, que era cómo abordar dos problemas específicos. I Creo que fallé porque no recordaba cómo aplicar mis habilidades de resolución de problemas. habilidades. Aprendí esas habilidades por primera vez cuando tomé Física 101 en la Universidad de Alberta, pero no los apliqué con regularidad. Cuando reprobé mi examen, Me senté, descubrí la causa raíz y volví a lo básico revisando cómo resolver un problema. En física, eso comenzó con un dibujo. Reescribí mi examen siguiendo los principios de resolución de problemas y lo aprobé. Luego presenté lo que funcionó para mí al siguiente grupo de personas que Quería escribir el examen, lo que reforzaba lo que había aprendido.



La resolución de problemas es una de esas habilidades transferibles que se pueden aplicar al diseño de pruebas, la depuración, el entrenamiento o la enseñanza. Quizás la habilidad de pensamiento más útil sea saber cómo ayudar a su equipo a abordar sus problemas, en lugar de intervenir y solucionar los síntomas. Cursos como Liderazgo en resolución de problemas (PSL) (Derby et al., 2014) son una buena manera de aprender cómo su equipo puede replantear problemas, resolver conflictos y comunicarse de manera más efectiva. No es necesario ocupar un puesto directivo para brindar liderazgo a su equipo y ayudarlos a mejorar su eficacia en la resolución de problemas.

Herramientas que nos ayudan a visualizar nuestro pensamiento, como mapas mentales, mapas de impacto (consulte el Capítulo 9, “¿Estamos construyendo lo correcto?”) y

Las herramientas de análisis de causa raíz son buenas adiciones a un conjunto de herramientas para realizar pruebas. Los "5 porqué" (Wikipedia, 2014a) utilizan una técnica iterativa de preguntas para explorar las causas fundamentales de un problema.

Los diagramas de espina de pescado o de Ishikawa (Wikipedia, 2014h) se pueden utilizar para prevenir defectos y para identificar riesgos y posibles dificultades.

Después de generar ideas en una sesión de lluvia de ideas, utilice técnicas como diagramas de afinidad o mapas de impacto (consulte la Figura 4-2) para organizar los conocimientos y los experimentos potenciales. Pruebe diferentes herramientas de pensamiento para ayudar a obtener requisitos y ejemplos de los clientes. Dibujar en una pizarra física o virtual mejora la comunicación y la creatividad. Pruébelo la próxima vez que su equipo se reúna para identificar impedimentos y realizar experimentos para eliminarlos.

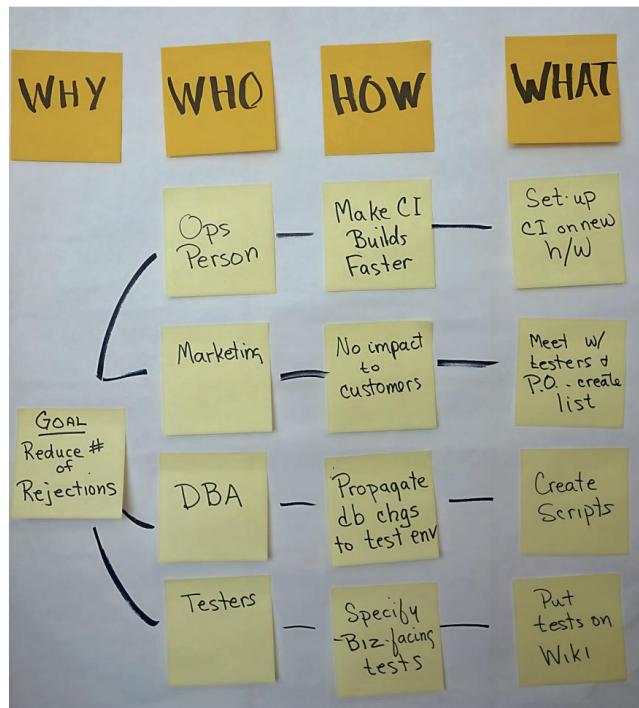


Figura 4-2 Ejemplo de mapa de impacto

Dar y recibir comentarios

Dar y recibir comentarios es casi una forma de arte. Idealmente, la intención del donante es ayudar en el aprendizaje del receptor y hacer crecer la relación entre ellos. Aprendimos de Ellen Gottesdiener que la retroalimentación puede en realidad decir más sobre quien da que sobre quien recibe. La persona que da retroalimentación la basa en sus percepciones y está determinada por las emociones de la persona en el momento de darla. Dado que tendemos a dar retroalimentación sobre lo que nos importa a nosotros y no sobre lo que más le importa a la otra persona, es fácil pasar por alto el contexto. Nuestro tono, las palabras que elegimos y nuestra comunicación no verbal pueden oscurecer nuestro mensaje.

Tenga esto en cuenta cuando sienta la necesidad de comunicar sus observaciones a otra persona. Piense en cómo le gustaría recibir la misma retroalimentación. Saber mantener el foco en el trabajo, no en la persona, es fundamental. La notificación de errores es una forma de proporcionar comentarios, pero normalmente no es la más eficaz. Se necesita tiempo y práctica para aprender a involucrar a los colegas en un intercambio positivo cuando se habla de temas negativos. Por ejemplo, “Leí esto para decir... ¿Podría cambiarse para que tenga más significado? suena mejor que: “Debes cambiar esto....”



El entrenamiento “Feedback Wrap” de Jurgen Appelo (Appelo, 2013) describe formas de brindar comentarios constructivos que ayuden a generar confianza dentro del equipo. Como señala Jurgen, dar retroalimentación se vuelve más difícil ya que muchos equipos tienen menos tiempo cara a cara con sus compañeros de trabajo, debido a los equipos distribuidos, el trabajo remoto y los horarios flexibles. Señala que la retroalimentación escrita, realizada de manera honesta y amigable, permite pensar con más atención y presentar observaciones y sentimientos de una manera más equilibrada. Se puede hacer de forma rápida y frecuente. Experimente con diferentes formas de brindar comentarios oportunos a los miembros del equipo y siga inspeccionando y adaptando su proceso de comentarios.

La empatía es esencial para proporcionar una buena retroalimentación. Piense en cómo le gustaría recibir información no deseada. Toastmasters International, una organización sin fines de lucro para hablar en público, enseña habilidades para realizar evaluaciones de discursos, y estas son transferibles para brindar retroalimentación a los equipos de software. Dar retroalimentación y proporcionar información son parte de las pruebas y hay muchas oportunidades para practicar. En

colaboración con las partes interesadas, identificar formas de medir el éxito que podría tener una nueva característica. Haga que las pruebas sean visibles y transparentes. Si sus clientes siempre conocen el estado actual del equipo y confían en que se les informará sobre cualquier riesgo o problema, serán más receptivos a cualquier noticia (buena o mala) que tenga que darles. Los evaluadores aprenden formas constructivas de brindar retroalimentación, como mostrar defectos a los programadores, sin ofender ni herir sentimientos. Esta sensibilidad se aplica cada vez que entrega comentarios.

Hemos hablado mucho sobre dar retroalimentación. ¿Qué tal recibirlo?

Esfuércese por comprender de qué se trata el mensaje que se transmite antes de sacar conclusiones precipitadas. Agradezca a las personas que le brindan comentarios por sus comentarios honestos. Haga preguntas para aclarar. Con demasiada frecuencia nosotros, como receptores, captamos la emoción y la entrega en lugar del mensaje. Los receptores también pueden tener experiencias previas que podrían hacer que escuchen algo diferente de lo que realmente se dijo. Escuche para aprender. Consulte la bibliografía de la Parte II para obtener lecturas recomendadas para aprender buenas técnicas de retroalimentación.

Aprender el dominio empresarial

El conocimiento del dominio es un ejemplo de una habilidad que puede ser parte de su amplio conjunto de habilidades o parte de su especialidad: profunda y exhaustiva. Los equipos con un conocimiento amplio y profundo del negocio pueden ayudar mejor a las partes interesadas a priorizar funciones, simplificar soluciones o incluso ofrecer alternativas además del nuevo software para resolver un problema.

La historia de Lisa

Aprender de primera mano cómo los clientes utilizan un producto de software nos ayuda a hacer un mejor trabajo a la hora de ofrecerles el valor adecuado. En mi equipo actual, probadores Manejar la atención al cliente por correo electrónico, con la ayuda de los programadores. Nosotros También supervise el foro comunitario de nuestro producto. Hemos tenido que practicar nuestro paciencia y tacto para poder hacer buenas preguntas, escuchar las respuestas, y establecer una relación con los usuarios para que se sientan libres de compartir sus comentarios. Nosotros Aprenda de primera mano cómo nuestros clientes usan nuestro producto y obtenga comentarios valiosos sobre qué características les proporcionarían el mayor valor. Usamos nuestro juicio para decidir cuándo es necesario escalar un problema o una solicitud de función. A menudo nos encontramos con escenarios que proporcionan casos de prueba o pruebas útiles. cartas para que las utilicemos mientras probamos nuevas funciones.

He tenido mucha experiencia trabajando en atención al cliente para un software. empresa de productos. Comprender el producto desde la perspectiva de los usuarios. ha demostrado ser valioso para ayudar a construir la calidad que desean.



Saber cómo funciona la empresa le permite explorar el software de la misma manera que lo utilizarán los usuarios finales reales. Esto no solo evita que los errores entren en producción, sino que los evaluadores y otros miembros del equipo con conocimiento del dominio también pueden brindar a los expertos empresariales ideas para nuevas funciones.

Aprendiendo el dominio con ayuda del Call Center

Mike habla , Probador de software de Nueva Zelanda, comparte una experimentar el había colaborado con el personal del call center y aprendizaje más sobre cómo usaron su producto.

Hace unos años, trabajé para un equipo diverso pero muy centrado en la entrega. en un banco. Lo extraordinario de esta situación fue que tenía acceso a todos los que estaban conectados al producto.

Lo que resultó realmente útil fue tener acceso al personal del centro de llamadas. Me dieron una introducción práctica sobre cómo usaban el producto día a día. Algunas características y comportamiento me parecieron un poco sorprendentes, pero me llevaron a conocer cómo funcionaba realmente el sistema. I Aprendí a demostrar ese conocimiento a gerentes de marketing, analistas de negocios y propietarios de productos.

Fue genial estar ubicado para que este tipo de relaciones pudieran ser construido. La relación con el call center fue realmente algo importante. A veces les mostraba el comportamiento en las próximas compilaciones, pero de igual manera me mostrarían cualquier "rareza" en la producción. parece llamar El personal del centro tiene una mentalidad diferente a la de los evaluadores, en el sentido de que si algo sale mal, no quieren volver a tocarlo.

Fue un equipo realmente gratificante porque sentí que todos estábamos contribuyendo. hacia una meta, sin barreras para la comunicación.

Habilidades de entrenamiento y escucha

Un miembro del equipo con buenas habilidades de entrenamiento está en mejores condiciones de ayudar a los miembros del equipo menos experimentados. Dado que todos los miembros de un equipo ágil participan en actividades de prueba, es mucho más valioso guiar a otros en la solución de sus propios problemas de calidad y pruebas que simplemente darles las respuestas.

Contar historias de su propia experiencia es una manera poderosa de demostrar cómo se puede hacer algo. Despersonaliza las críticas y permite que las personas recuerden cómo manejaste un problema específico; pueden aprender a adaptar su solución a su contexto. Piensa en cómo has manejado ciertas situaciones y cómo puedes aplicar esa experiencia a tu contexto actual. Contar historias no es algo natural para muchas personas; se necesita práctica. Consulte la bibliografía de la Parte II para obtener referencias sobre coaching.

Observar y escuchar (ver Figura 4-3) son habilidades críticas de comunicación y colaboración. ¿Alguien se queja? Quizás sea una queja legítima. ¿Un compañero de equipo tiene una idea pero se siente demasiado tímido para hablar? Bríndale una cómoda oportunidad de compartir contigo tomando un café.

Saber cuándo y cómo escuchar ayuda al equipo a crecer y mejorar. Como

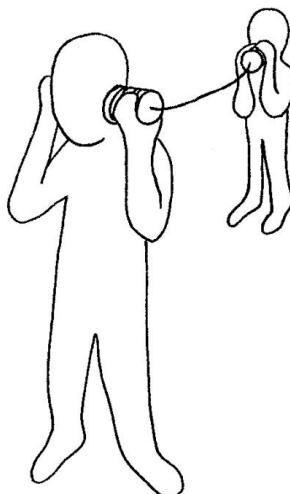


Figura 4-3 Escuchar: tratar de comprender

Como ha señalado Naomi Karten (Karten, 2009), escuchar es un componente que a veces se pasa por alto para ser un buen colaborador.

Pensar diferente

Hemos discutido varias habilidades de pensamiento que encontramos útiles cuando participamos en actividades de prueba. Sea consciente del estilo de pensamiento que adopta para cada situación. La siguiente historia explica algunas herramientas y estilos de pensamiento.

Pensando para probar

Sharon Robson , a Líder de práctica de pruebas de software de fuera-
tralia, compartió sus experiencias con personas encargadas de
formas en qu^{edas} exámenes maximizan su tiempo aplicando habilidades básicas de pensamiento.

Pensar es una habilidad que se puede aprender y mejorar. Como ocurre con cualquier habilidad, La práctica y la exposición a nuevas técnicas y enfoques pueden perfeccionar un talento innato u orgánico. Aprendiendo y practicando nuevas formas del pensamiento, se pueden perfeccionar las habilidades básicas y exponer nuevos niveles de habilidades. Todos pensamos, pero pensando de manera constructiva, con un propósito y la meta de lograr los resultados deseados es un conjunto de habilidades en constante evolución.

Hay muchas formas diferentes de pensar, como el pensamiento crítico. (examinar la exactitud de una afirmación), pensamiento analítico (descomponer el tema en sus componentes y considerarlos y sus relaciones) y el pensamiento creativo (sintetizar nuevos conocimientos a partir de datos existentes). Los evaluadores deben reconocer la habilidad necesaria en cada uno situación y elegir las herramientas que les ayudarán a obtener el resultado deseado.

También hay muchas herramientas de pensamiento diferentes disponibles, como como cuestionamiento socrático (Wikipedia, 2014k), descomposición funcional, priorización, comparación y contraste, diagramas de Ishikawa para el enfoques más críticos o analíticos, así como el pensamiento creativo enfoques como la lluvia de ideas, los mapas mentales, la elaboración y mapeo de relaciones.

En Pensar 2009, Aragonio analiza Aprendiendo el concepto de Modo L (lineal y lento) y Modo R (no lineal, rápido, modo "rico") y cómo emplear estos modos cuando sea necesario. Herramientas como trabajar para definir SMART (específico, medible, alcanzable, relevantes y con un límite de tiempo) ayudan a las personas a darse cuenta de qué modo necesita estar y luego emplear otras herramientas para pasar al modo correcto para maximizar el pensamiento efectivo. Una de las herramientas más poderosas de Hunt es

llama la "mente del principiante"; sigue preguntando: "¿Qué pasaría si...?" . . ?" ¡mucho! Caza También aboga por tomarse el tiempo para ver y comprender claramente lo que está sucediendo. Articula hábilmente que la información son datos sin procesar. Conocimiento imparte significado a la información y el contexto permite una verdadera comprensión. Como evaluadores, debemos centrarnos en obtener una comprensión verdadera. y no detenernos en la información o el conocimiento. Gran parte de la percepción de una persona se basa en la predicción, y la predicción se basa en el contexto y experiencia pasada. Desde el punto de vista de un evaluador, esto es muy poderoso. ya que nuestras percepciones podrían nublarse y no reflejar la realidad. Probadores necesitan usar sus habilidades de pensamiento para ir más allá de la pura percepción hacia resultados basados en evidencia.

La clave es reconocer el tipo de pensamiento requerido y tomar la decisión Esfuerzo por adoptar el modo de pensamiento correcto para satisfacer las necesidades de la situación. Hay una variedad de trabajos sobre el pensamiento que cubren estos diversos modos de pensamiento. Daniel Kahneman (Kahneman, 2011), analiza el Sistema Pensamiento, Rápido y Lento , 1 (intuitivo, rápido) y el Sistema 2 (sistématico, lento) y habla de aspectos que son importantes para los evaluadores. como reconocer el sesgo cognitivo o el uso de heurísticas como riesgo, y luego métodos que se pueden utilizar para pasar del Sistema 1 al Sistema 2 para cuantificar y establecer evidencia de la información proporcionada. por el Sistema 1.

Edward de Bono (de Bono, 1998), en herramientas Pensando para Acción , discute el que pueden usarse para mejorar la forma en que los evaluadores miran y consideran la información que se les presenta. Técnicas como la OPV (otras puntos de vista de las personas) y las burbujas lógicas (asumiendo que todos son muy inteligentes y que piensan y actúan lógicamente desde sus perspectivas) pueden Permitir a los probadores identificar cuándo y dónde pueden ocurrir defectos. Profesor de Bono también utiliza el enfoque de "plantear una hipótesis y luego probarla": haga una suposición y luego encuentre la evidencia para probar o refutar la suposición. Se centra mucho en la mentalidad para realizar pruebas en los tipos de herramientas de recopilación de información que presenta: verificar (respuestas sí/no) versus explorar (hacer preguntas abiertas).

Elisabeth Hendrickson (Hendrickson, 2013) utiliza algunas herramientas fabulosas para un pensamiento estructurado y centrado en . Ella [Explica](#) sobre el uso diagramas y mapas para considerar la solución desde diferentes aspectos, y las heurísticas de "siempre/nevera", "invertir el resultado" y "sustantivos y verbos" para enfocar la mente que prueba en las cosas correctas. Estos Las herramientas son muy útiles para cambiar la forma en que pensamos, que a menudo Tener que hacer.

Dan Ariely (Ariely, 2008) analiza las decisiones que tomamos y cómo, como Los humanos tendemos hacia la posición predeterminada. Como probadores es importante

que reconozcamos la posición por defecto y controlemos nuestras tendencias y aquellos de las personas que utilizan nuestras soluciones para dejarse influir (justa o injustamente) por las opciones que se les presentan. Esto puede manifestarse en muchos maneras, incluso sesgos cognitivos (estamos de acuerdo con hallazgos/evidencias que respaldan nuestra posición y podemos no ver cosas que refutan nuestra posición).

Dependiendo de lo que estén haciendo, es posible que los evaluadores necesiten reconocer su modo y hacer un esfuerzo (usando herramientas) para pasar a un nuevo modo.

Consulte la bibliografía de la Parte II para obtener referencias y aprender más sobre las ideas que menciona Sharon.

organizando

El tiempo siempre es una limitación, por lo que si vamos a realizar actividades de prueba esenciales, es vital tener buenas habilidades organizativas. Saber cómo planificar y administrar bien su tiempo, utilizando enfoques como las pruebas basadas en riesgos, puede ayudarlo a concentrarse en las tareas correctas. Con tantas exigencias de tiempo, incluidas reuniones, correo electrónico, mensajes instantáneos, planificación y seguimiento de actividades, puede resultar difícil realizar pruebas reales. Es demasiado fácil terminar goleando, repitiendo las mismas pruebas una y otra vez. Saber cómo organizar su tiempo también ayuda a garantizar que tenga tiempo para aprender cualquier otra habilidad que su proyecto pueda requerir.

La historia de Janet

En el Capítulo 3, "Roles y Competencias", hablamos sobre las competencias y fortalezas. Compartiré una historia personal sobre una de mis fortalezas y cómo lo incorporamos al proceso de escritura del libro. Tenemos fechas límite que acordamos con el editor. Utilizando mis habilidades organizativas, Creé el plan de lanzamiento y lo mantuve visible para que Lisa y yo pudiéramos hablar sobre riesgos y planificar en consecuencia. Creé la hoja de cálculo compartida para realizar un seguimiento de quién nos dio historias y cuándo se actualizaron. Estas sencillas herramientas ayudó a mantenernos encaminados; La organización es esencial en casi todo. hacemos.

Por supuesto, también tengo debilidades. Por ejemplo, escribir palabras no es mi fuerza, muchas veces cuando tengo una idea, la transmito de la mejor manera que puedo y luego haz que Lisa haga su magia.

Colaborando

Sea sensible a las desventajas del cambio de tareas mientras planifica cada día. Y si se siente abrumado, no tema pedir ayuda. Las pruebas ágiles deben ser un esfuerzo colaborativo.

Un proceso de colaboración eficaz

Hemos discutido una amplia gama de ~~de pensamiento~~ ~~de pensamientos~~ e interpersonales. ~~habilidades en este capítulo de estos~~ Sharon Robson ~~muchos~~ juntos muestran cómo realizar pruebas ~~poder~~ equipos de ayuda colaboran para más efectivas.

Uno de los objetivos clave de un equipo ágil (idealmente cualquier equipo) es colaborar en el trabajo que están realizando. La colaboración aumenta la calidad de los resultados y la aceptación del equipo que trabaja en conjunto. Sin embargo, ¡La colaboración es difícil! Para colaborar bien, los miembros del equipo deben entender por qué están colaborando y luego planificar cómo lo harán colaborar eficaz y eficientemente.

El Proceso de colaboración

Todos estos pasos deben realizarse en equipo para cada colaboración. sesión. Cada sesión debe centrarse en un objetivo e idealmente no exceder una hora. Algunas sesiones necesitarán más definición del proceso. que otros.

1. Defina el objetivo de la sesión: asegúrese de que haya un objetivo claro y resultado específico, por ejemplo, para definir las historias XYZ, para Elaborar la historia 57, para coordinar nuestro trabajo del día. Una vez el El objetivo se entiende y se acuerda claramente, documentarlo de forma sencilla y pasar al siguiente paso.
2. Definir el idioma a utilizar. Cada sesión utilizará específicos palabras que tienen significado contextual; dentro del contexto de la sesión el significado debe ser entendido claramente por todos los colaboradores. Cualquier palabra poco clara, ambigua o confusa debe tener definiciones o suposiciones hechas sobre ellos para permitir que el equipo para discutir el paso 3.
3. Defina el proceso para lograr el objetivo (p. ej., lluvia de ideas, elaboración de estatutos, discusión, diagramación) en función del objetivo de la sesión. ¿Qué actividades deben realizarse para lograr el objetivo? Suelen ser en forma de etapa de descubrimiento o investigación. (lluvia de ideas), luego una etapa de análisis (agrupación, discusión).

Estos conducen a una etapa de comprensión o evaluación (diagramación o mapeo), seguido de una etapa de conclusión o decisión.

4. Establecer las casillas de tiempo para las etapas del proceso. Una vez completadas las etapas se han planteado, asignar a cada uno de ellos tiempos que se ajusten a el cuadro de tiempo general. Nota: ¡Dé tiempo para volver a trabajar!
5. Siga el proceso, agregando o modificando el proceso, las suposiciones y el lenguaje a medida que avanza. ¡Sigue centrándose en el objetivo! Mantener preguntándose si la actividad actual conduce hacia la meta.
6. Evaluar el objetivo. ¿Se ha cumplido? En caso afirmativo, concluya la sesión; si No, cambia algo.
7. Repita los pasos del 2 al 6 según sea necesario hasta alcanzar el objetivo.

Las sesiones colaborativas que Sharon describe requieren una facilitación competente. Incluso si su equipo tiene un facilitador experimentado, comprender la dinámica de las reuniones y aprender habilidades de facilitación ayudará a todos los miembros del equipo a obtener más valor de las reuniones y sesiones de colaboración.

Resumen

Las habilidades de pensamiento juegan un papel importante en todos los aspectos de las pruebas de software. Algunos de los más importantes para practicar incluyen

- Facilitar
- Resolución de problemas
- Dar y recibir comentarios
- Aprender el ámbito empresarial
- Habilidades de entrenamiento y escucha
- Pensar de manera diferente y aplicar diferentes estilos de pensamiento a diferentes actividades de prueba.
- Organizar
- Colaborar, utilizando un proceso paso a paso

Esta página se dejó en blanco intencionalmente.

Capítulo 5

Conciencia técnica

5. Conciencia técnica

Guiar el desarrollo con ejemplos

Habilidades de automatización y codificación

Habilidades técnicas generales

Entornos de desarrollo

Entornos de prueba

Sistemas de integración continua y control de código fuente

Prueba de atributos de calidad

Técnicas de diseño de pruebas

Las habilidades de pensamiento ayudan a todo el equipo a trabajar bien en conjunto para garantizar que se planifiquen y ejecuten todas las actividades de prueba necesarias. Las habilidades técnicas de prueba ayudan a cerrar la brecha entre lo que la empresa necesita y cómo el equipo de entrega lo suministra. Hemos utilizado el término conciencia técnica para cubrir las ideas de habilidades técnicas necesarias para realizar pruebas y comunicarse con otros miembros del equipo de desarrollo. La primera vez que Janet escuchó el término conciencia técnica fue en un taller de pruebas local donde Lynn McKee lo utilizó durante una de las discusiones. Parecía captar la intención sin obsesionarse con los términos existentes. Creemos que las pruebas son en sí mismas una habilidad técnica especializada, por lo que hemos decidido dedicar este capítulo a muchos tipos de habilidades técnicas que serán útiles en su conjunto de herramientas de pruebas ágiles.

Guiar el desarrollo con ejemplos



Mantenemos conversaciones con los clientes sobre ejemplos de comportamiento deseado y no deseado para cada nueva característica e historia. Estos ejemplos se pueden convertir en pruebas empresariales automatizadas que guíen el desarrollo. Algunos enfoques bien conocidos para esto son la especificación por ejemplo (SBE), el desarrollo impulsado por pruebas de aceptación (ATDD) y el desarrollo impulsado por el comportamiento (BDD). Si aún no estás familiarizado con

Para estos, consulte la bibliografía de la Parte II, "Aprendizaje para mejores pruebas", para conocer algunos buenos lugares para comenzar a aprender. ATDD by Ejemplo de Markus Gärtner (Gärtner, 2012), Especificación por ejemplo de Gojko Adzic (Adzic, 2011) y The Cucumber Book de Matt Wynne y Aslak Hellesøy (Wynne y Hellesøy, 2012) son buenas introducciones. Capturar ejemplos apropiados y convertirlos en pruebas automatizadas requiere cierta conciencia técnica para poder colaborar con sus programadores. Hablaremos con más detalle sobre esta práctica en la Parte IV, "Prueba del valor empresarial".

Habilidades de automatización y codificación

Cuando los evaluadores colaboran con programadores, administradores de sistemas, expertos en bases de datos y personas con otras funciones técnicas, pueden ayudarse mutuamente a diseñar pruebas efectivas en todos los niveles y automatizar tareas diarias, como la implementación de código en entornos de prueba. Esta colaboración es más fácil cuando los evaluadores tienen algunos conocimientos técnicos y todos los miembros del equipo de desarrollo tienen algunas habilidades de prueba.

Si los evaluadores aprenden a utilizar el mismo entorno de desarrollo integrado (IDE) que los codificadores, el emparejamiento para ver el código se vuelve más fácil y el lenguaje utilizado para discutir la aplicación se convierte en un lenguaje compartido.

Las pruebas se pueden especificar en un estilo natural, utilizando un lenguaje de dominio específico (DSL) o utilizando palabras clave o datos para guiar las pruebas. Sin embargo, existe un código subyacente que ejecuta las pruebas y produce los resultados. Alguien tiene que escribir eso, y el código de automatización de pruebas es, bueno, código. Si usted, como evaluador, no sabe codificar, es importante poder comprender qué hace el código.

Cuando un equipo guía el desarrollo con ejemplos comerciales y practica el desarrollo basado en pruebas (TDD), las pruebas ayudan a crear un buen código. Proporcionan documentación viva de cómo se comporta el código de producción y garantizan que siga comportándose de esa manera. Si es necesario cambiar el código más adelante, las pruebas hacen que hacerlo sea más seguro y rápido. Por lo tanto, debemos tratar el código de prueba con el mismo cuidado y respeto que el código de producción: son igualmente valiosos.

La aplicación de buenas prácticas de diseño ayuda a mantener la automatización rentable. Por ejemplo, nos esforzamos por mantener cada prueba automatizada centrada en un solo

propósito claro y extraer duplicaciones en macros y módulos. Nuestro objetivo es facilitar el diagnóstico de fallas en las pruebas y cambiar las pruebas en un solo lugar cuando cambia el código. Puede encontrar más detalles al respecto en el Capítulo 16, "Patrones y enfoques de diseño de automatización de pruebas".



Aprender a escribir pseudocódigo puede ayudarle a diseñar pruebas automatizadas. Obtenga una comprensión básica de los principios orientados a objetos como SOLID (responsabilidad única, abierto/cerrado, sustitución de Liskov, segregación de interfaz, inversión de dependencia) (consulte Wikipedia, 2014m).

Si sabe leer el código y comprende los estándares de codificación de su equipo, es posible que pueda asociarse con programadores para escribir pruebas, revisar el código y tal vez ayudar con los problemas de depuración. Como mínimo, podrá tener conversaciones significativas con los programadores de su equipo para saber qué áreas del código son más frágiles y dónde centrar los esfuerzos de prueba. Esta también es una oportunidad para mejorar sus habilidades de codificación y secuencias de comandos. Como beneficio adicional, es posible que descubras defectos mientras realizas el emparejamiento.

Si no tienes experiencia en programación, practica algunos conceptos básicos por tu cuenta. Consulte la bibliografía de la Parte II para obtener libros útiles sobre cómo aprender a codificar, como *Everyday Scripting with Ruby* de Brian Marick (Mar-ick, 2007). También hay cursos en línea, tutoriales y screencasts.

Las habilidades de codificación también son útiles para ayudar a configurar datos y escenarios para cartas de pruebas exploratorias. Incluso si no tiene habilidades de programación, saber qué podría automatizarse lo ayudará a colaborar con los programadores de su equipo para realizar una automatización que ahorre tiempo y liberar a los evaluadores para actividades en las que aporten el mayor valor.

Conozca la arquitectura de su producto de software, al menos a un alto nivel. Pida a sus compañeros de equipo que identifiquen las áreas de riesgo. El equipo de Lisa representa áreas más frágiles de la arquitectura con líneas de puntos en diagramas. Esta visibilidad le ayuda a centrarse en diferentes tipos de pruebas de forma eficaz y a considerar estrategias de automatización de pruebas junto con su equipo. El conocimiento del diseño de sistemas puede ayudarle a realizar pruebas de manera eficiente. Por ejemplo, si sabe que una función de búsqueda utilizada en toda la aplicación está encapsulada en una parte del código, es posible que pueda probarla exhaustivamente a través de una API y solo necesite comprobaciones superficiales en otras partes.

Comprender las interfaces del sistema le permite reconocer sus posibles debilidades. Hay más que la interfaz de usuario (UI); esté atento a otros, como el registro y la supervisión de operaciones, mensajería o protocolos de comunicación.

La Figura 5-1 muestra un ejemplo del tipo de diagrama de arquitectura que proporciona un buen punto de partida para comprender los componentes de un sistema y cómo se relacionan entre sí. Este muestra la arquitectura de una API que desarrolló el equipo de Lisa. Ayudó al equipo a visualizar cómo el software generaría documentación de usuario y solicitaría datos de validación, así como a ejecutar comandos para agregar, cambiar o eliminar datos. recursos.

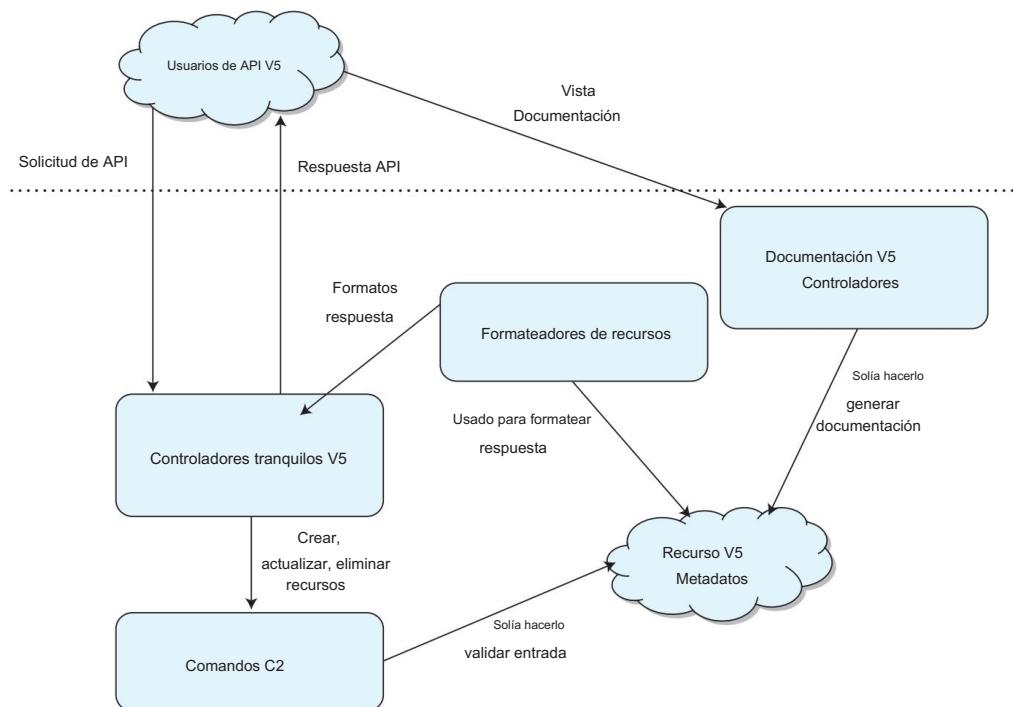


Figura 5-1 Diagrama de arquitectura de ejemplo para una API

Habilidades técnicas generales

Cada equipo y organización con la que trabajamos tiene diferentes necesidades y diferentes contextos. Las habilidades técnicas necesarias para realizar pruebas de forma eficaz varían de un equipo a otro. Sin embargo, existen algunas habilidades técnicas básicas que le serán de gran utilidad en cualquier entorno.

Saber cómo monitorear procesos, memoria y CPU, leer archivos de registro e interpretar estadísticas de perfiles puede ayudarlo a identificar posibles fugas o uso indebido de recursos. También observe y comprenda características de rendimiento importantes que, de otro modo, podrían pasar desapercibidas hasta que su producto entre en producción. El uso de la supervisión y el registro es otra área en la que un evaluador y un programador o un practicante de DevOps pueden trabajar juntos para profundizar realmente en problemas difíciles de alcanzar. La capacidad de ver o seguir un archivo de registro durante la prueba es una necesidad con la mayoría de los productos de software. Sentirse cómodo en una línea de comandos con comandos básicos de shell de UNIX es útil para una amplia gama de actividades, como mantener entornos de prueba, monitorear archivos de registro y probar API.

La mayoría de las pruebas requieren al menos algún conocimiento de bases de datos. La verificación de datos es solo un aspecto de las pruebas. También necesitamos verificar la integridad y las restricciones relacionales. Las habilidades básicas de SQL u otros lenguajes de consulta son imprescindibles para probar cualquier aplicación que utilice una base de datos relacional para datos persistentes. Emparejar a un experto en bases de datos con un evaluador es una excelente manera de verificar los atributos de calidad de la base de datos, mientras se transfieren habilidades. Abordaremos más las pruebas de datos y bases de datos en el Capítulo 22, “Pruebas ágiles para almacenes de datos y sistemas de inteligencia empresarial”.

Entornos de desarrollo

La capacidad de actualizar, crear e implementar el código más reciente de su equipo en su máquina local proporciona más flexibilidad para probar y depurar problemas.

Si está ayudando a automatizar pruebas, probablemente querrá consultar el código más reciente, escribir las pruebas, ejecutarlas localmente y registrar las nuevas pruebas.

Por supuesto, esto depende de su contexto. Por ejemplo, Adam Knight nos dijo que su equipo extrae la compilación de los artefactos del sistema de integración continua (CI) y ejecuta múltiples ejecuciones de prueba en paralelo para que los evaluadores tengan un esfuerzo de automatización masivo sin involucrarse en la compilación del código.

Otro beneficio de poder ver el código más reciente es que a menudo podemos ahorrar tiempo haciendo algunas de las “correcciones” nosotros mismos. Cuando Lisa ve un error ortográfico en una página de ayuda, puede corregirlo ella misma, ejecutar las pruebas automatizadas y verificar la corrección. Cualquier control y equilibrio que exista para sus programadores también se aplica a los evaluadores. No importa cuál sea su función, si está tocando código, debe seguir los estándares y prácticas de codificación y asegurarse de que se realicen las pruebas adecuadas.

Para realizar este tipo de trabajo, los evaluadores deben ponerse al día con la herramienta de control de código fuente que utiliza su equipo e, idealmente, con el IDE. Esto es algo que se hace fácilmente con el emparejamiento probador/programador. Cuando Lisa se une a un compañero de equipo para trabajar en el código, toma notas de las técnicas que podría necesitar más adelante y las mantiene en la wiki del equipo para consultarlas en el futuro y compartirlas con otros evaluadores.

Entornos de prueba

Muchos equipos ágiles luchan por crear y mantener entornos de prueba útiles. Algunos equipos configuran sus compilaciones de CI para implementar automáticamente artefactos de compilación en entornos de prueba cuando se aprueban los conjuntos de pruebas. En otros equipos, los evaluadores implementan los artefactos que desean probar cuando estén listos. Lamentablemente, algunos equipos aún no cuentan con un proceso de CI y algunos ni siquiera cuentan con entornos de prueba adecuados. Sin un proceso de construcción confiable, es difícil brindar retroalimentación rápida al equipo sobre las historias entregadas. Sin entornos de prueba adecuados, es difícil proporcionar información correcta sobre el estado del código.

Una práctica estándar que hemos visto funcionar en muchos equipos es tener varios entornos de prueba para diferentes propósitos. Los programadores generalmente utilizan una zona de pruebas local que ellos mismos mantienen con fines de prueba. Cada evaluador también puede tener su propio entorno de prueba local. Lo ideal es que un equipo tenga al menos un entorno de prueba que imite el sistema de producción para realizar pruebas exploratorias realistas. Los equipos también necesitan un entorno de prueba que sea una copia o al menos una buena representación de la producción donde puedan probar cada versión, incluida cualquier migración de datos. Una práctica común es copiar una instantánea de los datos de producción, con todos los datos confidenciales eliminados, en el entorno de prueba, de modo que esté probando con datos realistas.

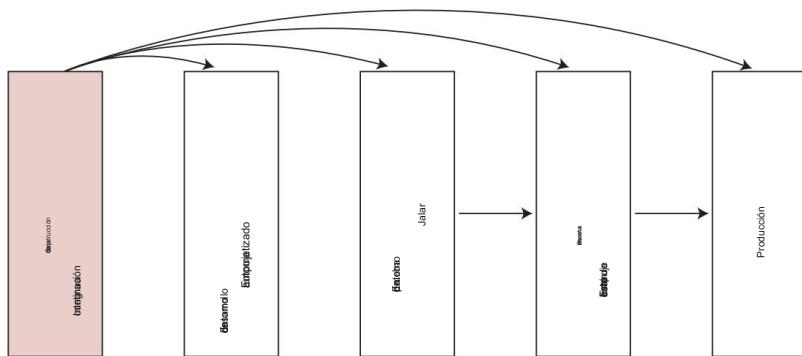


Figura 5-2 Canalización de compilación simple

Puede haber muchos más entornos de prueba, como uno especializado para medir la carga y el rendimiento, o uno para ejecutar conjuntos de pruebas automatizadas. La Figura 5-2 representa la forma más simple de una canalización de compilación, pero puede ser compleja. En el Capítulo 23, "Pruebas y DevOps", un colaborador comparte cómo gestionó sus entornos de prueba.

Los equipos que tienen múltiples bases de código o ramifican su código desde la versión de control del código fuente maestro (troncal de desarrollo principal) generalmente necesitan entornos de prueba adicionales. El equipo de Lisa tiene servidores de prueba que se utilizan para probar picos de diseño arquitectónico o de código, o proyectos especiales que aún no forman parte del código de producción. También son útiles al actualizar a una nueva versión del lenguaje de programación o marco de desarrollo.

Los evaluadores deben aprender sobre los diferentes entornos de prueba y saber cuándo solicitar diferentes configuraciones o servidores adicionales. Según nuestra experiencia, las pruebas son más efectivas cuando los evaluadores saben qué código y datos deben estar en un entorno de prueba determinado y tienen las habilidades para garantizar que estén allí. Janet trabajó con un equipo que enumeró todos los entornos de prueba en una pizarra grande. Mostraba exactamente qué versión de la compilación se encontraba en qué entorno y cuándo se había implementado. Una rápida verificación visual aseguró que el equipo supiera lo que estaban probando.

El equipo documenta los distintos entornos de prueba y preparación, incluido el propósito de cada uno, en una hoja de cálculo en la wiki del equipo.

Integración continua y código fuente

Sistemas de control

A medida que los equipos crecen, también lo hacen la complejidad y la cantidad de compilaciones de CI y conjuntos de pruebas. Los equipos comienzan a encontrar el dilema donde pasan las pruebas unitarias, pero fallan las pruebas para un navegador en particular o un conjunto de pruebas de integración. Necesitamos preguntarnos: "¿Vale la pena probar este código en cualquier entorno?" Es importante tener un entendimiento común de lo que significa para el equipo lograr que todas las pruebas pasen (llegar a verde) nuevamente.

Tómese el tiempo para comprender las configuraciones de CI de su equipo. Por ejemplo, si su proceso de CI se implementa automáticamente en un entorno de prueba, asegúrese de saber qué conjuntos de pruebas deben aprobarse antes de que se pueda implementar la compilación.

Los evaluadores deben aprender cuáles son los diferentes trabajos de CI y cómo se relaciona el éxito o el fracaso de cada uno con las pruebas. Si está probando la lógica empresarial en el lado del servidor, puede que no importe que un conjunto de pruebas de navegador en particular haya fallado. Pero si está probando la lógica del lado del cliente, es posible que prefiera esperar a que pasen todas las pruebas de regresión en todos los navegadores.

Comprenda cuál es el riesgo para su sistema. Sepa con qué programadores hablar si un conjunto de pruebas en particular falla. A veces, las pruebas que fallan no afectan la característica particular que desea probar, por lo que es posible implementar manualmente el código y continuar con las pruebas.

Tener varios equipos de desarrollo complica el proceso de CI. Lo que hemos visto con más frecuencia es que los equipos individuales tienen su propio proceso de CI para su código, y la mayoría de las pruebas se pueden realizar en el propio entorno de prueba del equipo. Su código también se prueba como parte de un proceso de CI en toda la empresa. Las fallas de regresión en el código de otros equipos podrían afectar su capacidad para probar el sistema más grande. Si aún no existe un proceso para identificar la fuente de fallas en un CI de toda la empresa y algunos medios para lograr que el equipo responsable solucione el problema, intente reunir a las personas adecuadas para implementar uno y determinar cómo resolverlo. la cuestión.

La ramificación complica los problemas de CI. Si su equipo siempre permanece en el maestro (a veces llamado principal o troncal), la vida es simple. Pero digamos que tu

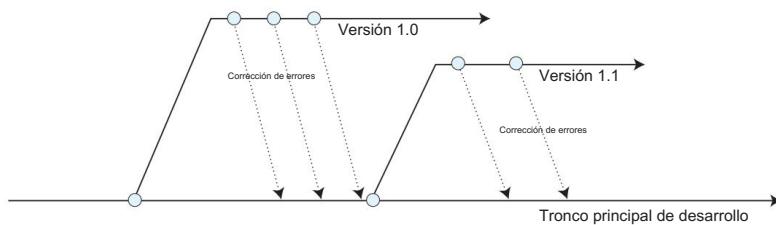


Figura 5-3 Una posible estrategia de ramificación

El equipo está haciendo una gran refactorización en master, mientras que es necesario corregir un error en la rama de producción. Debe asegurarse de comprobar lo correcto y/o implementar la compilación correcta en el entorno de prueba adecuado para su verificación. Esta es otra área en la que es esencial colaborar con programadores y posiblemente con DevOps. En el diagrama que se muestra en la Figura 5-3, las correcciones de errores se fusionan en el tronco de desarrollo principal o se realizan dos perforaciones (ambas se actualizan con la misma solución).

Una estrategia de ramificación simple

Augusto Evangelista, un pruebas ágiles entusiasta de Irlanda, comparte su información sobre cómo su organización maneja las sucursales de la forma más sencilla posible.

Hemos resuelto el problema relacionado con la bifurcación y la fusión de manera bastante drástica al eliminar la bifurcación por completo y realizar operaciones puramente continuas. integración, con todos codificando directamente en el troncal. Tenemos suerte de que sólo tenemos dos o tres equipos empujando a la vez, además de un Sistema CI muy avanzado y rápido que lo hace muy fluido. Tenemos Función de alternancia introducida recientemente porque estamos haciendo cambios continuos. entrega y a veces nuestros clientes no están listos para usar un nuevo usuario historia. Lo probamos en nuestros entornos internos, pero lo alternamos en producción a través de la configuración de Spring.

Usamos Git, una herramienta de administración de código fuente distribuido, y cada La casilla del desarrollador es una "sucursal" como tal hasta que él o ella se registre en el código.



Idealmente, todos podríamos trabajar en el maestro de nuestro código fuente, y muchas organizaciones pueden tener éxito con esa estrategia. Sin embargo, a medida que las organizaciones crecen o quizás tienen sistemas más complicados, otras estrategias evolucionan.

Una estrategia de ramificación más compleja

Adán Caballero, director de control de calidad y del Reino Unido, comparte las experiencias de ramificación de su equipo, que son completamente diferentes a las de Augusto y mucho más complejas.

son

Porque somos una empresa de productos de bases de datos con un modelo de entrega tradicional de versiones de software compatibles, ramificaciones y fusiones. son hechos desafortunados de la vida. A medida que nuestros clientes implementan nuestro producto en el sitio como parte de instalaciones importantes, necesitamos mantener sucursales de versiones anteriores que todavía están en producción en los sitios de los clientes alrededor el mundo.

Usamos Subversion y guardamos las pruebas junto con el código. En el En el punto de cada lanzamiento, ramificamos las pruebas junto con el código en un rama de lanzamiento para esa versión. En caso de que se requiera alguna solución importante, aplíquelos al código troncal y pruebe. Luego presionamos tanto la solución como las nuevas pruebas regresan a cualquier rama de versión relevante y actualizan el problema libera según sea necesario.

Uno de los principales desafíos que enfrentamos es tratar de mantener estos lanzamientos. sucursales "lineales", ya que a menudo diferentes clientes requieren diferentes soluciones y no necesariamente quiero arriesgarme a tener otras correcciones en su actualización lanzamientos. Hasta ahora he logrado ganar este juego, ya que creo que habiendo parches individuales separados exponen una gran cantidad de riesgo dada la número exponencial de combinaciones no probadas en las que esos parches podría aplicarse. He tenido que explicar a algunos clientes que tienen cuestionó este enfoque la gran diferencia de riesgo entre un parche y una versión completa del software que ha pasado por todas nuestras pruebas automatizadas antes de su lanzamiento.

Otra situación en la que se pueden crear sucursales es si estamos involucrados en una prueba de concepto (POC) donde se cumplen los requisitos del POC exigir trabajo adicional por hacer en la base del código para lograr a ellos. En esta situación bifurcamos el código y agregamos un código de alta prioridad. historia de creación de prototipos para la iteración para intentar alcanzar los objetivos especificados (generalmente en torno al rendimiento de consultas o la simultaneidad para un dato específico colocar). Una vez que estos elementos están completos, creamos historias para posteriores. iteraciones para cada característica que queremos incorporar al tronco principal.

Esas historias implican reelaborar las funciones para que sean de calidad e integrarlas completamente con el resto del producto mediante las pruebas adecuadas. El principal riesgo de este enfoque es que existe la percepción de un mayor nivel de finalización de esos elementos prototípico del que realmente existe y, en consecuencia, una expectativa poco realista sobre la rapidez con la que se pueden convertir en código liberable.

Como dije, la necesidad de mantener múltiples sucursales no es una situación deseable sino una realidad dado nuestro contexto y modelo de entrega.

Como señala Adam, los evaluadores y sus equipos de desarrollo también deben gestionar las pruebas y el código de prueba dentro del contexto del sistema de control del código fuente del equipo. Las pruebas automatizadas deben versionarse con el código que prueban. Si aún no está familiarizado con el sistema de control de código fuente de su equipo, asóciense con un programador, administrador del sistema o una persona de DevOps para aprender cómo usarlo y cómo está organizado el código de su equipo.

Prueba de atributos de calidad

Agile Testing explicó cómo utilizar los cuadrantes de pruebas ágiles (Figura 5-4) para ayudar a garantizar que el equipo planifique y ejecute los tipos de

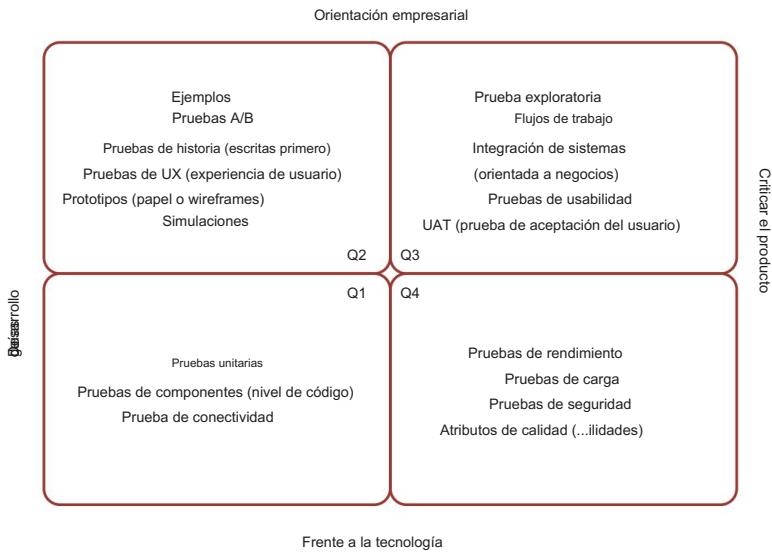


Figura 5-4 Cuadrantes de pruebas ágiles

pruebas que necesitan. El Capítulo 8, "Uso de modelos para ayudar a planificar", detalla la planificación para los cuatro cuadrantes, pero en este capítulo cubriremos algunas de las habilidades técnicas que quizás desee mejorar.



Las pruebas del Cuadrante 4 (pruebas tecnológicas que critican el producto) a veces se ignoran o subestiman, ya que a menudo es donde las habilidades de prueba son más débiles. Este cuadrante contiene importantes atributos o limitaciones de calidad que frecuentemente no se especifican en las historias. Un ejemplo de restricción no declarada es que "el rendimiento de todas las páginas web debe responder en menos de tres segundos". Si su equipo utiliza herramientas analíticas para medir el desempeño, querrá aprender cómo ejecutarlas e interpretar los resultados.

Es posible que las organizaciones carezcan de personas con las habilidades necesarias para algunas áreas de especialidad. Por ejemplo, tal vez su equipo no tenga especialistas en pruebas de seguridad. Su equipo puede aprender algunas técnicas básicas de pruebas de seguridad para ayudar a brindarle a su empresa cierto grado de seguridad. Aprender sobre inyección SQL y secuencias de comandos entre sitios cubrirá algunas de las áreas básicas de vulnerabilidad. Janet tuvo una experiencia reciente con un sitio web que quería realizar un pago y solicitaba los datos completos de la tarjeta de crédito. Se dio cuenta de que no era un sitio HTTPS. También lo hicieron algunas pruebas. El sitio ni siquiera ocultó los detalles de la página, y un simple "ver fuente" después del envío mostró todos los detalles que había ingresado en texto sin formato. Sin embargo, hay mucho más para garantizar la seguridad de un producto de software. Educar a su empresa sobre el valor de una auditoría de seguridad puede ser la contribución más valiosa que pueda hacer.

Descubra qué es importante para su organización y adquiera conciencia técnica en esa área. Quizás sea suficiente saber que puedes recomendarles que contraten a un experto.



Colocamos las pruebas exploratorias en el Cuadrante 3 (pruebas orientadas al negocio que critican el producto) y hemos dedicado el Capítulo 12, "Pruebas exploratorias", al tema. Sin embargo, queríamos mencionarlo aquí porque es un conjunto de habilidades que debes continuar practicando y creciendo. Enseñar habilidades de pruebas exploratorias a los programadores de su equipo puede ayudarlos a ser mejores codificadores. En su libro ¡Exploralo! (Hendrickson, 2013), Elisabeth Hendrickson ofrece ejemplos de cómo los programadores también pueden utilizar pruebas exploratorias a nivel de código. A veces, pensando en la onda expansiva

Los efectos de pequeños fragmentos de código les ayudarán a evitar dañar otras partes de la aplicación con código nuevo o cambios en el código existente. La vinculación con los evaluadores ayuda a los programadores a adquirir conciencia sobre las pruebas.

Técnicas de diseño de pruebas

Al decidir qué probar y cómo, es importante tener una caja de herramientas llena de técnicas, como el uso de diagramas de transición de estados o tablas de decisión. Hemos incluido libros y cursos recomendados sobre diseño de pruebas y pruebas de dominio en la bibliografía de la Parte II.

Resumen

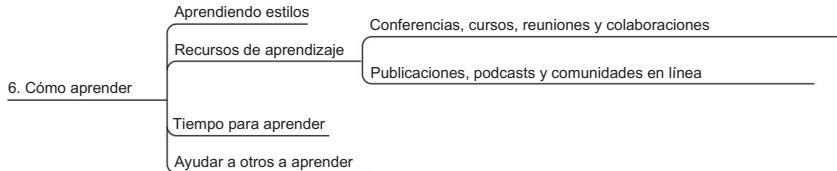
Los miembros del equipo "en forma de T" necesitan conocimientos técnicos en una variedad de áreas para planificar y ejecutar actividades de prueba de manera efectiva. Existe una amplia variedad de conocimientos básicos que ayudarán a los miembros del equipo en todos los roles a colaborar de manera más efectiva para las pruebas. Los evaluadores aportan al grupo habilidades de prueba profundas, pero necesitan comunicarse de manera efectiva con programadores, administradores de sistemas y otras funciones técnicas. Al mismo tiempo, los miembros del equipo que desempeñan otros roles y que participan en actividades de prueba también deben practicar estas habilidades. Éstas incluyen

- Guiar el desarrollo con ejemplos
- Conocimiento técnico de arquitectura y diseño de código.
- Habilidades de automatización y codificación
- Mantenimiento de entornos de prueba
- Comprender el control del código fuente y la integración continua
- Probar atributos de calidad, guiados por los cuadrantes de pruebas ágiles
- Técnicas de diseño de pruebas

Esta página se dejó en blanco intencionalmente.

Capítulo 6

Cómo aprender



En el Capítulo 3, “Roles y competencias”, explicamos la importancia de convertirse en un especialista generalizador que sea capaz de colaborar con miembros del equipo que tienen otras especialidades y al mismo tiempo contribuir con su propia experiencia profunda. En el Capítulo 4, “Habilidades de pensamiento para las pruebas”, y en el Capítulo 5, “Conciencia técnica”, exploramos una variedad de habilidades que ayudan a los equipos de software a hacer un mejor trabajo al probar y entregar un producto de alta calidad y valor. Desarrollar todas estas habilidades diferentes amplía nuestra perspectiva y nos permite abordar nuestro trabajo de manera más creativa. En este capítulo, veremos formas de aprender estas diversas habilidades. Hay muchos recursos educativos disponibles, pero primero, piense en su estilo de aprendizaje y su cultura de equipo, los cuales afectan su éxito en la adquisición de nuevas habilidades.

Aprendiendo estilos

Cada uno de nosotros tiene formas preferidas de aprender. Algunos de nosotros aprendemos mejor escuchando: somos aprendices auditivos. A algunos de nosotros nos gusta ver imágenes; tenemos una preferencia visual. Muchos de nosotros aprendemos haciendo; algunas personas lo llaman aprendizaje cinestésico. Muchas veces necesitamos absorber información de más de una forma. Por ejemplo, Janet aprende auditivamente, pero necesita practicar habilidades para internalizarlas. Sin embargo, cuando intenta transmitir un concepto a los demás, necesita dibujarlo para ayudar a exponerlo. Diferentes vías de aprendizaje se aplican a todos, pero en diferentes grados.

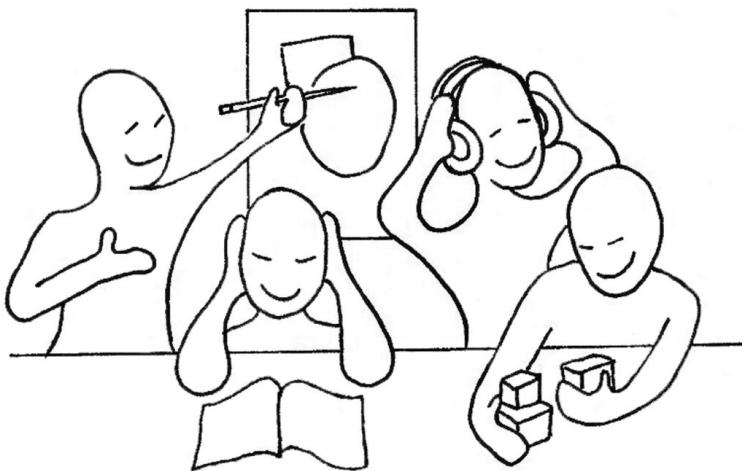


Figura 6-1 ¿Cómo se aprende?

¿Cuál es tu estilo preferido? Comprenda el suyo para que pueda aprovechar al máximo cada experiencia de aprendizaje.

Las emociones afectan la forma en que asimilamos la información. Todos tenemos puntos ciegos que pueden impedirnos aprender, factores desencadenantes que nos hacen cerrarnos y no escuchar más el mensaje. Para aprender y preguntar, necesitamos un entorno seguro. Si está ayudando a otros miembros del equipo a aprender una nueva habilidad, tenga en cuenta que las personas pueden tener reacciones emocionales a lo que usted dice o a cómo lo dice. Es posible que no comprendan el mensaje si no les gusta el formato de entrega. La figura 6-1 muestra los muchos estilos que tiene la gente: lo que funciona para una persona puede no funcionar para otra.

Mientras aprende, ya sea una clase de capacitación, un tutorial en línea o una sesión individual, reflexione sobre los “puntos calientes” emocionales que tiene. Cuando sienta que está bloqueando información debido a la forma en que se presenta, intente concentrarse en el valor que puede obtener del instructor o del material. Tenga esto en cuenta cuando colabore con compañeros de equipo de desarrolladores y expertos en negocios, y obtenga toda la información que mejorará las pruebas. Busque sus propios puntos ciegos que puedan impedirle aprender y observe para reconocerlos en los demás (Gregory, 2010).

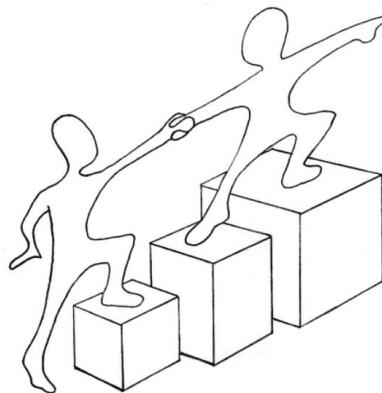


Figura 6-2 Ayudantes

David Hussman y Jean Tabaka facilitaron un taller llamado “Zen, la mente del principiante”, diseñado para abrir las mentes a nuevas ideas (Levison, 2008), en Agile 2008. Janet participó y aprendió que cuando abordamos problemas con mentes cerradas y comenzamos cada frase con “Pero eso no funcionará porque...” . . .” o “Eso no funcionó cuando lo probamos en . . .” Cerramos nuestra mente a las oportunidades. Tal vez fue sólo un pequeño problema solucionable lo que hizo que fallara antes. Eso no significa que la idea fundamental fuera errónea. O tal vez no estabas preparado para oírlo. A veces no tenemos la información básica que nos permitirá captar un concepto. Abre tu mente a la maravilla del aprendizaje; Leer libros de filosofía u otros estudios puede mostrarle diferentes formas de abordar un problema.

Mire más allá de la profesión de desarrollo y pruebas de software en busca de oportunidades de aprendizaje. Los instructores, entrenadores o mentores en otros campos pueden adaptarse a su estilo de aprendizaje y brindarle una perspectiva diferente. El ayudante adecuado puede mostrarnos formas de alcanzar nuevas ideas (ver Figura 6-2).

La historia de Janet

Cuando Lisa y yo comenzamos a desarrollar e impartir nuestro curso de pruebas ágiles, Empecé a leer sobre estrategias de enseñanza. Recogí algunos consejos de libros y los apliqué. También observé a otros instructores y traté de encontrar el estilo que me convenía. Sin embargo, una de las personas a las que recurrió más para ser mentora fue mi hermana, Carol Vaage. Ella era maestra de primer grado con

su maestría en aprendizaje en la primera infancia y me abrió los ojos a nuevas formas de abordar la enseñanza y el trabajo con mis clases.

Por ejemplo, introdujo formas de utilizar preguntas abiertas para conseguir sus clases para empezar a pensar en un tema. He incluido algunos aquí, pero el La lista es mucho más larga y se incluye en el Apéndice B, "Iniciadores de provocación".

- Averigüemos cómo podría ser eso. . .
- ¿Qué pasaría después?
- ¿Por qué sería eso?
- ¿Por qué cree que pasó?
- ¿Te diste cuenta?

Si pensamos en cómo aprenden los niños y nos damos permiso para explorar nuestra curiosidad natural sobre el mundo, pensar hasta dónde podríamos llegar.

Carol también tiene algunas historias sorprendentes e imágenes de hasta dónde pueden llegar los niños para aprender, que hemos incluido en la bibliografía de la Parte II, "Aprender para mejorar las pruebas".

Puedes aprender de personas que admirás incluso sin una relación formal de mentoría. La mayoría de las personas estarán felices de ayudar, así que tenga el coraje de hacer preguntas.

Recursos de aprendizaje

Busque lugares para perfeccionar sus habilidades. Puede encontrar buenos recursos en línea, en su comunidad local o más lejos. Veamos algunos ejemplos de buenas oportunidades de aprendizaje. Consulte la sección de bibliografía de la Parte II "Cursos, conferencias, comunidades en línea, podcasts" para obtener enlaces a las actividades mencionadas en esta sección.

Conferencias, cursos, reuniones y colaboraciones

Una buena conferencia puede brindar una variedad de conclusiones, que van desde técnicas específicas que puede probar de inmediato hasta nuevas ideas o tecnologías innovadoras que puede continuar investigando. Lo más importante es que conocerá a profesionales y líderes de opinión con quienes podrá formar una red duradera, una fuente constante de inspiración e ideas.

Considere diferentes tipos de conferencias. Las conferencias de prueba son una opción obvia para los evaluadores, pero considere otras, como aquellas que ayudan

Trabajas en habilidades específicas, como lenguajes de programación. Si su empleador no puede permitirse el lujo de enviarlo a una conferencia y usted no puede pagar sus propios gastos, considere proponer un artículo o una sesión para una conferencia sobre pruebas o desarrollo de software. Esto puede otorgarle una inscripción a la conferencia gratuita o con descuento. Y recuerde, enseñar una habilidad a otra persona es la mejor manera de aprenderla usted mismo. Descubra si las conferencias necesitan voluntarios o si califica para subvenciones o descuentos. Si no puede viajar para asistir a una conferencia, considere las conferencias virtuales, que se están volviendo más populares, o regístrate para seminarios web.

Las conferencias no son solo para aprender habilidades técnicas y de pruebas específicas. Muchas conferencias de software tienen sesiones y temas sobre colaboración, cultura organizacional, aprendizaje, coaching, trabajo con clientes y tutoría. Janet incluso ha asistido a sesiones de Portia Tung sobre el "Power of Play". La Figura 6-3 muestra personas aprendiendo y jugando juntas.

El aprendizaje más importante en las conferencias suele tener lugar en los descansos y en los pasillos y comedores, a medida que conoces gente nueva que pasa a formar parte de tu red.

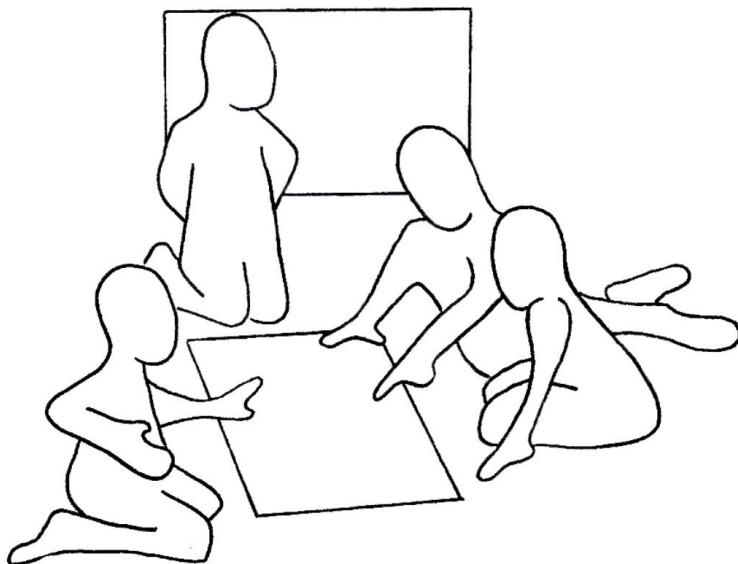


Figura 6-3 Aprendizaje y juego colaborativos

La mayoría de las áreas metropolitanas tienen grupos y reuniones de usuarios de prueba y desarrollo, que se reúnen periódicamente. Estos pueden ser un gran recurso de capacitación e información gratuita, así como un lugar para conocer personas que han probado diferentes herramientas y técnicas. Aproveche al máximo estas oportunidades de networking.

Si desea mejorar sus habilidades de enseñanza y entrenamiento, considere un retiro de entrenamiento como Agile Coach Camp. Las reuniones de juegos ágiles como Play4Agile son un excelente lugar para obtener ideas para el aprendizaje participativo y generar confianza en los equipos. Portia Tung (Tung, 2011) dice que el juego rompe barreras entre las personas y abre la mente para permitir el aprendizaje.

Existen talleres y cursos diseñados para ayudarle a aprender habilidades de liderazgo y relaciones, como el Liderazgo en resolución de problemas (Derby et al., 2014) y los que ofrece Satir Global Network (Satir Global Network).

Para mejorar en casi cualquier cosa, necesitamos practicar. Músicos, atletas y jugadores de videojuegos dedican tiempo a practicar. En los retiros de código (Haines et al.), los programadores practican repetidamente la escritura de código que desecharán. Lisa considera que los retiros de programación son una excelente experiencia de aprendizaje incluso si no eres un gran programador. Aprendió lecciones sobre resolución de problemas y práctica de su oficio. También hay buenas formas de practicar las habilidades para realizar pruebas; por ejemplo, asiste a un dojo de pruebas u organiza el tuyo propio.

Grupos como Weekend Testing le brindan la oportunidad de practicar técnicas de prueba en tiempo real con evaluadores de su continente o de todo el mundo. Si tienes una hora libre y quieres practicar una habilidad específica, consigue a alguien que te acompañe. Lisa recientemente contactó a sus contactos de Twitter y varios se ofrecieron como voluntarios para unirse virtualmente a ella para practicar Ruby Koans. Es posible que tus propios compañeros de equipo estén listos y dispuestos a ayudarte a practicar.

Es posible que pueda encontrar cursos ofrecidos en temas que le interesen. Muchos de ellos tratan sobre un tema específico, como la automatización de pruebas de la interfaz de usuario (UI). Algunas organizaciones contratan instructores para que enseñen a todo el equipo a la vez, de modo que todos tengan una comprensión común del tema.

Los cursos en línea, screencasts, seminarios web y tutoriales ofrecen oportunidades de aprendizaje convenientes para todos los aspectos de las pruebas y el desarrollo ágil.

así como las habilidades de pensamiento no técnico. Algunas capacitaciones en línea son gratuitas o económicas, aunque los cursos más sofisticados, o aquellos que incluyen tiempo en línea con el instructor, pueden costar más. Por lo general, puedes resolverlos a tu propio ritmo.

La historia de Janet

Recientemente hice un curso sobre kanban personal a través de www.udemy.com. I Podría tomarme mi tiempo y seguir el curso en cualquier lugar al que viajara. y a mi propio ritmo; Ambos aspectos fueron realmente importantes para mí. usé lo que Aprendí a ayudarme a concentrarme en las tareas que eran de máxima prioridad y no a obtener desviado hacia tareas que eran menos importantes en el momento actual.

Si desea adquirir una habilidad específica, busque videos en línea de sesiones de conferencias, reuniones de grupos de usuarios y cursos de capacitación.

Publicaciones, podcasts y comunidades en línea

La información sobre todos los aspectos de las pruebas de software está disponible tanto en forma de libro como en Internet. Tenemos la suerte de tener muchos buenos libros disponibles sobre todos los aspectos de las pruebas de software. Encontrará referencias a libros, artículos y publicaciones de blogs a lo largo de este libro, y la bibliografía de la Parte II tiene una lista completa.

Los podcasts son una forma conveniente de aprender. Puede escuchar entrevistas con expertos, sesiones de capacitación sobre temas específicos, conferencias magistrales, seminarios web y paneles de discusión.

Los podcasts cambiaron mi vida

Steve Rogalsky , un agilista comprometido con la historia del aprendizaje por podcasts.

A mi esposa le gusta contar la historia de cuando llegó a casa del trabajo y anunció: "Los podcasts han cambiado mi vida". Su respuesta natural ¿Era que? ¿Pódcasts? ¿No yo, ni nuestros hijos, sino los podcasts? I persistí con mi afirmación original. "Sí, podcasts. han cambiado mi vida."

Hasta ese momento había dependido en gran medida de las organizaciones que Trabajé para hacer el arduo trabajo de identificar lo que necesitaba aprender. siguiente y cómo. Cuando un amigo comentó hace varios años que escuchaba podcasts de camino al trabajo, decidí intentarlo. Después Sólo unos pocos episodios me engancharon. Esperaba con ansias mi tiempo en el coche cuando pude explorar lo que otras personas estaban descubriendo sobre desarrollo de software, pruebas, gestión y otros temas. En Muchas veces escuchaba un nuevo término, herramienta o acrónimo en una reunión, descargaba un podcast relacionado antes de salir del trabajo y devolvía el mensaje. al día siguiente tenga suficiente conocimiento para aplicar la información en el trabajo. Había descubierto una forma de aprender que disfrutaba, que se adaptaba a mi horario actual y que hacía que mi trabajo fuera más agradable. A través de podcasts Me presentaron muchas personas nuevas e ideas que han hecho mi experiencia laboral más exitosa y gratificante. Sí, dándome un Vehículo para apropiarme de mi aprendizaje personal, los podcasts cambiaron mi vida.

Busque en la comunidad global de pruebas y desarrollo de software más personas con quienes pueda intercambiar ideas y experiencias. Por ejemplo, la lista de correo de Agile Testing es un buen lugar para preguntar si otras personas han tenido el mismo problema que usted y cómo lo resolvieron. Las comunidades en línea como Software Testing Club son un excelente lugar para aprender participando en debates en foros y escribiendo en blogs sus propias experiencias. Las listas de correo y las redes sociales como Twitter pueden presentarle artículos y publicaciones de blogs sobre temas que le interesen.

Los proyectos de código abierto pueden ser un buen lugar para practicar tus habilidades, especialmente si quieres algo de experiencia en codificación. Puede perfeccionar otras habilidades a medida que contribuye a proyectos de código abierto, como realizar pruebas, redactar documentación de ayuda y ofrecer cursos de capacitación. (Consulte la sección de bibliografía de la Parte II “Cursos, conferencias, comunidades en línea, podcasts” para obtener enlaces a las actividades mencionadas en esta sección).

Si los evaluadores están integrados en los equipos de entrega de software de su empresa, una comunidad de práctica (CoP) de pruebas es un buen lugar para aprender y compartir experiencias con otras personas interesadas en las pruebas. Las sesiones de almuerzo y aprendizaje y los grupos de lectura dentro de una organización son otros mecanismos de aprendizaje eficaces. Por ejemplo, varios equipos han leído juntos nuestro libro anterior, leyendo un capítulo diferente cada semana. Pasan la hora del almuerzo o una reunión discutiendo lo que han leído y cómo se aplica.

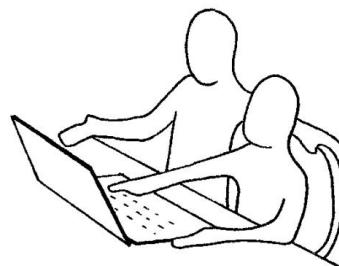


Figura 6-4 Emparejamiento: una excelente manera de aprender

a su situación (Ruhland, 2013a). ¡Una ventaja adicional de liderar el esfuerzo del club de diario de su equipo es perfeccionar sus propias habilidades de facilitación!

Únase a sus compañeros de equipo (consulte la Figura 6-4). Únase a personas ajenas a su equipo de desarrollo. El personal de marketing y ventas sabe mucho sobre sus clientes, al igual que sus diseñadores de experiencia de usuario (UX).

Las personas de otros departamentos, como recursos humanos (RR.HH.) o contabilidad, le brindarían una visión fresca y excelente de su producto. Hay mucho que puedes aprender de tus propios compañeros de trabajo y mucho que puedes enseñarles.

La historia de Lisa

Yo era parte de un equipo que decidió que trabajaríamos juntos en cada tarea de codificación y prueba. Cuando me emparejé con un programador para automatizar las pruebas de aceptación de la interfaz de usuario eso guiaría la codificación, noté con qué naturalidad el programador encontró duplicación en el código de prueba e inmediatamente lo trajimos en nuestro biblioteca de macros. Perfeccioné considerablemente mis habilidades de diseño de pruebas automatizadas como resultado.

Las demostraciones de iteración son otra oportunidad de aprendizaje útil. Intente rotar la tarea de demostrar lo que entregó el equipo. Es una excelente manera de practicar sus habilidades de presentación y facilitación.

Tiempo para aprender

Aprender lleva tiempo. Si corre constantemente sólo para mantenerse al día, no tendrá tiempo para aprender y probar nuevas ideas. En el Capítulo 2, “La importancia de la cultura organizacional”, hablamos sobre la importancia de



Figura 6-5 Tómese tiempo para aprender.

Fomentar una cultura de aprendizaje para que los miembros del equipo sientan que pueden tomarse el tiempo para investigar, experimentar con nuevas técnicas o simplemente pensar (ver Figura 6-5).

Bernice Niel Ruhland sugiere programar tiempo para leer u otras actividades de aprendizaje cuando tenga más energía. Por ejemplo, si eres una persona madrugadora, intenta levantarte un poco más temprano. Bernice dedica algunas horas de almuerzo y domingos por la tarde a la lectura.

Actualizar constantemente

Mike habla , software de Nueva Zelanda, explica cómo el aprendizaje es un desafío constante que todos deberían estar dispuestos a afrontar.

La vida laboral de la mayoría de las personas abarcará al menos 40 años. cuando miras en el campo del desarrollo de software y calcular cuánto cambio sucedió dentro de ese tiempo, es obvio que las habilidades de los graduados hoy se sentirán obsoletos cuando se jubilen. Solo mirando hacia atrás 10 o 20 años; es como entrar en otro mundo. Teléfonos inteligentes, tabletas, Internet de banda ancha (cosas que ya hemos empezado a dar por sentado) Todos estos son acontecimientos recientes.

Cuando comencé como programador hace 15 años, me dijeron que FOR-TRAN y C serían los únicos lenguajes que necesitaría, y estos Desde entonces, han sido reemplazados por Java, C++ y C#.

Lo que esto significa es que los profesionales del software no pueden simplemente deslizarse a lo largo de sus carreras con su conjunto de habilidades actuales. Nuevos desarrollos significará nuevos aprendizajes. Aferrarse al mantra "Siempre hemos hecho hacerlo de esta manera" no es suficiente. Una forma de seguir siendo relevante es encontrar formas de aprender continuamente y adoptar nuevas ideas.

Lo que realmente me gustaría que conste en acta es que aprender es en sí mismo un proceso ágil. No mire el aprendizaje como un proceso de "gran explosión": no sabes nada, lees un libro y dos días después eres un experto.

El aprendizaje, al igual que las funciones, es algo que se hace en iteraciones, añadiendo un poco más conocimiento a la vez y luego desarrollarlo en la siguiente iteración.

No es una carrera hasta el final y siempre hay algo más que hacer. aprender. A menudo, las personas que parecen adquirir conocimientos más lentamente son los que realmente lo están aprendiendo a un nivel mucho más profundo.

En mi libro (Talke, 2012), "Pensamientos ágiles"

habla sobre canales de aprendizaje y es realmente como mi informe de experiencia. sobre el aprendizaje.

Si su equipo supera la "Prueba Ácida Ágil" (Hendrickson, 2010) y entrega software con frecuencia a un ritmo sostenible, debería tener algo de tiempo libre fuera del trabajo para crecer profesionalmente. Así como los músicos practican sus instrumentos fuera de las actuaciones, todos necesitamos perfeccionar nuestras habilidades fuera del trabajo. Si amas lo que haces, esto es una alegría, no una carga. Como dice Steve Rogalsky, aprender más aumentará tu pasión y tu alegría.

Ayudar a otros a aprender

Todos los miembros del equipo, incluidos los evaluadores, pueden utilizar sus habilidades de entrenamiento y liderazgo para ayudar a sus compañeros a aprender.

Ganar confianza

Aldo Rall , gerente en sur un África, cuenta cómo ayudó nuevos miembros del equipo ganemos confianza.

Los nuevos evaluadores de nuestro equipo tenían demasiado miedo para participar en lo normal. Mecanismos colaborativos de trabajo en equipos ágiles. ellos también tenían

algunas desventajas culturales que los hacían reticentes a cuestionar o desafiar el pensamiento o proceso actual del equipo.

La organización tampoco estaba muy bien informada sobre qué pruebas implicado. Implementé mucha educación continua para los clientes, el analistas de negocios, los programadores y el director del proyecto para abrir sus mentes a diferentes maneras de hacer las cosas y crear conciencia sobre enfoques alternativos al desarrollo.

Como resultado de nuestros esfuerzos de capacitación, los evaluadores comenzaron a actuar como una comunidad de práctica. Los evaluadores compartieron problemas entre sí y debatieron las mejores soluciones para sus respectivos proyectos dentro de esta comunidad de práctica.

Se realizaron muchas más pruebas exploratorias. Los probadores ya no estaban tienen miedo de decir lo que piensan a sus equipos de proyecto. Los probadores fueron aceptado como equipo valorado, contribuyente, cooperador y colaborador miembros. Los programadores recibieron consejos de los evaluadores sobre los lanzamientos. en producción en serio. Los incidentes de producción se redujeron a un mínimo.

Es más, los evaluadores desarrollaron un atributo muy importante:
Tenía confianza en su capacidad para realizar pruebas.

Lograr que las personas superen su miedo a lo desconocido es un paso de gigante para capacitarlas para que tomen el control de su propio desarrollo profesional.

Como dice Aldo, la capacitación y el apoyo al aprendizaje también ayudan a las personas a acercarse y colaborar con otros miembros del equipo.



El aprendizaje es a menudo un efecto secundario importante de intentar una serie de pequeños experimentos para resolver un problema que está frenando al equipo.
¡Lo que aprenda puede ser más valioso que superar el obstáculo original!

Aprendizaje sorpresa

claire musgo , a probadoren el Unido Estados Unidos, comparte su historia de cómo dirigió a su equipo serie de pequeños experimentos para resolver un un problema. Generó oportunidades inesperadas para que el equipo mejore la comunicación y aprenda a recopilar actividades de poseer activamente prueba.

Como evaluador de un equipo Scrum multifuncional, conocía los defectos.

Como pensador visual, pongo notas adhesivas en una pequeña pizarra junto a mi

escritorio. Como nunca antes había usado un tablero como este, probé algunas categorías diferentes para prepararme mejor para hablar con los miembros de mi equipo de producto sobre los resultados de las pruebas. Aunque otras personas podían ver el tablero, siempre pensé que era una forma de mantener las cosas claras en mi cabeza. La Figura 6-6 muestra mi intento inicial.



Figura 6-6 Tablero de errores inicial

Un día, entré a la oficina y descubrí que mis compañeros de equipo habían reorganizado mis notas adhesivas, interactuando con mi pizarra. Me fascinó que encontraran interesante y útil el mantenimiento de mi registro personal. Al priorizar mi acumulación de defectos, mi equipo de propietarios de producto reveló su deseo de obtener más información sobre las pruebas. Aproveché la oportunidad para utilizar este gran gráfico visible como un medio para lograr una mejor comprensión compartida.

Presté mucha atención a los cambios que hicieron mis colegas e investigué un poco para ver cómo otros equipos ágiles utilizaban gráficos grandes y visibles. Así comenzó mi serie de experimentos.

La pregunta más común sobre los defectos parecía ser: "¿Qué tan grave es?". así que intenté reorganizar los errores según la gravedad (es decir, el impacto en el usuario). Dado que mis desarrolladores resolvieron escrupulosamente y rápidamente los peores problemas, terminamos con grandes cantidades de problemas de bajo impacto. Aunque nuestra comunicación definitivamente mejoró, todavía necesitábamos representar el criterio de la empresa sobre las prioridades. No todos los errores de una determinada gravedad son igualmente perjudiciales a los ojos de la empresa.

Para satisfacer esta necesidad, probé otra forma de ver el problema. Aunque quería brindar información más profunda sobre las fuentes de los errores, inicialmente informé los síntomas que enfrenta el usuario. Al utilizar el mapa del sitio que había creado para la planificación de pruebas, mostré una visión más amplia de dónde aparecían los errores. El equipo podría literalmente dar un paso atrás para ver grupos de problemas que podrían indicar la necesidad de rediseñar la experiencia de usuario o refactorizar el código. Consulte la Figura 6-7 para ver un ejemplo. Eso también funcionó por un tiempo.



Figura 6-7 Panel de errores: agrupado

Durante todo este tiempo, los miembros de nuestro equipo estaban aprendiendo a hablar más cómodamente sobre el trabajo de los demás. Cada uno de nosotros adoptó una forma más de T, ampliando nuestra comprensión de áreas fuera de nuestra especialización. Nos dimos cuenta de que la desconexión entre la acumulación de errores y la acumulación de historias de usuario era un problema que nos habíamos creado nosotros mismos. Entonces eliminamos el impedimento.

Comenzamos a detectar errores en los trabajos pendientes de sprint según la prioridad, que incluía el análisis del impacto en el usuario.

Incorporamos errores relacionados a nuestras historias para darnos más contexto en el cual ejecutar las historias de usuario que habíamos planeado. Eliminamos los errores durante una historia, ya sea corrigiéndolos o retrasándolos según lo consideráramos apropiado. Tener esta decisión deliberada nos ayudó a

centrarse en toda la cartera de productos para que ya no necesitáramos o Se valoró un tablero de errores separado.

Representar todo el trabajo de desarrollo del equipo como un único backlog fue

Un paso en la dirección correcta. Adaptamos nuestros grandes gráficos visibles después conversaciones sobre el valor que aportaban. Sin embargo, todos estos

Las iteraciones se centraron en comprender los informes de defectos, que habían sido la parte más visible de las actividades de prueba. Nuestro equipo estaba bien en el camino hacia un enfoque de prueba de todo el equipo, pero aún necesitábamos profundizar más profundo, más allá de los defectos. Nuestros grandes gráficos visibles nos ayudaron a encontrar el camino y abrió la comunicación del equipo para fomentar el colectivo propiedad de las pruebas.

Si ves a alguien a quien puedas ayudar, toma la iniciativa. Haga un gráfico grande y visible como lo hizo Claire, inicie una comunidad de práctica de pruebas, o tal vez un diario de pruebas o un club de lectura. A veces, una nueva idea es tan simple como ver y aprovechar una oportunidad.

Resumen

Hay mucho que considerar a medida que desarrollas las habilidades en forma de T necesarias para tener éxito en las pruebas. Amplíe su búsqueda para incluir oportunidades menos obvias, quizás fuera del campo del desarrollo de software.

- Conozca su estilo de aprendizaje preferido y utilícelo como base para optimizar las oportunidades de aprendizaje.
- Existe una amplia variedad de lugares para aprender. Prueba más de uno y vea cuáles funcionan para usted.
- Dedique tiempo para aprender.
- Pruebe experimentos para involucrar a todos los miembros del equipo; trabajar con las partes interesadas del negocio y ayudarlos a aprender lo que necesitan saber.
- Manténgase en contacto con las comunidades de prueba más grandes en grupos locales y de usuarios de Internet, y luego pruebe las nuevas ideas que obtenga.
- Dedique tiempo a practicar sus habilidades.
- Tome el control de su propio desarrollo profesional.

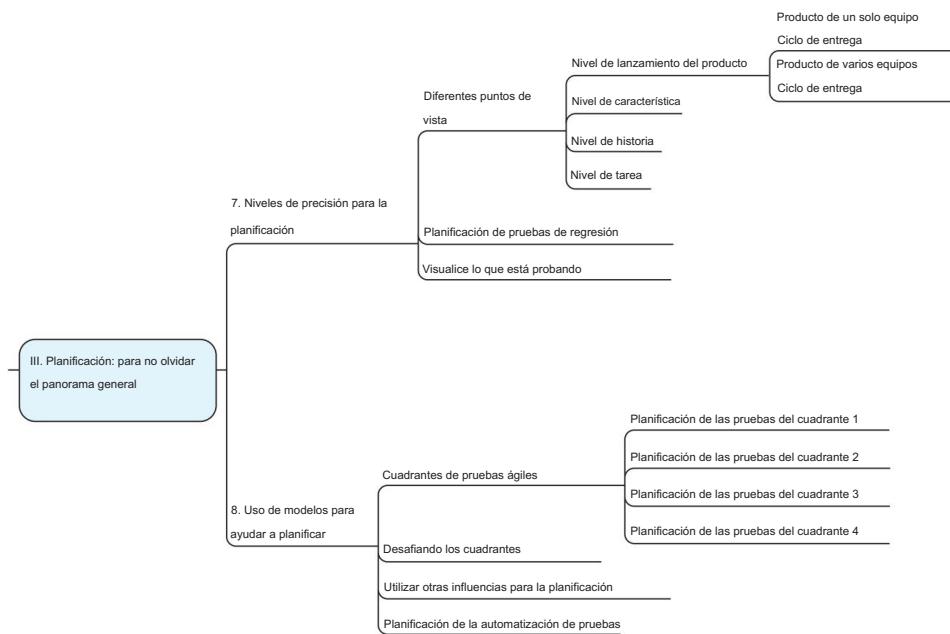
Esta página se dejó en blanco intencionalmente.

Parte III

Planificar, para que usted no lo haga Olvídese del panorama general

Nos sorprende cuántas veces todavía escuchamos: "Pero no necesitamos planificar ahora porque lo estamos haciendo de manera ágil". Estamos de acuerdo en que en el desarrollo ágil no hacemos una gran planificación inicial, todo en una sola fase. Pero la planificación es importante y forma parte de la metodología ágil. Planificamos según sea necesario, justo a tiempo. El pensamiento Lean se refiere a la idea del último momento responsable, para no perder el tiempo en detalles que aún no necesitas.

Dividimos esta parte en dos capítulos. En el Capítulo 7, abordamos la idea de niveles de precisión para la planificación: comprender lo que se necesita en el nivel en el que se está planificando actualmente. En el Capítulo 8, hablamos de cómo los diferentes modelos pueden ayudarnos a planificar nuestras pruebas. Cubrimos los cuadrantes de pruebas ágiles, no con el detalle que hicimos en Pruebas ágiles, sino para reiterar los conceptos detrás de ellos. Introducimos dos variaciones de los Cuadrantes que expresan algunos de los cambios en el pensamiento de los últimos años. Analizamos algunos de los otros modelos que pueden ayudar con la planificación, incluida la pirámide de automatización de pruebas.

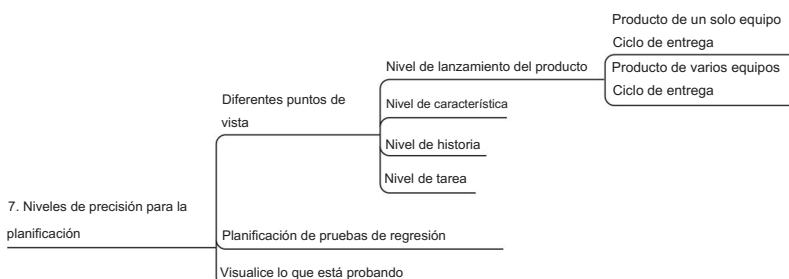


■ Capítulo 7, "Niveles de precisión para la planificación"

■ Capítulo 8, "Uso de modelos para ayudar a planificar"

Capítulo 7

Niveles de precisión para la planificación



Una percepción errónea frecuente sobre el desarrollo ágil es que no requiere planificación, porque el Manifiesto Ágil dice que los profesionales ágiles valoran responder al cambio en lugar de seguir un plan. En realidad, un buen equipo ágil tiende a planificar más que equipos de proyectos por fases y cerrados. Se hace en partes más pequeñas, con suficiente para lo que se necesita, y utiliza retroalimentación rápida para aprender y adaptarse. La planificación de pruebas es la misma: pequeños fragmentos y breves ciclos de retroalimentación. Mientras planifica las pruebas, concéntrese en lo que necesita saber ahora.

Diferentes puntos de vista

En equipos pequeños y ubicados en el mismo lugar, la planificación de las pruebas puede ser sencilla. El equipo considera los tipos de pruebas que necesitan realizar, tal vez utilizando un modelo como los cuadrantes de pruebas ágiles (consulte el Capítulo 8, “Uso de modelos para ayudar a planificar”) para ayudar a identificar qué pruebas deben realizarse o cuáles podrían ser sus requisitos de automatización. Los miembros del equipo pueden expresar sus necesidades de pruebas en sesiones de planificación o representarlas visualmente, tal vez en una pizarra.

En organizaciones grandes donde varios equipos trabajan en el mismo producto, la planificación de pruebas puede resultar complicada. Independientemente del tamaño de la empresa o del proyecto, se aplican los principios básicos.

En proyectos ágiles, no conocemos todos los detalles sobre cada característica al comienzo de un ciclo de entrega. Algunas pueden ser sencillas y predecibles porque ya hemos creado funciones similares antes. Otros pueden ser más complicados o pueden ser tan nuevos y desconocidos que sea difícil encontrar ejemplos de comportamiento deseado. Nuestro enfoque debe ser diferente del que supone que los requisitos se conocen desde el principio y donde la planificación se basa en esa información. Utilizando principios ágiles de simplicidad y desarrollo iterativo, analizamos los niveles de planificación (o niveles de precisión de detalle) y preguntamos qué información el equipo y las partes interesadas necesitan saber en cada nivel. Diferentes organizaciones tendrán diferentes contextos, por lo que necesita saber qué es importante para su equipo y su organización. Discutimos este enfoque en el contexto de iteraciones con límites de tiempo, pero los conceptos se pueden aplicar a métodos basados en flujo como kanban. Los equipos que utilizan un proceso basado en flujo pueden publicar después de que se complete cada historia, o pueden acumular las historias hasta que se complete una función y luego lanzarla a producción.

La Figura 7-1 muestra cuatro niveles de detalle que debemos considerar al planificar, teniendo al mismo tiempo en mente el producto completo (el sistema). Usamos los siguientes términos para estos diferentes niveles de precisión o puntos de vista para la planificación de pruebas:

- Lanzamiento del producto: uno o más equipos que trabajan en un solo producto y lo lanzan en intervalos específicos o en fechas definidas. El lanzamiento de un producto puede tener una o varias características.
- Característica: alguna capacidad empresarial o pieza de funcionalidad que sea útil para la empresa; podría ser parte de un conjunto de funciones más amplio. Una característica generalmente tiene muchas historias y la característica completa puede requerir múltiples iteraciones para completarse. Un equipo de entrega de funciones es el equipo multifuncional que trabaja en una o más funciones para un flujo de productos. Puede que sea el único equipo o quizás uno de los muchos equipos que trabajan para lograr un lanzamiento de producto más amplio.
- Historia (puede denominarse "historia de usuario"): una porción pequeña y comprobable de funcionalidad que generalmente se puede terminar ("hacer")

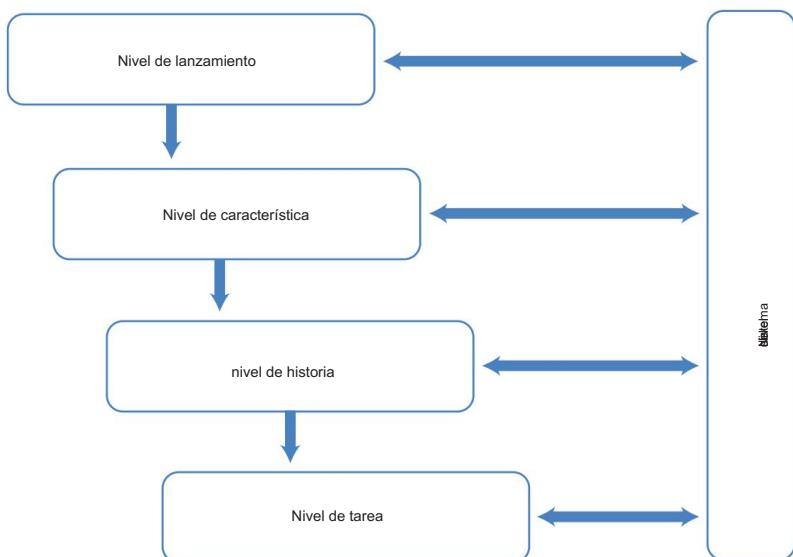


Figura 7-1 Niveles de detalle (precisión) para la planificación

dentro de uno a tres días. Puede o no lanzarse a producción por sí solo. Una historia tiene muchas tareas.

- **Tarea:** un trabajo que forma parte de una historia y que tarda menos de un día en completarse.

Para mantener la coherencia de los términos, supongamos que tenemos un lanzamiento de producto de tres meses (trimestralmente) con iteraciones de dos semanas para cada equipo. Hay varias funciones que cada equipo completa en el ciclo de lanzamiento. La Figura 7-2 muestra esta configuración.

Exploraremos cómo puede adaptar su planificación de pruebas a cada nivel, así como también analicemos qué documentación y/o artefactos podrían esperarse en cada uno de los cuatro niveles. En cada nivel, consideraremos un nivel de riesgo diferente.

Nivel de lanzamiento del producto

En el nivel de lanzamiento del producto, los equipos deben tener una buena idea de la visión del producto, tal vez mediante el uso de mapas de impacto como se analiza en el Capítulo 9.

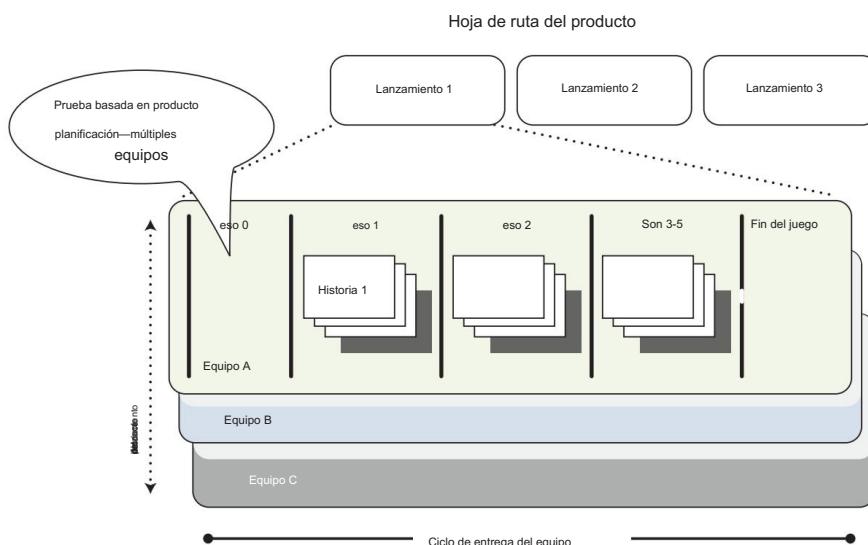


Figura 7-2 Niveles de precisión de la planificación para el nivel de producto

"¿Estamos construyendo lo correcto?" El enfoque de prueba de alto nivel debe cubrir lo que es importante para este ciclo de entrega de producto. El momento en el que planifica lo que se incluirá en la versión es el mejor momento para identificar la necesidad de nuevos entornos o herramientas de prueba.

Este también es un buen momento para pensar en las pruebas adicionales que se necesitarán realizar durante el final del juego, el tiempo antes de lanzar un producto a producción. Estas actividades pueden involucrar a más grupos fuera de su equipo de entrega, como operaciones, atención al cliente, capacitación y marketing. Por ejemplo, es posible que necesite realizar pruebas de aceptación del usuario final (UAT) o probar la integración con un sistema externo que no se pudo realizar en el entorno de prueba del equipo de entrega. Consulte las páginas 456–57 en Agile Test-ing y los enlaces en la bibliografía de la Parte III para obtener más información.



Si el lanzamiento incluye nueva tecnología o características que aún no se comprenden bien, el equipo podría planear realizar algunos picos, pequeños experimentos para aprender más sobre un posible problema o solución de diseño. Puede ser apropiado esperar a aprender de los picos antes de intentar planificar, incluso a un nivel alto.

Ciclo de entrega de productos de un solo equipo

Cuando su equipo es el único responsable de entregar la versión a sus clientes, su equipo es responsable de completar todas las actividades de prueba. Es posible que aún necesites recurrir a ayuda externa, como expertos en rendimiento, pero tu equipo es responsable. Si el suyo es el único equipo que trabaja en un producto, puede pasar a la siguiente sección sobre planificación de pruebas a nivel de funciones.

Ciclo de entrega de productos de varios equipos

Tener varios equipos trabajando en el mismo producto agrega complejidad y un enfoque de prueba general debe tener en cuenta las dependencias entre equipos o entre funciones. En este alto nivel, debería haber algunas personas que entiendan el panorama general y cómo esta versión se integra en el sistema en su conjunto.

Janet descubre que comenzar con un diagrama de contexto de alto nivel (consulte la figura 18-2 para ver un ejemplo) ayuda a visualizar los riesgos y proporciona un punto de partida para pensar en dependencias fuera de los equipos de productos o entre funciones. Al observar continuamente el panorama general, podemos intentar encontrar posibles problemas de integración tempranamente y no solo al final del ciclo de entrega. Esto reducirá el riesgo.

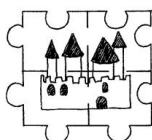
Considere el retorno de la inversión para pruebas como la interoperabilidad, la compatibilidad del navegador o las pruebas en diferentes dispositivos móviles. Por ejemplo, es posible que los equipos individuales no puedan manejar las pruebas de rendimiento y carga en sus entornos de prueba existentes. Considere los tipos de pruebas que podrían requerir un equipo de pruebas especializado, como la integración de sistemas. El Capítulo 18, “Pruebas ágiles en la empresa”, presenta ideas sobre cómo manejar las pruebas en grandes organizaciones ágiles.

Si se considera útil tener escrito un plan de prueba de lanzamiento del producto antes de comenzar el ciclo de entrega, le recomendamos que lo mantenga lo más simple y claro posible. No debe duplicar información en el documento de estrategia de prueba o enfoque de prueba de su organización (consulte Crispin y Greg-ory, 2009, p. 87). Ese documento contiene información que es común a todas las pruebas realizadas para su producto. Para la planificación de la prueba de lanzamiento de su producto actual, aborde solo lo que es diferente para esta versión en particular a alto nivel. Considere quién es la audiencia y qué es importante que sepan. Mientras menos tonterías haya, más probabilidades habrá de que vean la información pertinente sobre los riesgos potenciales en la nueva versión.

A nivel de lanzamiento del producto, la planificación de pruebas debe incluir la identificación de riesgos y suposiciones de prueba para el lanzamiento actual. Debería resaltar los problemas de prueba para posibles dependencias entre equipos, así como problemas de integración de productos.

Nivel de característica

A medida que las características que representan alguna capacidad de valor comercial se dividen en historias, debemos asegurarnos de que sean comprobables para que el equipo entre en una cadencia de codificación y prueba de cada historia individual hasta que la característica completa esté "lista". Recuerde, una función suele tener muchas historias y cada equipo de entrega se concentra en una función en particular. Puede haber varios equipos trabajando en paralelo y puede haber dependencias entre los equipos. La figura 7-3 muestra cómo pensar en las dependencias entre equipos.



Cada equipo de entrega trabaja con el propietario de su producto para determinar cuánto trabajo podría completarse durante el ciclo de entrega. A menudo, se lleva a cabo una sesión de planificación de lanzamientos en equipo para que los equipos puedan evaluar las características e historias en la cartera de productos de su equipo. Los evaluadores ayudan durante esta sesión de planificación haciendo preguntas aclaratorias que pueden ayudar al equipo a determinar el tamaño de la característica o historia. El equipo debe considerar los impactos en el sistema en su conjunto.

Los evaluadores también pueden ayudar cuando aplican una mentalidad exploratoria a los problemas de rendimiento o seguridad que puedan existir, o si una característica o historia afectará otras funciones. Al igual que con la planificación a nivel de lanzamiento, los equipos pueden posponer la planificación de las pruebas hasta que se hayan realizado los experimentos o las soluciones de pico necesarios y se sepa más sobre las funciones complejas.

Capture los riesgos y suposiciones de las pruebas a nivel de funciones que deben monitorearse en una pizarra, wiki u otro medio que sea visible tanto para el equipo de entrega como para los clientes. Experimente usando diagramas, mapas mentales, viñetas, una matriz u otro formato liviano y comprensible para todos. Observe cómo los equipos utilizan esta información y discútala en retrospectivas si encuentra que existen brechas de comunicación.

Busque la mejor opción para su situación.

A veces, los clientes o gerentes insisten en un documento de prueba formal que describa lo que su equipo va a probar en este ciclo de entrega. Antes de ti

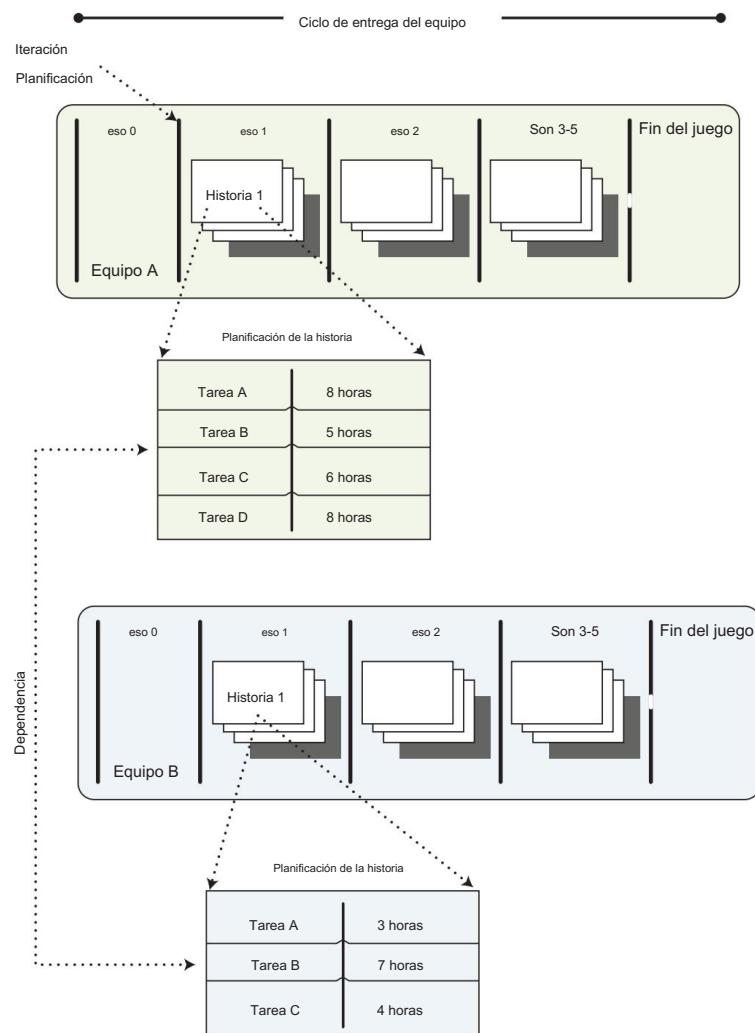


Figura 7-3 Niveles de precisión de la planificación a nivel de equipo

cree un entregable formal, piense quién lo usará y por qué lo quiere. Manténgalo lo más simple posible, preferiblemente menos de una página.

Dado que la funcionalidad aún no está configurada, evite entrar en detalles del alcance a menos que exista un riesgo que deba comunicarse a otros.

ers. Si no es necesario un documento, simplemente registre los riesgos de las pruebas y

suposiciones que deben ser monitoreadas en una pizarra que sea visible para el equipo.

En la conferencia Agile Testing Days de 2011, Huib Schoots contó una historia sobre un gerente resistente a la metodología ágil que exigió un documento de las pruebas de historia y características de alto nivel. En lugar de proporcionar lo que se esperaba (un documento), Huib creó un mapa mental para capturar esos detalles. El mapa mental satisfizo al gerente y le permitió a Huib mantener principios ágiles de simplicidad, lo que le permitió ceñirse a un plan de prueba de una página para capturar la información importante.

Bernice Niel Ruhland también es una gran admiradora del plan de prueba de una página (consulte la Figura 7-4 para ver un ejemplo). Se desafía a sí misma a limitarlo a una sola página como ejemplo para los evaluadores y, como gerente de práctica, creó una plantilla básica para que los evaluadores comenzaran. Eligen lo que necesitan y son libres de agregar lo que quieran, pero ella los entrena para encontrar el equilibrio adecuado.

Cuando llegue el momento de dividir la función en historias, trabaje con el propietario del producto de su equipo para crear pruebas de aceptación de alto nivel, utilizando ejemplos de comportamientos esperados y malos comportamientos para la función. Hacerlo ayudará a definir el alcance y mantener visible el valor del negocio. Los mapas mentales son herramientas poderosas para generar ideas de prueba en cualquier nivel, y consideramos valioso crearlos para cada característica. Intente crear un mapa mental de prueba de la función con todo su equipo antes de dividirlo en historias. Es una manera de revelar y dar la oportunidad de explorar algunos de los posibles problemas antes de que surjan. Utilice una herramienta de mapas mentales en línea si su equipo no está ubicado. La Figura 7-5 muestra un ejemplo de un mapa mental de prueba. Consulte el Capítulo 9, “¿Estamos construyendo lo correcto?”, para obtener más detalles sobre las formas de ayudar a los clientes a definir las características más valiosas y dividirlas en historias más pequeñas para su desarrollo.

La planificación de actividades de prueba para funciones grandes que contienen múltiples historias que cruzan iteraciones puede resultar complicada. Janet ha descubierto que los equipos se benefician al crear una historia de "Pruebe la característica" para estos. Las tareas de esta historia tratan principalmente de actividades de prueba, como "Explorar la función", "Realizar pruebas de carga" o "Automatizar el flujo de trabajo". El equipo de Lisa escribe cartas de pruebas exploratorias como historias en su herramienta de seguimiento de proyectos en línea. En el momento adecuado, cualquier miembro o pareja del equipo puede realizar estas cartas o las historias de prueba de funciones. Este tipo de historia asegura

Plan de prueba para la función de aplicación Fitness
Agregar calculadora de IMC

Alcance de la prueba

- Se está agregando una calculadora de IMC (índice de masa corporal) a la aplicación de fitness para calcular la grasa corporal de un individuo. Los datos ingresados por el usuario se almacenan en la nube y se comparten entre calculadoras, con la posibilidad de que el usuario cambie los datos dentro de cada calculadora. Las pruebas iniciales se realizarán utilizando un iPhone, iPad mini y una tableta Nexus.

Enfoque de prueba

- La funcionalidad de la calculadora de IMC se probará basándose en la identificación de las combinaciones y límites apropiados según los requisitos de IMC.

Se realizarán pruebas exploratorias en función de cómo los usuarios suelen utilizar la aplicación Fitness y el uso previsto de la nueva calculadora. Para esta versión, las pruebas se limitan a la funcionalidad, la integración, la exploración y la regresión. Las pruebas específicas del dispositivo, como batería baja u otras condiciones del dispositivo, no forman parte de esta prueba.

Riesgos

- La apariencia en todos los dispositivos puede no ser consistente. Es posible que sea necesario probar dispositivos adicionales además de la muestra inicial.

Regresión

- Se probarán las calculadoras que utilizan altura y/o peso para garantizar que los campos compartidos entre las calculadoras reflejen la información correcta y que los cálculos sean correctos.

Funciones para probar

- Calculadora de IMC (pruebas funcionales)
- Gráfico de IMC (prueba funcional)
- Informe de progreso de los resultados actuales en todas las calculadoras (pruebas de integración)
- Calculadora de BMR y Calculadora de grasa corporal (prueba de regresión)
- Pruebas entre dispositivos basadas en la usabilidad de las pantallas

Funciones que no se deben probar

- Calculadora de relación cintura-cadera
- Seguimiento e informe de objetivos de peso
- Plan de seguimiento de alimentos
- Opciones de configuración

Validación de datos

- Basado en escala métrica y estadounidense que utiliza altura y peso
- Identificar combinaciones y límites (bajo peso, normal, sobrepeso y obesidad; y rango de edad) para identificar los datos de las pruebas.
- Para otras calculadoras que usan altura y/o peso, identifique pruebas anteriores para comparar resultados al realizar pruebas de regresión. Identifique las pruebas a realizar en todas las calculadoras para garantizar que el informe de progreso de los resultados actuales refleje la información correcta.

Consideraciones Generales

- Las funciones no identificadas como parte del alcance de la prueba deben someterse a pruebas de humo antes de que comiencen las pruebas de integración y exploratorias.

Figura 7-4 Ejemplo de plan de prueba liviano

que no se olvide la prueba de funciones. Si su organización tiene un equipo de integración que realiza pruebas posteriores al desarrollo, este debería participar en la definición de estas historias y tareas de prueba.

También es posible que quieras pensar en desarrollar personajes y recorridos para obtener ideas de pruebas exploratorias adicionales y cobertura para cada característica o para el juego final. El capítulo 12, "Pruebas exploratorias", entra en más detalles sobre estas técnicas.

Nivel de historia

Una vez que estamos trabajando en el nivel de la historia, comenzamos a entrar en más detalles.

En este nivel de precisión, realmente no importa si está utilizando iteraciones con límites de tiempo o métodos basados en flujo. Para cada historia, necesitamos pruebas de aceptación de alto nivel: un ejemplo de comportamiento esperado y al menos un ejemplo de mala conducta que defina el alcance de la historia.

El capítulo 11, "Obtención de ejemplos", brinda más detalles sobre este tema. Nuestra planificación justo a tiempo para el nivel de la historia ocurre durante las sesiones de preparación de la historia (a veces llamada planificación previa o refinamiento del trabajo pendiente).

Nuestras pruebas para la historia evolucionan a partir de estas sesiones, así como de otras conversaciones, y luego pueden ampliarse durante la iteración. Comience a escribir ideas y variaciones de prueba durante las sesiones de preparación de la historia, continúe durante la planificación de la iteración y luego use todas las herramientas en su caja de herramientas para definir otras pruebas que demostrarán que la historia funciona como se esperaba.

Recuerde escribir ideas de pruebas exploratorias durante todo esto, de modo que las cartas de pruebas exploratorias estén listas cuando se complete el código.

Es probable que piense en pruebas adicionales a medida que explore más adelante.

Nivel de tarea

Los programadores practican el desarrollo basado en pruebas (TDD), escribiendo una prueba a la vez antes de codificar. Esta es una forma de diseñar, de planificar a nivel de detalle lo que van a construir.

Cuando piense en probar actividades a nivel de tarea, planifique menos y más haga y adapte. Por ejemplo, si su tarea es crear datos de prueba, es posible que tenga que cambiar el lugar donde obtiene los datos.

Algunos equipos optan por estimar las tareas en horas reales durante la planificación de iteraciones. Creemos que esto puede ser útil para los equipos que son nuevos en Agile. El objetivo no es ver con qué precisión puedes estimar, sino saber a dónde va tu tiempo.

En Pruebas ágiles, Capítulo 17, hablamos de varias formas de crear tareas de prueba, pero descubrimos que crear tarjetas de tareas separadas (virtuales, si está utilizando una herramienta de seguimiento en línea) para tareas de prueba independientes parece funcionar mejor. Cuando una tarea de prueba lleva mucho más tiempo de lo previsto, llame la atención sobre ella durante las reuniones diarias para pedir ayuda o para ver si afectará el resto de la historia. El equipo y el propietario del producto deben determinar la mejor solución para finalizar las actividades de prueba.

Planificación de pruebas de regresión

Las pruebas de regresión consisten en verificar que el sistema haga lo que hizo ayer. Las pruebas de regresión automatizadas que se ejecutan en un sistema de integración continua brindan retroalimentación rápida al equipo. A medida que el equipo prueba historias y características a nivel de unidad y de negocio, muchas (si no la mayoría) de las pruebas se agregarán a las suites de regresión. Su equipo necesita planificar cómo mantener las suites actualizadas determinando qué pruebas se actualizarán a medida que cambie o agregue funcionalidad. Cuando elimina funciones, también debe recordar eliminar las pruebas correspondientes.

Las pruebas de regresión deben organizarse por funcionalidad del sistema para que pueda ubicarlas fácilmente para cambiarlas, actualizarlas o eliminarlas, o usarlas para demostrarle a alguien cómo se comporta el sistema. Muchos equipos ágiles utilizan una herramienta de seguimiento en línea para mantener sus historias y algunos adjuntan pruebas a las historias. Esto funciona cuando se prueba dentro de una iteración, pero no se puede mantener a largo plazo. Muchos equipos descubren esto por las malas cuando intentan localizar pruebas seis meses después de que la historia esté "terminada".

Ya sea que sus pruebas sean manuales o automatizadas, deben migrarse de una agrupación basada en historias a una organización basada en la funcionalidad del producto o el área comercial.

Algunos equipos agregan las pruebas de la historia al conjunto de regresión tan pronto como completan una historia; algunos esperan hasta que se complete la función. Algunos equipos

agregue sus pruebas al final de cada iteración y decida cuáles proporcionan una cobertura suficientemente valiosa para conservar. Las pruebas a nivel de unidad creadas como parte de TDD se agregan automáticamente a medida que se registran en el control de versiones junto con el código que admiten. Su equipo debe experimentar con diferentes formas de crear y mantener conjuntos de regresión, evaluando el retorno de la inversión a intervalos regulares.

Visualice lo que está probando

Descubrimos que las ayudas visuales ayudan a mantener a todo el equipo al tanto de las actividades de prueba que deben realizarse y a realizar un seguimiento de lo que aún queda por hacer. Como se mencionó anteriormente en este capítulo, una forma eficaz de visualizar durante las pruebas de lluvia de ideas es el mapeo mental.

Empiece por colocar su tema o problema central en el “nodo” principal y, cuando piense en temas relacionados, dibuje nuevos nodos en una jerarquía adecuada. No existe una manera correcta o incorrecta; simplemente capturar las ideas sin criticarlas. Tanto las pizarras blancas como las herramientas de mapas mentales en línea le permiten mover fácilmente los nodos a medida que avanza, y muchas herramientas en línea tienen un marcador de finalización de tareas para marcar el progreso. La codificación de colores es una forma eficaz de mostrar relaciones. Podría terminar con un mapa mental como el de la Figura 7-5.

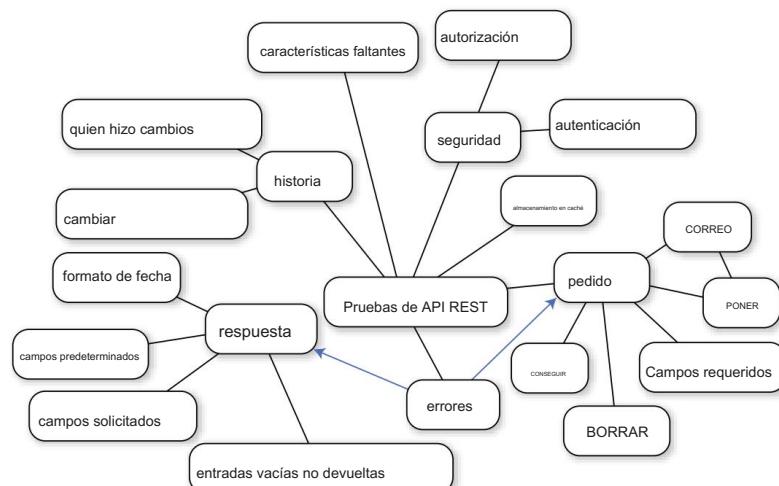


Figura 7-5 Mapa mental de ideas de prueba

Un equipo ubicado puede mantener el mapa mental de pruebas en el área del equipo, tener conversaciones sobre él y actualizarlo, marcando las actividades de prueba completadas. Los equipos distribuidos pueden utilizar una herramienta de mapas mentales en línea, aunque se necesitará más práctica y disciplina para desarrollar el hábito y mantenerlo virtualmente visible. Un mapa mental de pruebas mantiene el estado de las pruebas en primer plano y recuerda a todo el equipo qué pruebas quedan por hacer.

Otra forma eficaz de capturar ideas de prueba de alto nivel a nivel de lanzamiento es una matriz de prueba. Se pueden utilizar técnicas colaborativas como los mapas mentales para generar ideas de prueba para una matriz de prueba a nivel de versión, como se muestra en la Figura 7-6. Imágenes como estas proporcionan un punto de vista diferente sobre las actividades de prueba planificadas. Las filas son características y las columnas son condiciones/capacidades de prueba. En el ejemplo de la Figura 7-6, el patrón muestra progreso: blanco significa no probado; a cuadros (o verde, si es de color) significa listo para comenzar; los puntos (amarillos, si son de color) significan que se han realizado algunas pruebas, pero es posible que se necesiten más; rayado (rojo, si es de color) significa roto; gris significa no aplicable, no necesita pruebas.

Muchas cosas comprando Versión 1.5	Integración	Móvil	Cálculos	Diseño	Localización	Dispositivos	Integración	Gamificación	Entrenamiento	Seguridad
Almacenar información del cliente	[checkered]	[checkered]								
Añadir al carrito de compras	[yellow]	[green]	[green]					[yellow]	[red]	
Calcular los costos de envío										
Vista móvil iOS únicamente										
Siguiente característica										

Leyenda

Bueno para ir	[checkered]
Algunas pruebas; podría usar más	[yellow]
Problema mayor	[red]
No se hicieron pruebas	
No aplica	[grey]

Figura 7-6 Matriz de prueba de nivel de versión

Independientemente de cómo visualice las pruebas planificadas, intente mantenerlas simples para que puedan seguir el ritmo de su equipo. Utilice gráficos grandes y visibles o sus sustitutos virtuales para que todos los miembros del equipo y las partes interesadas puedan ver las pruebas completadas y planificadas de un vistazo. Y hágalo valioso para su equipo o dirección; después de todo, lo que realmente quieras es probar y no documentar.

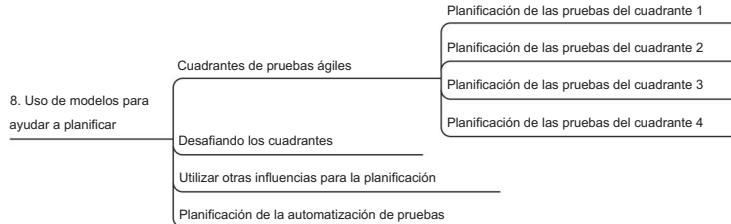
Resumen

En este capítulo, hablamos sobre comprender qué nivel de detalle se necesita para la planificación de pruebas, así como la importancia de la visualización.

- A nivel de lanzamiento del producto, piense en los impactos del sistema y qué pruebas deben realizarse durante el final del juego, incluidas, entre otras, las pruebas finales de UAT, carga y rendimiento.
- Si forma parte de varios equipos que trabajan en el nivel de lanzamiento del producto, considere también las dependencias de alto nivel entre componentes y equipos, y lo que podrían incluir las pruebas del sistema de integración.
- Para la planificación de su equipo de entrega a nivel de lanzamiento, piense en las dependencias de características o historias, riesgos, suposiciones y entornos de prueba.
- A nivel de característica, considere los impactos que afectan esta característica, las pruebas de aceptación y los tipos de pruebas necesarias, incluidos los atributos de calidad, las pruebas de usabilidad y las pruebas de aceptación del usuario.
- A nivel de historia, crear pruebas de aceptación, las pruebas de historia restantes, incluidas las pruebas exploratorias y tal vez probar algunos de los atributos de calidad.
- A nivel de tarea, la planificación tiende a ocurrir mientras se realiza la tarea, como TDD: escribir primero la prueba y luego el código.
- Las pruebas de regresión automatizadas no surgen por casualidad. Es necesario planificarlo y ampliarlo como parte de su trabajo diario.
- Encuentre una forma sencilla de hacer visibles sus pruebas, ya sea que estén almacenadas electrónicamente o en una sala de equipo o en un pasillo.

Capítulo 8

Uso de modelos para ayudar a planificar



A medida que el desarrollo ágil se vuelve cada vez más común, existen técnicas establecidas que los profesionales experimentados utilizan para ayudar a planificar actividades de prueba en proyectos ágiles, aunque los equipos menos experimentados a veces malinterpretan o hacen mal uso de estos útiles enfoques. Además, los avances en herramientas y marcos de prueba han alterado un poco los modelos originales que se aplicaban a principios de la década de 2000. Los modelos nos ayudan a ver las pruebas desde diferentes perspectivas. Veamos algunos fundamentos de la planificación ágil de pruebas y cómo están evolucionando.

Cuadrantes de pruebas ágiles

Los cuadrantes de pruebas ágiles (los Cuadrantes) se basan en una matriz que Brian Marick desarrolló en 2003 para describir los tipos de pruebas utilizadas en proyectos de Programación Extrema (XP) (Marick, 2003). Hemos descubierto que los Cuadrantes son bastante útiles a lo largo de los años mientras planificamos con diferentes niveles de precisión. Algunas personas han entendido mal el propósito de los Cuadrantes. Por ejemplo, pueden verlos como actividades secuenciales en lugar de una taxonomía de tipos de pruebas. Otras personas no están de acuerdo sobre qué actividades de prueba pertenecen a cada cuadrante y evitan usar los cuadrantes por completo. Nos gustaría aclarar estos conceptos erróneos.

La Figura 8-1 es la imagen que utilizamos actualmente para explicar este modelo. Notarás que hemos cambiado parte del texto desde que lo presentamos en Agile.

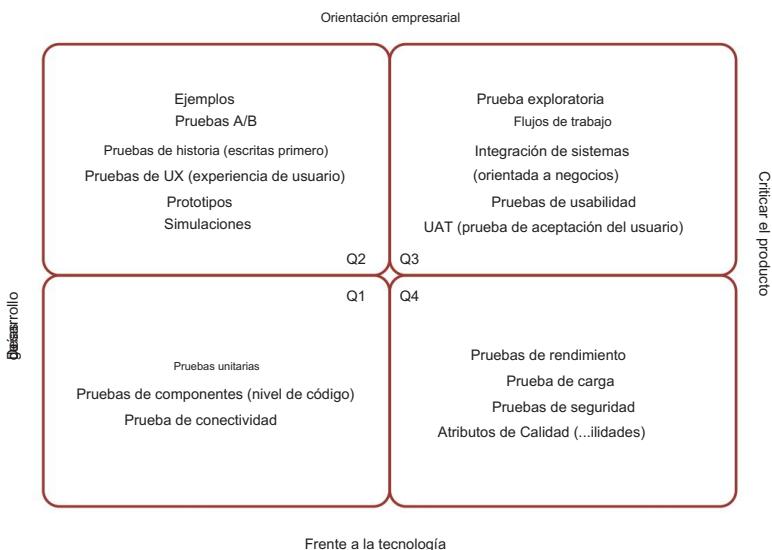


Figura 8-1 Cuadrantes de pruebas ágiles

Pruebas. Por ejemplo, ahora decimos “guiar el desarrollo” en lugar de “apoyar el desarrollo”. Esperamos que esto lo aclare.

Es importante comprender el propósito detrás de los Cuadrantes y la terminología utilizada para transmitir sus conceptos. El sistema de numeración de cuadrantes no implica ningún orden. No se trabaja en los cuadrantes del 1 al 4 de forma secuencial. Es un sistema de numeración arbitrario, de modo que cuando hablamos de los cuadrantes, podemos decir "Q1" en lugar de "pruebas tecnológicas que guían el desarrollo". Los cuadrantes son

- P1: pruebas tecnológicas que guían el desarrollo
- P2: pruebas empresariales que guían el desarrollo
- P3: pruebas comerciales que critican (evalúan) el producto
- P4: pruebas tecnológicas que critican (evalúan) el producto

El lado izquierdo de la matriz del cuadrante trata sobre la prevención de defectos antes y durante la codificación. El lado derecho trata de encontrar defectos y descubrir características faltantes, pero entendiendo que queremos encontrarlos lo más rápido posible. La mitad superior trata de exponer las pruebas al

negocio, y la mitad inferior trata sobre pruebas que son más internas para el equipo pero igualmente importantes para el éxito del producto de software.

“Enfrentar” simplemente se refiere al lenguaje de las pruebas; por ejemplo, las pruebas de rendimiento satisfacen una necesidad empresarial, pero la empresa no podría leer las pruebas; están preocupados por los resultados.



La mayoría de los equipos ágiles comenzarían especificando las pruebas del segundo trimestre, porque ahí es donde se obtienen los ejemplos que se convierten en especificaciones y pruebas que guían la codificación. En sus publicaciones de blog de 2003 sobre la matriz, Brian llamó a las pruebas Q2 y Q1 “ejemplos comprobados”. Originalmente los había llamado ejemplos de “guía” o “entrenamiento” y le da crédito a Ward Cunningham por el adjetivo “comprobado”. Los miembros del equipo construirían un ejemplo de lo que el código necesita hacer, verificarían que aún no lo haga, obligarían al código a hacerlo y verificarían que el ejemplo ahora sea verdadero (Marick, 2003). Incluimos prototipos y simulaciones en la Q2 porque son pequeños experimentos que nos ayudan a comprender una idea o concepto.

En algunos casos, tiene más sentido comenzar a probar una nueva característica utilizando pruebas de un cuadrante diferente. Lisa ha trabajado en proyectos en los que su equipo utilizó pruebas de rendimiento para determinar la arquitectura, porque ese era el atributo de calidad más importante para la característica. Esas pruebas caen en el cuarto trimestre. Si sus clientes no están seguros de sus requisitos, podría incluso hacer una investigación y comenzar con pruebas exploratorias (T3). Considere dónde podría estar el mayor riesgo y dónde las pruebas pueden agregar el mayor valor.

La mayoría de los equipos utilizan simultáneamente técnicas de prueba de todos los cuadrantes, trabajando en pequeños incrementos. Escriba una prueba (o verificación) para una pequeña parte de una historia, escriba el código y, una vez que la prueba pase, tal vez automatice más pruebas. Una vez que las pruebas (verificaciones automáticas) hayan pasado, utilice pruebas exploratorias para ver qué se omitió. Realice pruebas de seguridad o de carga, luego agregue la siguiente pequeña porción y realice todo el proceso nuevamente.

Michael Hütermann añade “de afuera hacia adentro, sin barreras, colaborativo” al medio de los cuadrantes (ver Figura 8-2). Utiliza el desarrollo impulsado por el comportamiento (BDD) como ejemplo de pruebas sin barreras. Estas pruebas están escritas en un lenguaje natural y ubicuo “dado_cuando_entonces” al que pueden acceder tanto los clientes como los desarrolladores e invita a la conversación.

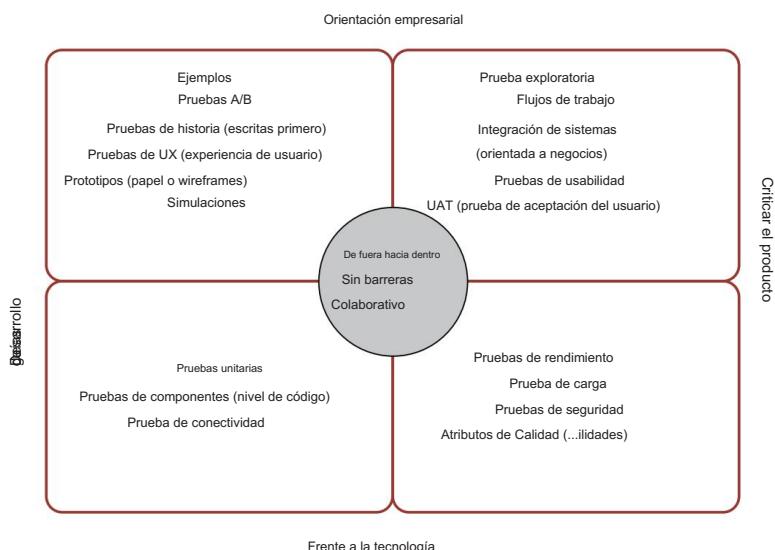


Figura 8-2 Cuadrantes de pruebas ágiles (con la adaptación de Michael Hüttermann)

entre la empresa y el equipo de entrega. Este formato se puede utilizar para la verificación del primer y segundo trimestre. Consulte el artículo Agile Record de Michael (Hüttermann, 2011b) o su libro Agile ALM (Hüttermann, 2011a) para obtener más ideas sobre cómo aumentar los cuadrantes.

Los Cuadrantes son simplemente una taxonomía o modelo para ayudar a los equipos a planificar sus pruebas y asegurarse de que tengan todos los recursos que necesitan para realizarlas. No existen reglas estrictas sobre lo que sucede en cada cuadrante. Adapte el modelo de Cuadrantes para mostrar qué pruebas debe considerar su equipo. Haga que las pruebas sean visibles para que su equipo piense primero en las pruebas mientras planifica el lanzamiento, las funciones y la historia. Esta visibilidad expone los tipos de pruebas que se están realizando actualmente y la cantidad de personas involucradas. Úselo para provocar debates sobre las pruebas y en qué áreas quizás desee dedicar más tiempo.



Al analizar los cuadrantes, es posible que se dé cuenta de que hay pruebas necesarias que su equipo no ha considerado o que le faltan ciertas habilidades o recursos para poder realizar todas las pruebas necesarias. Por ejemplo, un equipo en el que trabajó Lisa se dio cuenta de que estaban tan concentrados en convertir

ejemplos de negocios en las pruebas del segundo trimestre que guían el desarrollo que estaban ignorando por completo la necesidad de realizar pruebas de rendimiento y seguridad. Agregaron historias de usuarios para investigar qué capacitación y herramientas necesitarían y luego presupuestaron el tiempo para realizar esas pruebas del cuarto trimestre.

Planificación de las pruebas del cuadrante 1

A principios de la década de 1990, Lisa trabajó en un equipo en cascada cuyos programadores debían escribir planes de pruebas unitarias. Los planes de pruebas unitarias definitivamente eran excesivos, pero pensar en las pruebas unitarias con anticipación y automatizarlas todas fue una gran parte de la razón por la que nunca se notificaron errores críticos al centro de soporte. Los equipos ágiles no planifican las pruebas del primer trimestre por separado. En el desarrollo basado en pruebas (TDD), también llamado diseño basado en pruebas, las pruebas son una parte inseparable de la codificación. Una pareja de programadores puede sentarse y discutir algunas de las pruebas que quieren escribir, pero los detalles evolucionan a medida que evoluciona el código. Estas pruebas unitarias guían el desarrollo, pero también apoyan al equipo en el sentido de que un programador las ejecuta antes de registrar su código y se ejecutan en el CI en cada registro de código.

Hay otros tipos de pruebas técnicas que pueden considerarse como guías para el desarrollo. Puede que no sean obvios, pero pueden ser fundamentales para que el proceso siga funcionando. Por ejemplo, digamos que no puede realizar las pruebas porque hay un problema con la conectividad. Cree un script de prueba que pueda ejecutarse antes de la prueba de humo para asegurarse de que no haya problemas técnicos. Otra prueba que los programadores podrían escribir es una para verificar la configuración predeterminada. Muchas veces estos problemas no se conocen hasta que comienza la implementación y las pruebas.

Planificación de las pruebas del cuadrante 2



Las pruebas del segundo trimestre ayudan con la planificación a nivel de característica o historia. La Parte IV, "Prueba del valor empresarial", explorará cómo guiar el desarrollo con pruebas más detalladas de cara al negocio. Estas pruebas o ejemplos comprobados se derivan de la colaboración y conversaciones sobre lo que es importante para el artículo o la historia. Tener a las personas adecuadas en una sala para responder preguntas y dar ejemplos específicos nos ayuda a planificar las pruebas que necesitamos. Piense en los niveles de precisión discutidos en el capítulo anterior; Las preguntas y los ejemplos se vuelven más precisos a medida que profundizamos en los detalles de las historias.

El proceso de obtener ejemplos y crear pruebas a partir de ellos fomenta

colaboración entre roles y puede identificar defectos en forma de suposiciones ocultas o malentendidos antes de escribir cualquier código.

Muestre a todos, incluso a los dueños de negocios, lo que planea probar; Fíjate si estás defendiendo algo sagrado o si les preocupa que te estés perdiendo algo que tiene valor para ellos.

La creación de pruebas del segundo trimestre no se detiene cuando comienza la codificación. Los equipos de Lisa han descubierto que funciona bien comenzar con pruebas de camino feliz. A medida que la codificación comienza y las pruebas del camino feliz comienzan a pasar, los evaluadores y programadores desarrollan las pruebas para abarcar condiciones límite, pruebas negativas, casos extremos y escenarios más complicados.

Planificación de las pruebas del cuadrante 3 Las

pruebas siempre han sido fundamentales para el desarrollo ágil, y guiar el desarrollo con pruebas del segundo trimestre orientadas al cliente se hizo popular desde el principio entre los equipos ágiles. A medida que los equipos ágiles han madurado, también han adoptado las pruebas del tercer trimestre, en particular las pruebas exploratorias. Cada vez más equipos están contratando profesionales expertos en pruebas exploratorias, y los evaluadores de equipos ágiles están dedicando tiempo a ampliar sus habilidades exploratorias.



La planificación de las pruebas del tercer trimestre puede ser un desafío. Podemos comenzar a definir caracteres de prueba antes de que haya un código completo para explorar. Como explica Elisabeth Hendrickson en su libro ¡Exploralo! (Hendrickson, 2013), los estatutos nos permiten definir dónde explorar, qué recursos llevar con nosotros y qué información esperamos encontrar. Para que sean efectivas, algunas pruebas exploratorias pueden requerir la finalización de varias historias de usuarios pequeñas o esperar hasta que se complete la función. También es posible que necesites dedicar tiempo a crear las personas de usuario que podrías necesitar para las pruebas, aunque es posible que ya se hayan creado en el story-mapping u otros ejercicios de planificación de funciones. Definir cartas de pruebas exploratorias no siempre es fácil, pero es una excelente manera de compartir ideas de pruebas con el equipo y poder realizar un seguimiento de las pruebas que se completaron. Daremos ejemplos de dichos estatutos en el Capítulo 12, "Pruebas exploratorias", donde analizamos diferentes técnicas de pruebas exploratorias.

Una estrategia para dedicar tiempo a las pruebas exploratorias es escribir historias para explorar diferentes áreas de una característica o diferentes personajes. Otro

La estrategia, que prefiere Janet, es tener una tarea para pruebas exploratorias para cada historia, así como una o más para probar la característica. Si su equipo utiliza una definición de "hecho", realizar pruebas exploratorias adecuadas podría ser parte de eso. Puede dimensionar historias individuales asumiendo que dedicará una cantidad significativa de tiempo a realizar pruebas exploratorias. Tenga en cuenta que, a menos que se asigne tiempo específicamente durante la creación de la tarea, las pruebas exploratorias a menudo se ignoran.

El tercer trimestre también incluye pruebas de aceptación del usuario (UAT). La planificación de UAT debe realizarse durante la planificación del lanzamiento o lo antes posible. Incluya a sus clientes en la planificación para decidir la mejor forma de proceder. ¿Pueden venir a la oficina para probar cada característica nueva? Tal vez estén en un país diferente y necesites compartir la computadora. Trabaje para obtener la retroalimentación más frecuente y rápida posible de todas sus partes interesadas.

Planificación de las pruebas del cuadrante 4

Las pruebas del cuadrante 4 pueden ser las más fáciles de pasar por alto en la planificación, y muchos equipos tienden a centrarse en las pruebas para guiar el desarrollo. Las actividades del cuadrante 3, como la UAT y las pruebas exploratorias, pueden ser más fáciles de visualizar y, a menudo, resultan más familiares para la mayoría de los evaluadores que las pruebas del cuadrante 4. Por ejemplo, más equipos necesitan respaldar su aplicación a nivel global, por lo que las pruebas en el espacio de internacionalización y localización se han vuelto importantes. Los equipos ágiles han luchado por saber cómo hacer esto; Incluimos algunas ideas en el Capítulo 13, "Otros tipos de pruebas".

Algunos equipos hablan de atributos de calidad con criterios de aceptación en cada historia de una característica. Preferimos utilizar la palabra restricciones. En Discover to Deliver (Gottesdiener y Gorman, 2012), Ellen Gottesdiener y Mary Gorman recomiendan utilizar el Planguage de Tom y Kai Gilb (su lenguaje de planificación; consulte la bibliografía de la Parte III, "Planificación para no olvidar el panorama general") para hablar de estas limitaciones de una manera muy definida (Gilb, 2013).

Si su producto tiene una restricción como "Cada pantalla debe responder en menos de tres segundos", no es necesario repetir ese criterio para cada historia. Encuentre un mecanismo para recordarle a su equipo cuando esté discutiendo la historia que esta restricción debe incorporarse y probarse. Liz Keogh describe una técnica para escribir pruebas sobre

cómo se pueden monitorear capacidades como el rendimiento del sistema (Keogh, 2014a). Las organizaciones generalmente saben qué sistemas operativos o navegadores admiten al comienzo de una versión, así que agréguelos como restricciones e inclúyelos en sus estimaciones de prueba. Estos tipos de atributos de calidad suelen ser buenos candidatos para probarlos a nivel de características, pero si tiene sentido probarlos a nivel de historia, hágalo allí; Piense: "Pruebe temprano". El Capítulo 13, "Otros tipos de pruebas", cubrirá algunos tipos de pruebas diferentes con los que quizás haya tenido dificultades.

Desafiando los cuadrantes

A lo largo de los años, muchas personas han cuestionado la validez de los Cuadrantes o los han ajustado ligeramente para que sean más significativos para ellos. Decidimos compartir un par de estas historias porque creemos que es valioso cuestionar continuamente lo que "sabemos" que es verdad. Así es como aprendemos y evolucionamos para mejorar y satisfacer las demandas cambiantes.

El desafío de Gojko a los cuadrantes

Gojko Adzic , autor y entrega de software estratégico y
El sultante cuestiona la validez de la Cuadrantes en la corriente
entrega de software era.

El modelo de cuadrantes de pruebas ágiles es probablemente lo único que todos recuerdan del libro original. Era una herramienta de pensamiento increíblemente útil en el mundo de la entrega de software en aquel entonces, 2008. Él me ayudó a facilitar muchas discusiones útiles sobre el panorama general que falta desde el punto de vista de la calidad de los programadores típicos, y ayudó a muchos evaluadores averiguar en qué concentrarse. El mundo actual, a partir de 2014, luce significativamente diferente. Ha habido un aumento en la popularidad de los programas continuos, entrega, DevOps, análisis de Big Data, entrega lean startup y pruebas exploratorias. El modelo Quadrants necesita una actualización importante.

Uno de los problemas con el modelo de Cuadrantes original es que era fácilmente malentendido como una secuencia de tipos de pruebas, especialmente aquella Hay algún tipo de división entre las cosas de antes y las de después. desarrollo.

Este problema es incluso peor ahora que en 2008. Con el aumento de la popularidad de la entrega continua, la línea divisoria se está volviendo más borrosa. y está desapareciendo. Con iteraciones más cortas y entrega continua, generalmente es difícil trazar la línea entre las actividades que apoyan el equipo y aquellos que critican el producto. ¿Por qué el rendimiento

¿Las pruebas no estarán destinadas a apoyar al equipo? ¿Por qué las pruebas funcionales no critican el producto? ¿Por qué la UAT está separada de las pruebas funcionales? Siempre me pareció difícil justificar la dimensión horizontal de los cuadrantes, porque criticar el producto puede apoyar al equipo de manera bastante efectiva si se hace de manera oportuna. Por ejemplo, la especificación por ejemplo ayuda a los equipos a fusionar completamente las pruebas funcionales y UAT en algo que se verifica continuamente durante el desarrollo.

Muchos equipos con los que trabajé recientemente realizaron pruebas de rendimiento durante el desarrollo, principalmente para no estropear las cosas con cambios frecuentes. Estos son sólo dos ejemplos en los que las cosas del lado derecho de los cuadrantes ahora se utilizan más para apoyar al equipo que cualquier otra cosa.

Con los métodos de puesta en marcha ajustada, los productos reciben muchas críticas incluso antes de que se escriba una sola línea de código de producción.

Dividir las pruebas entre aquellas que apoyan el desarrollo y aquellas que evalúan el producto ya no ayuda realmente a facilitar discusiones útiles, por lo que necesitamos un modelo diferente, en particular, uno que ayude a abordar el eterno problema de los llamados requisitos no funcionales. Lo que para muchas personas en realidad significa: "Va a ser una discusión difícil, así que no la tengamos". El antiguo modelo de Cuadrantes coloca a las "ilidades" en un cuadrante de pruebas técnicas en gran parte olvidado después del desarrollo. Pero aspectos como la seguridad, el rendimiento, la escalabilidad, etc., no son realmente técnicos; implican muchas expectativas comerciales, como cumplimiento, cumplimiento de acuerdos de niveles de servicio, manejo de cargas máximas esperadas, etc. Tampoco son realmente no funcionales, ya que implican bastantes funciones como cifrado, almacenamiento en caché y distribución del trabajo. Por supuesto, esto se complica por el hecho de que algunas expectativas en esas áreas no son tan fáciles de definir o probar, especialmente las incógnitas. Si las tratamos como preocupaciones puramente técnicas, las expectativas comerciales a menudo no se declaran ni se verifican explícitamente. En lugar de ser no funcionales, estas preocupaciones suelen ser disfuncionales. Y aunque muchas "ilidades" son difíciles de probar antes de que el software entre en contacto con sus usuarios reales, la aparición de técnicas de prueba divididas A/B en los últimos cinco años ha hecho que verificarlas sea relativamente fácil, barato y de bajo riesgo. cosas en producción.

Otro aspecto de las pruebas que no se capta bien en los Cuadrantes del primer libro es el aumento de la popularidad y la importancia de las pruebas exploratorias. En el modelo antiguo, las pruebas exploratorias son algo que ocurre desde la perspectiva empresarial para evaluar el producto (a menudo mal entendido como posterior al desarrollo). En muchos contextos, bien documentados en el libro de Elisabeth Hendrickson sobre pruebas exploratorias (Hendrickson, 2013) y en el libro de James Whittaker (Whittaker et al., 2012), las pruebas exploratorias pueden ser increíblemente

Software

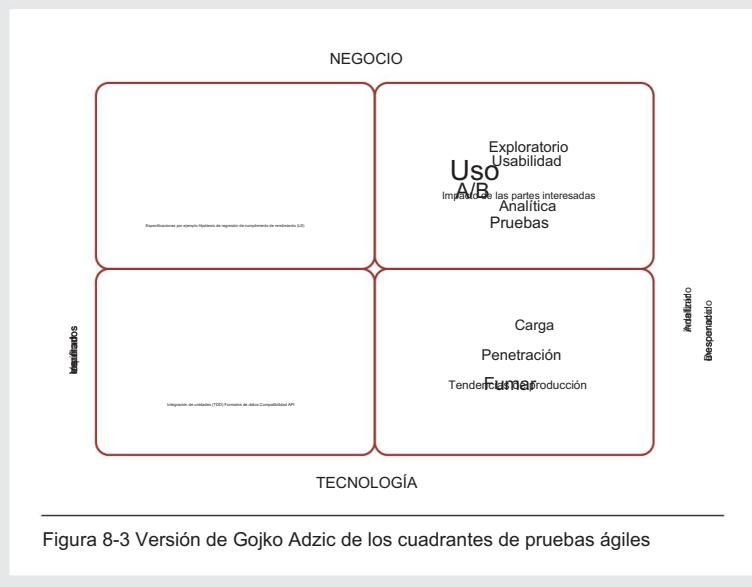
Cómo Google Pruebas

También es útil desde la perspectiva técnica y, lo que es más importante, es algo que debe hacerse durante el desarrollo.

El tercer aspecto que no se capta bien en los primeros cuadrantes es la posibilidad de cuantificar y medir los cambios de software mediante análisis de uso en producción. El aumento de la popularidad del análisis de Big Data, especialmente combinado con modelos de puesta en marcha eficiente y entrega continua, permite a los equipos probar de forma relativamente económica cosas que hace diez años eran muy costosas de probar; por ejemplo, los verdaderos impactos en el rendimiento. Cuando seguimos el modelo original, las pruebas de rendimiento serias a menudo implicaban tener una copia completa del hardware del sistema de producción.

Hoy en día, muchos equipos eliminan el riesgo de esos problemas con cambios continuos más pequeños y menos riesgosos, cuyo impacto se mide directamente en un subconjunto del entorno de producción. Muchos equipos también analizan las tendencias de sus registros de producción para detectar rápidamente problemas inesperados y previamente desconocidos.

Necesitamos cambiar el modelo (Figura 8-3) para facilitar todas esas discusiones, y creo que la división horizontal actual ya no ayuda. La comunidad de pruebas impulsadas por el contexto sostiene con mucha fuerza que buscar los resultados esperados no es realmente probar; en cambio, a eso lo llaman control. Sin entrar en una discusión sobre lo que es o no es prueba, encontré que la división es bastante útil para muchas discusiones recientes con clientes. Quizás ese sea un segundo eje más útil para el modelo: la diferencia entre buscar resultados esperados y analizar incógnitas, aspectos sin una respuesta definitiva de sí o no, donde los resultados requieren una interpretación analítica hábil. De todos modos, la mayor parte de la innovación en estos días parece ocurrir en la segunda parte. Verificar los resultados esperados, tanto desde una perspectiva técnica como comercial, es ahora un problema prácticamente resuelto.



Pensar en verificar los resultados esperados versus analizar resultados que no estaban predefinidos ayuda a explicar varios aspectos importantes.

Problemas que enfrentan los equipos de entrega de software hoy en día:

Las preocupaciones de seguridad podrían dividirse fácilmente en pruebas funcionales de cumplimiento, como cifrado, protección de datos, autenticación, etc.

activado (esencialmente todas las comprobaciones de resultados esperados predefinidos), y penetración/investigaciones (no predefinidas). Esto ayudará a involucrarse el equipo de entrega y los patrocinadores comerciales en una discusión más útil sobre describir la parte funcional de la seguridad desde el principio.

Las preocupaciones sobre el desempeño podrían dividirse en escenarios comerciales en ejecución para demostrar los niveles de servicio y la capacidad acordados, de forma continua. estilo de entrega (predefinido) y pruebas de carga (¿dónde se romperá?). Este ayudará a involucrar al equipo de entrega y a la empresa en la definición de las expectativas de desempeño y evitará que las personas traten el desempeño como algo una preocupación puramente técnica. Evitando el apoyo al equipo/evaluación las divisiones de productos, permitimos una discusión sobre el desempeño de ejecución pruebas en diferentes entornos y en diferentes momentos.

La exploración sería mucho más visible y podría ser claramente dividido entre pruebas exploratorias técnicas y orientadas al negocio.

Esto puede respaldar una discusión sobre pruebas exploratorias técnicas que los desarrolladores deben realizar o que los evaluadores pueden ejecutar reutilizando las existentes. marcos de automatización. También puede apoyar un debate general sobre qué debería incluirse en las pruebas exploratorias orientadas a los negocios.

Las pruebas de productos de construcción, medida y aprendizaje encajarían muy bien en el modelo, y el modelo facilitaría una discusión significativa sobre cómo esas pruebas requieren una hipótesis definida y en qué se diferencia eso de simplemente sacar las cosas para ver qué sucede a través del análisis de uso.

Podemos facilitar una conversación sobre cómo detectar problemas desconocidos. monitoreando los registros de producción como una forma de probar continuamente inquietudes técnicas que son difíciles de verificar y costosas de automatizar antes del despliegue, pero sigue siendo útil para apoyar al equipo. Moviéndose alejar la discusión del apoyo al desarrollo o la evaluación de producto para comprobar las expectativas o inspeccionar lo desconocido. También sería una buena manera de diferenciar esas pruebas de los análisis de uso de producción orientados al negocio.

Lo más importante es que al utilizar un eje horizontal diferente, podemos elevar Conciencia sobre toda una categoría de cosas que no encajan en los planes o informes de prueba típicos, pero que aún así son increíblemente valiosas. Los primeros Los cuadrantes fueron útiles porque crearon conciencia sobre un todo categoría de cosas en la esquina superior izquierda que la mayoría de los equipos no estaban Realmente estoy pensando pero ahora se toman como sentido común. La década de 2010 Los cuadrantes deben ayudarnos a crear conciencia sobre algunas cuestiones más importantes de hoy.

	CONFIRMAR	INVESTIGAR
NEGOCIO	ORIENTADO AL NEGOCIO EXPECTATIVAS	RIESGOS EXTERNOS ATRIBUTOS DE CALIDAD
TECNOLOGÍA	TECNOLOGÍA- FRENTE A EXPECTATIVAS	RIESGOS INTERNOS ATRIBUTOS DE CALIDAD

Figura 8-4 Versión de Elisabeth Hendrickson de los cuadrantes de pruebas ágiles

Elisabeth Hendrickson también presentó una alternativa a los Cuadrantes existentes en su charla sobre “The Thinking Tester” (Hendrickson, 2012).

Es similar a la versión de Gojko pero tiene un aspecto diferente. Puede ver en la Figura 8-4 que volvió a etiquetar las columnas verticales para “confirmar” e “investigar”, mientras que las filas horizontales todavía representan negocios y tecnología.

El cuadrante superior izquierdo representa las expectativas del negocio, que podrían adoptar la forma de especificaciones ejecutables (automatizadas). Otros pueden estar representados por prototipos en papel o estructuras alámbricas. En la parte superior derecha se encuentran las pruebas que ayudan a investigar los riesgos relacionados con la calidad externa del producto. Es muy parecido a la idea del cuadrante original de pruebas exploratorias, escenarios o pruebas de usabilidad. Al igual que el modelo de Gojko, el cuadrante inferior derecho destaca los riesgos del funcionamiento interno del sistema.



Ambos modelos alternativos aportan valor. Creemos que hay espacio para múltiples variaciones para adaptarse a un espectro de necesidades. Por ejemplo, las organizaciones que son capaces de adoptar la entrega continua son capaces de pensar en este espacio, pero a muchas organizaciones les faltan años para lograrlo. Consulte la bibliografía de la Parte III para obtener enlaces a modelos de cuadrantes de prueba adicionales. Úselos para asegurarse de que su equipo cubra todo

los diferentes tipos de pruebas que necesita para ofrecer el valor adecuado a sus clientes.

Utilizar otras influencias para la planificación

Hay muchos modelos e ideas útiles que nos ayudarán en la planificación de nuestras pruebas y no deberíamos desecharlos. Como han dicho Tim Ottinger y Jeff Langr (Ottinger y Langr, 2009b), sigue siendo útil una mnemónica para pensar en los llamados requisitos no funcionales. El modelo FURPS (ver Figura 8-5) fue desarrollado en Hewlett-Packard y fue elaborado públicamente por primera vez por Grady y Caswell (Wikipedia, 2014f); ahora se usa ampliamente en la industria del software. El + se añadió posteriormente al modelo después de varias campañas en HP para ampliar el acrónimo y enfatizar varios atributos.

James Whittaker desarrolló una metodología que llama matriz de capacidad de componentes de atributos (ACC) (Whittaker, 2011) para ayudar a definir qué probar en función del riesgo. ACC consta de tres partes diferentes que definen el sistema bajo prueba: atributos, componentes y capacidades. Él los define como:

- Los atributos (adjetivos del sistema) son cualidades y características que promocionan el producto y lo distinguen de la competencia; algunos ejemplos son "Rápido", "Seguro", "Estable" y "Elegante".
- Componentes (sustantivos del sistema) son bloques de construcción que juntos constituyen el sistema en cuestión. Algunos ejemplos de

FURPS+	
Funcionalidad	Más:
Usabilidad	Restricciones de diseño
Fiabilidad	Requisitos de implementación
Actuación	Requisitos de interfaz
Soportabilidad	requisitos físicos

Figura 8-5 Tarjeta flash FURPS+ (Ottinger y Langr, 2011)

Los componentes son "Firmware", "Impresión" y "Sistema de archivos" para un proyecto de sistema operativo, o "Base de datos", "Carrito" y "Explorador de productos" para un sitio de compras en línea.

- Capacidades (verbos del sistema) describen las capacidades de un Componente particular para satisfacer los Atributos del sistema. Un ejemplo de capacidad para un sitio de compras podría ser "Procesa transacciones monetarias mediante HTTPS". Puede ver que esto podría ser una capacidad del componente "Carrito" cuando intenta cumplir con el atributo "Seguro". El aspecto más importante de las capacidades es que se pueden probar.

Crear una matriz de alto nivel utilizando este modelo puede ser una forma sencilla de visualizar su sistema. La Figura 8-6 muestra un ejemplo de cómo podría verse dicha matriz. Gojko Adzic está de acuerdo en que exponer las características del sistema y proporcionar más visibilidad es definitivamente una buena idea (Adzic, 2010a), aunque advierte que si bien podemos aprender de otros campos, debemos tener cuidado al usarlos como metáfora del desarrollo de software.

Utilice heurísticas como el "Truco de heurística de prueba" de Elisabeth Hendrickson. Sheet" (Hendrickson, 2011) o técnicas probadas y verdaderas como diagramas de estado o tablas de verdad para pensar en nuevas ideas para atributos. Combine estas ideas con modelos como los Cuadrantes para que las conversaciones sobre las limitaciones del sistema o la usabilidad puedan extraer ejemplos claros. Utilizar todas las herramientas de su caja de herramientas sólo puede ayudar a aumentar la calidad del producto.

Componentes		Capacidades	Atributos		
Firmware de aplicaciones móviles	Impresión		Rápido	Seguro	Estable
		Administrar perfil			
		Enviar mensajes			
		Actualizar red			
ÁREA DE INFLUENCIA			RIESGO / IMPORTANCIA		

Figura 8-6 Ejemplo de ACC

Planificación de la automatización de pruebas

Desde que Mike Cohn ideó su pirámide de automatización de pruebas en 2003, muchos equipos han encontrado en ella un modelo útil para planificar su automatización de pruebas.

Para aprovechar la retroalimentación rápida, debemos considerar en qué nivel deben estar nuestras pruebas de automatización. Cuando miramos la pirámide estándar, Figura 8-7, vemos tres niveles.

El nivel más bajo es la base: las pruebas unitarias. Cuando consideramos realizar pruebas, debemos intentar llevarlas lo más bajo posible para obtener el mayor retorno de la inversión (ROI) y la retroalimentación más rápida.

Sin embargo, cuando tenemos una lógica empresarial en la que las pruebas deben ser visibles para la empresa, debemos utilizar herramientas colaborativas que creen pruebas en la capa de servicio (la API) para especificarlas de una manera que documente el comportamiento del sistema. Consulte el Capítulo 16, "Patrones y enfoques de diseño de automatización de pruebas", para

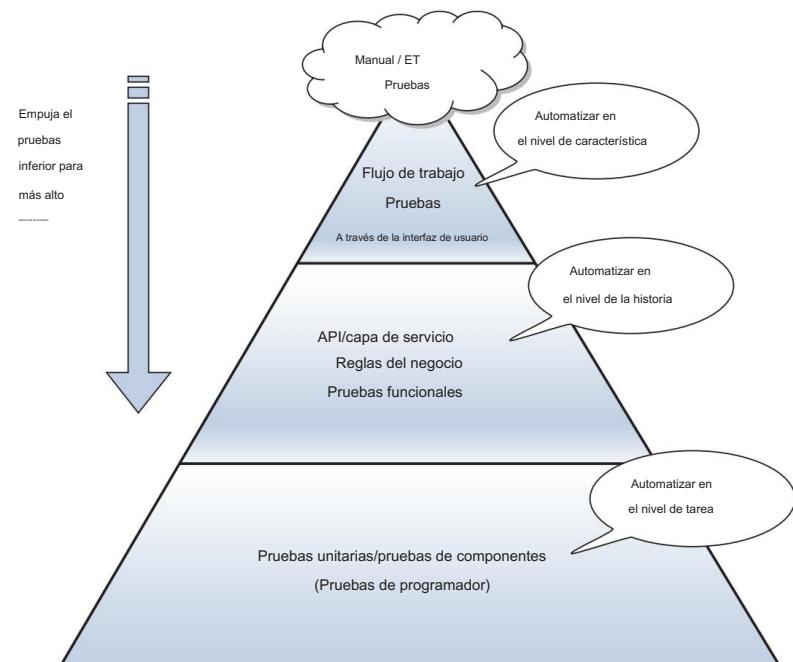


Figura 8-7 Pirámide de automatización

más detalles. Es en esta capa donde podemos automatizar a nivel de historia para que las pruebas y la automatización puedan mantenerse al día con la codificación.

La capa superior de la pirámide consta de las pruebas de flujo de trabajo a través de la interfaz de usuario (UI). Si tenemos un alto grado de confianza en las pruebas unitarias y en las pruebas de nivel de servicio o de nivel API, podemos mantener al mínimo estas pruebas automatizadas más lentas y frágiles. Consulte el Capítulo 15, "Pirámides de la automatización", para obtener más detalles sobre modelos piramidales alternativos.

Prácticas como guiar el desarrollo con ejemplos pueden ayudar a definir cuál es el mejor nivel para la prueba. La cadencia de un equipo puede establecerse en función de qué tan bien planifiquen y ejecuten su automatización y qué tan bien comprendan el nivel de detalle que necesitan. Considere también cómo hacer que sus pruebas de automatización sean visibles, ya sea que se muestren en el entorno de integración continua o en un monitor abierto.

Resumen

Los modelos son una herramienta útil para la planificación. En este capítulo, cubrimos los siguientes puntos:

- Los cuadrantes de pruebas ágiles proporcionan un modelo para pensar sobre las pruebas en un mundo ágil.
- Los cuadrantes ayudan a enfatizar la responsabilidad de todo el equipo en las pruebas.
- Proporcionan un mecanismo visible para hablar sobre las pruebas, necesario.
- El lado izquierdo trata de guiar el desarrollo, aprender qué construir y prevenir defectos: realizar pruebas tempranas.
- El lado derecho trata de criticar el producto, encontrar defectos y aprender qué capacidades aún faltan.
- Gojko Adzic ofrece una forma alternativa de pensar sobre la Cuadrantes si se encuentra en un entorno de startup eficiente o de entrega continua.
- También presentamos un diagrama de cuadrante alternativo de Elisa-beth Hendrickson que destaca los controles de confirmación versus las pruebas de investigación.

- Ya existen muchas herramientas en nuestra caja de herramientas de pruebas ágiles y podemos combinarlas con otros modelos, como los Cuadrantes, para que nuestras pruebas sean lo más efectivas posible.
- FURPS y ACC son ejemplos adicionales de modelos que puede utilizar para ayudar a planificar en función del riesgo y una variedad de características de calidad.
- La pirámide de automatización es un recordatorio para pensar en la automatización. ción y planificarla en los diferentes niveles.

Esta página se dejó en blanco intencionalmente.

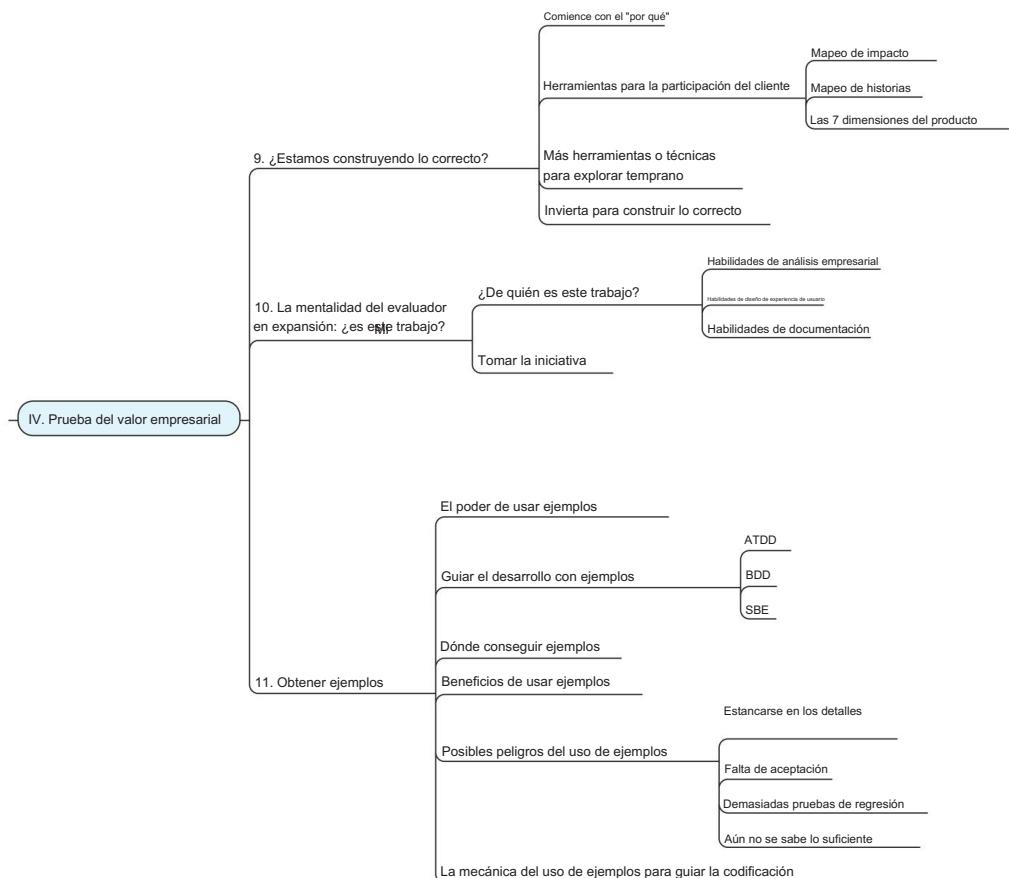
Parte IV

Prueba del valor empresarial

En su libro ¡Exploralo! (Hendrickson, 2013), Elisabeth Hendrickson describe dos facetas de una estrategia de prueba (págs. 5-6). Según Elisabeth, las "verificaciones" son pruebas que verifican que "la implementación se comporta según lo previsto en las configuraciones y condiciones admitidas". Ella explica que la exploración es una prueba que implica explorar las áreas que los controles no cubren, "diseñar y ejecutar pequeños experimentos en rápida sucesión utilizando los resultados del último experimento para informar el siguiente". En opinión de Elisabeth, una estrategia de prueba integral necesita tanto ser comprobada como explorada.

La distinción entre comprobar y explorar es interesante, pero no estamos seguros de que la definición de prueba vaya lo suficientemente lejos. Las pruebas son más que "solo" software de prueba. Se trata de probar ideas, ayudar a los expertos en negocios a identificar las características más valiosas para desarrollar a continuación, encontrar un entendimiento común para cada característica, prevenir defectos y probar el valor comercial.

Un enfoque de calidad de todo el equipo significa incorporar actividades de prueba desde el momento en que se conceptualiza una característica por primera vez. En esta parte, cubriremos algunas herramientas y técnicas que nos permiten ayudar a los clientes a centrarse en funciones que les ayuden a alcanzar sus objetivos comerciales más valiosos. Los evaluadores hacen contribuciones importantes durante estas actividades y señalaremos lo que pueden aprender de otros especialistas para ayudar a los clientes a articular requisitos y ejemplos del comportamiento deseado del sistema. También entraremos en detalles sobre cómo guiar el desarrollo con ejemplos y buenas formas de obtener esos ejemplos.

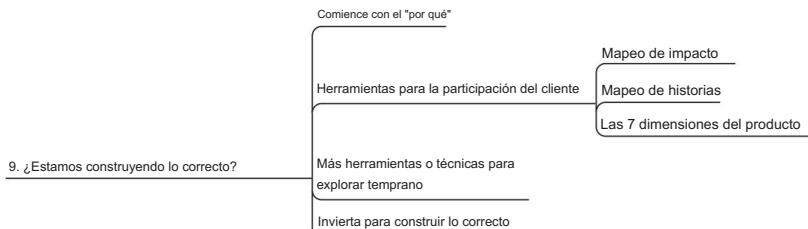


■ Capítulo 9, “¿Estamos construyendo lo correcto?” ■

Capítulo 10, “La mentalidad del evaluador en expansión: ¿es este mi trabajo?” ■ Capítulo 11, “Obtención de ejemplos”

Capítulo 9

¿Estamos construyendo el derecho? ¿Cosa?



La mayoría de nosotros sentimos que nunca hay tiempo suficiente para realizar pruebas. Pero la verdad es que probar el software es sólo una parte de la ecuación. Desafortunadamente, a menudo perdemos tiempo desarrollando software que no satisface las necesidades de mayor prioridad de los clientes o software que nunca usarán, o simplemente entregamos productos cuyo costo excede su valor para la empresa.

Muchos equipos en los que hemos trabajado dominan las prácticas de desarrollo y aprendieron a entregar aplicaciones sólidas. La mayoría de los errores o desviaciones reportados resultaron ser requisitos omitidos o faltantes.

Cuando los equipos experimentan con mejores formas de hacer a sus clientes las preguntas correctas desde el principio, comprenden más claramente el propósito de cada característica y son más capaces de desarrollar lo que los clientes realmente necesitan.

Piense en ello como una prueba temprana o una prueba del valor comercial. Al igual que con todas las actividades de prueba, todo nuestro equipo de entrega debe asumir la responsabilidad de asegurarse de que sepamos lo que nuestros clientes realmente necesitan.

Comience con el "por qué"

Según nuestra experiencia, una de las formas más importantes en que los evaluadores pueden ayudar tanto a sus clientes como a sus equipos de entrega es centrándose en el propósito de cada posible nueva característica o historia de usuario. Cuando empezamos a preguntar por qué los clientes quieren la característica (el problema que están tratando de resolver), estamos

más probabilidades de construir lo correcto. A medida que analizamos los objetivos que debe cumplir la característica y cómo podemos medir el éxito, podemos obtener buenos ejemplos del comportamiento deseable del sistema. Los equipos de desarrolladores y clientes pueden esbozar una hoja de ruta y establecer hitos para ayudarnos a saber cuándo hemos terminado la función.

Una discusión sobre el “por qué” a menudo lleva a la conclusión de que el cliente no está pidiendo lo que realmente necesita. Si simplemente construyéramos lo que pide el cliente, el negocio podría verse afectado. He aquí un ejemplo.

La historia de Lisa



En una reunión de estimación de software de servicios financieros, el propietario de nuestro producto leyó esta historia: “Como administrador externo, quiero cargar un préstamo, con mi propio plazo, tasa de interés y monto, y poder generar una cuenta corregida. acción activa para liquidar los fondos y procesar el cheque de salida”.

Sin entrar en muchos detalles sobre lo que eso significa, reconocí instantáneamente que el PO nos estaba dando un “qué” e incluso nos estaba diciendo cómo implementar una nueva característica, sin decirnos por qué la empresa la quería.

Nuestra aplicación ya contaba con un sistema de préstamos con todas las funciones, por lo que claramente algo andaba extraño. Le pregunté: “¿Cuál es el propósito de esta historia? ¿Qué problema empresarial se está resolviendo?

Resultó que administradores externos usaban nuestro sistema pero tenían reglas diferentes sobre los préstamos y diferentes términos y tasas de interés. No podían utilizar nuestro sistema de préstamos tal como está. El PO estaba tratando de encontrar una manera económica de darles a esos usuarios lo que necesitaban. Se aferró a una función que habíamos desarrollado poco antes, que implicaba cargar archivos para distribuir fondos.

Una vez que entendimos el por qué de la historia, la reescribimos. “Como administrador externo con reglas de préstamo especiales, quiero poder solicitar, en nombre de un participante 401(k), un préstamo que no esté sujeto a las validaciones normales del sistema, para que el participante pueda recibir los fondos del préstamo y pagar el préstamo”.

La mejor manera de implementar la función fue actualizar nuestro sistema existente para tener validaciones especiales para préstamos cada vez que un administrador externo solicitara un préstamo en nombre de un cliente. No se requirió ningún código nuevo para procesar el préstamo y desembolsar los fondos.

Tanto la funcionalidad como el retorno de la inversión son componentes clave del valor comercial del software. Siga pensando en crear “lo correcto” y haga las preguntas necesarias para mantener alta la calidad del producto.

Utilice Real Options para ayudarle a optimizar el proceso de decidir qué

solución para entregar. Si su equipo de entrega utiliza buenas prácticas para poder implementar nuevas funciones rápidamente, puede tomarse más tiempo para saber qué opción será más valiosa (Matts y Maassen, 2007).

Herramientas para la participación del cliente

Existen muchos ejercicios efectivos para ayudar a las partes interesadas y a los equipos de ejecución a descubrir qué cosas deben construir. Propusimos algunas herramientas para obtener ejemplos y requisitos en Agile Testing (págs. 153-164), incluidas listas de verificación, mapas mentales, hojas de cálculo, maquetas, diagramas de flujo y wikis.

Aquí sugerimos algunas prácticas adicionales que consideramos valiosas para comprender las necesidades del cliente y utilizarlas para crear pruebas que guíen el desarrollo. Experimente con ellos y vea cuál funciona mejor para su equipo para ayudar a generar calidad en su producto.

Mapeo de impacto

En su libro sobre mapeo de impacto (Adzic, 2012), Gojko Adzic explica cómo los equipos de entrega y las partes interesadas del negocio pueden colaborar para identificar rápidamente caminos alternativos hacia los entregables que brinden el mejor valor. Al responder las preguntas "¿Por qué?" "¿OMS?" "¿Cómo?" ¿y qué?" Los equipos construyen una hoja de ruta que les ayuda a aprender rápidamente si un enfoque particular producirá los resultados deseados y será adaptable a los cambios inevitables.

Los mapas de impacto han sido adaptados de otras herramientas de planificación y lluvia de ideas, como los mapas mentales y los mapas de historias. Es una forma sencilla y estructurada de centrar al equipo en el propósito de qué crear y ayuda a identificar las características más valiosas que se deben crear primero. El foco cambia de "¿Qué vamos a hacer?" a "¿Por qué hacemos esto, cuál es nuestro objetivo, quién puede ayudarnos y quién se interpone en nuestro camino?" Esto también ayuda a determinar si un problema en particular puede resolverse mejor fuera del software. Por ejemplo, la solución podría ser una mejor formación para los usuarios del sistema.

Según nuestra experiencia, las partes interesadas del negocio tienden a decirnos qué quieren que implementemos: "Crea una interfaz como X, pero haz que en su lugar haga Y". Incluso pueden pensar que nos están ahorrando tiempo al reutilizar los existentes.

código. Una vez que entendamos el propósito, el equipo podría proponer una implementación técnica simple.

Trabajaremos con un ejemplo para ayudarle a comprender. Digamos que somos una startup de un sitio minorista de Internet y las partes interesadas de nuestro negocio quieren alentar a nuevos clientes y mantener a los existentes. Vienen a nosotros con una característica:

Obligue a los clientes a crear una cuenta cuando realicen el pago, para que podamos guardar su información de facturación, envío y pago.

Nos gustaría entender mejor lo que quiere la empresa. Reunimos a nuestros expertos comerciales con el equipo de entrega para crear un mapa de impacto. Primero, preguntamos a los expertos en negocios por qué quieren esta función y descubrimos que quieren retener a los clientes. Trabajamos juntos para crear una meta específica, medible, viable, realista y oportuna. En este caso, el objetivo es alcanzar una tasa de retención de clientes del 90% en los próximos tres meses.

A continuación, analizamos quién puede ayudarnos a lograr o influir en nuestra capacidad para lograr ese objetivo. Podrían ser personas que puedan ayudarnos, pero también deberíamos pensar en quién podría interponerse en nuestro camino. Puede haber muchas otras partes interesadas (actores) además de nuestro equipo de entrega, como el departamento de marketing y los proveedores que suministran nuestros productos.

Una vez identificados, queremos pensar en cómo estos diferentes actores ayudan o dificultan nuestra capacidad para lograr nuestro objetivo. El marketing podría idear un programa de fidelización atractivo. Nuestros diseñadores de experiencia de usuario (UX) podrían hacer que nuestro sitio sea agradable de usar. Nuestros expertos en seguridad podrían encontrar formas de brindarles a nuestros clientes la confianza de que nuestro sitio es seguro. Estos "cómo" son los impactos que los diferentes actores pueden tener en el objetivo del negocio.

Después de identificar los impactos, abordamos lo que nosotros y los actores que identificamos podemos lograr para hacer avanzar el negocio hacia la meta. Estos son los entregables. En nuestro escenario de ejemplo, un resultado podrían ser cupones creados por nuestro grupo de marketing para fomentar la repetición de compras. Otro producto entregable podría ser una interfaz segura y fácil de usar donde

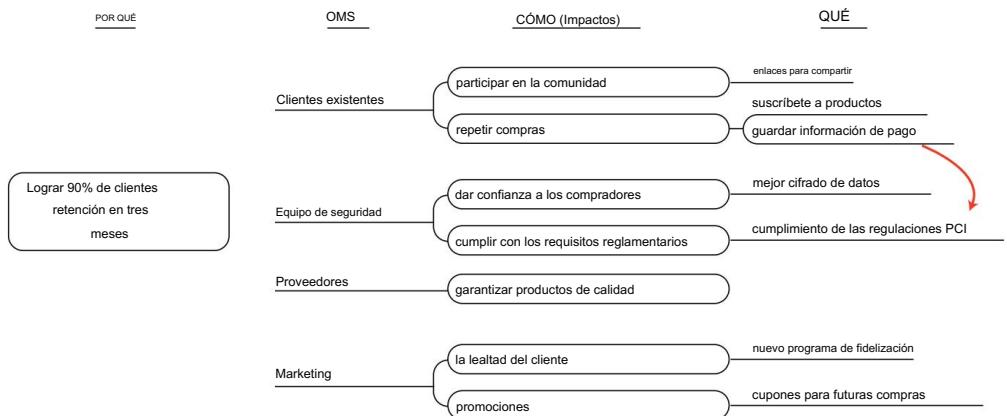


Figura 9-1 Ejemplo de mapa de impacto

los clientes pueden guardar su información de pago para facilitar futuras compras. La Figura 9-1 muestra un ejemplo de información que podría estar en nuestro mapa de impacto para esta característica.

A medida que pensamos en estas dimensiones sobre quién podría influir en nuestra capacidad para alcanzar nuestra meta, cómo podrían ayudar u obstaculizar, y qué resultados podrían llevarnos hacia la meta, identificamos experimentos potenciales que podemos probar con ciclos de retroalimentación rápida incorporados. La sesión de mapeo es una forma de concentrarse en el siguiente elemento más valioso que se debe entregar al negocio. Ayuda a identificar la forma más sencilla de resolver problemas empresariales y evitar perder tiempo en funciones que resultan no deseadas o demasiado caras.

Este es un buen momento para empezar a pensar en los riesgos de las pruebas. ¿Se puede impedir que un cliente vuelve a comprar más adelante? ¿Con quién debemos hablar para obtener datos de prueba? ¿Qué otros impactos podría haber? Nuestra experiencia en el campo puede ayudarnos a pensar en formas de lograr el objetivo sin desarrollar nuevas funciones de software. El mapeo de impacto es una forma de pensar creativamente sobre las pruebas que necesita realizar y priorizar las áreas que merecen mayor atención. Cuando pensamos en las pruebas, buscamos formas de promover la colaboración en el equipo y planificar las pruebas en un formato que sea fácil de mantener actualizado.

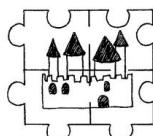
El mapa de impacto en sí podría servir como una forma de plan de prueba y, como se analiza en el Capítulo 7, “Niveles de precisión para la planificación”, los mapas mentales son excelentes herramientas de planificación. La forma física de un entregable generalmente no es importante. Lo que importa es que anticipemos nuestras necesidades de pruebas y nos aseguremos de presupuestar suficiente tiempo y otros recursos para realizar pruebas adecuadas. Los caminos alternativos que podemos generar a través del mapeo de impacto ayudan a garantizar que mitiguemos los riesgos de manera adecuada.

Mapeo de historias

El mapeo de historias es una forma práctica de modelar una característica de alto nivel y dividirla en historias de usuarios de forma bidimensional. Jeff Patton escribió por primera vez sobre esto en un artículo de Better Software de enero de 2005 , “How You Slice It” (Patton, 2005). El proceso de crear y recorrer un story map ayuda a identificar por dónde empezar con una nueva característica, visualizando el producto mínimo viable (MVP). El mapa de historia también ayuda al equipo a mantenerse enfocado en las áreas correctas, realizar un seguimiento del progreso y realizar correcciones en el rumbo a medida que avanza el desarrollo. Puede comenzar el aprendizaje con un mapa de impacto y, cuando haya identificado sus características (los resultados), puede entrar en más detalles con el mapa de la historia.

Reúna a su equipo de desarrollo y a las partes interesadas del negocio para un taller. Cree personas para representar los distintos tipos de usuarios y sus roles, y luego piense en cómo cada persona usaría su sistema o función. ¿Qué harían primero? ¿Qué harían a continuación?

Haga una línea de tiempo de las actividades de los usuarios utilizando fichas (o notas adhesivas) en una pared, mesa o piso. Luego regrese y observe la actividad de cada usuario en detalle y cree tareas de usuario y detalles sobre esas tareas. Escriba esos también en tarjetas y apílelas verticalmente debajo de la actividad del usuario correspondiente.



Una vez que el mapa de historias esté en su lugar, puede dividir las historias en historias de usuarios de alto nivel y planificar cuáles pueden incluirse en cada iteración y lanzamiento. Las actividades de mayor prioridad se encuentran en la parte superior del mapa de la historia y pueden convertirse en su MVP. Algunas personas llaman a esto un “esqueleto ambulante” (Freeman y Pryce, 2009): la cantidad mínima que puede entregar y que agregará valor para su cliente. Revise el mapa de la historia con las partes interesadas y vea si puede pensar en otros temas. Aún no entra en detalles, pero está identificando sectores de características que brindarán el mayor valor.

El proceso de mapeo de historias ayuda a los clientes y a los equipos de entrega a identificar la versión más pequeña posible, determinar qué está dentro del alcance de esa versión y priorizar las historias en consecuencia. Para obtener más información sobre el mapeo de historias, lea el libro User Story Mapping de Jeff Patton (Patton, 2014).

Consulte la bibliografía en la Parte IV, "Prueba del valor empresarial", para obtener referencias adicionales.

Mapas de historias y pruebas

Steve Rogalsky , un agilista de Winnipeg, Canadá, cuenta por qué cree que los evaluadores deberían hacerlo sobre el mapeo de historias.

Entonces eres un evaluador ágil y te preguntas por qué deberían preocuparse los mapas de historias. Es posible que ya esté convencido de que modelar su trabajo pendiente en dos dimensiones es útil para ayudar a todo el equipo a visualizar el panorama general. Sin embargo, los mapas de historias también son una valiosa herramienta de prueba, ya que brindan dos vías de prueba adicionales. En el primer caso, el mapa en sí ofrece la posibilidad de probar la validez de una solución. En el segundo, un mapa de historia mejora la capacidad de un equipo para identificar partes de la historia y luego probarlas.

Pruebas Qué a Construir

Los mapas de historias de usuario son una representación: proporcionan un medio para visualizar un sistema que podría construirse y son útiles para probar la validez de ese sistema antes de invertir mucho tiempo y dinero. Una historia compartida en un evento reciente de Agile Winnipeg demostró bien este principio. La empresa involucrada utilizó story mapping para probar una idea antes de crear cualquier software. El equipo tenía una idea de proyecto que pensaba que sería de gran utilidad para su cliente. Después de construir rápidamente un story map en torno a esa idea, presentaron el mapa al cliente en la siguiente conferencia de clientes. Aunque pronto quedó claro que la idea no acertaba, el cliente pudo colaborar con el equipo en el momento para ajustar el mapa hasta que representara lo que realmente querían construir. El mapa en sí fue la herramienta que permitió probar (y luego ajustar) la idea y hacer avanzar el proyecto.

Pruebas Solicitud Rebanadas

Como demostraron Crispin y Gregory en su primer libro, identificar porciones finas y trozos pequeños es importante para probar proyectos ágiles. Los mapas de historias ayudan a identificar esas porciones, pero, quizás lo más importante, nos ayudan a comprender cómo esas historias finamente cortadas podrían encajar entre sí para formar una porción delgada de toda la aplicación. Al emprender un proyecto ágil, los evaluadores deben realizar un cambio vital en su forma de pensar: probar solo piezas pequeñas a la vez. A pesar de este cambio fundamental, también es

Es importante garantizar que las primeras piezas encajen, lo que permitirá realizar pruebas de un extremo a otro lo antes posible. El story map ayuda a identificar y priorizar esa primera porción de la aplicación. Puede basarse en un escenario de usuario o simplemente en una serie de historias que representan las historias más pequeñas que permiten el movimiento de izquierda a derecha en el mapa.

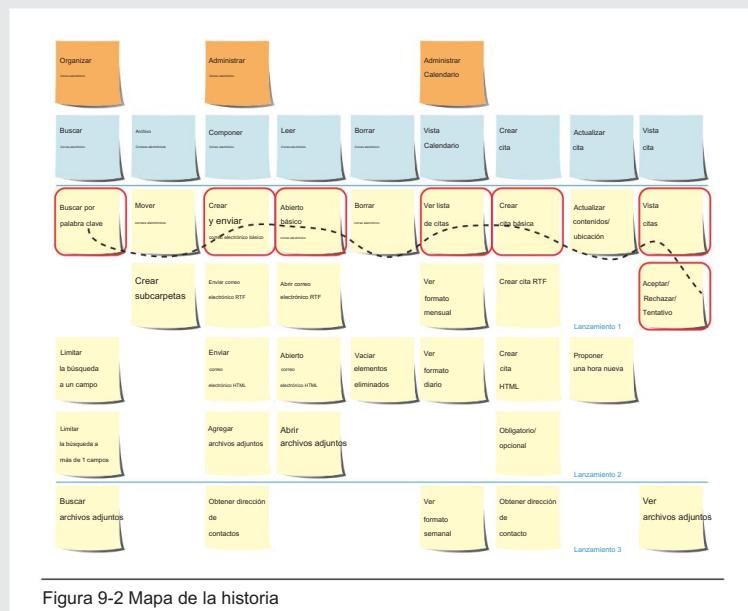


Figura 9-2 Mapa de la historia

A medida que el equipo identifica esa primera porción, es crucial utilizar excelentes habilidades de prueba. Al observar el mapa (consulte la Figura 9-2), puede identificar áreas que serán difíciles de probar, áreas donde las variaciones de las pruebas aún son relativamente desconocidas o áreas que representan un mayor riesgo. Esta actividad puede ayudar a identificar historias que deberían incluirse en el primer segmento de la solicitud.

Cuando comienzan la codificación y las pruebas, se pueden revisar las personas y los escenarios de usuario que se crearon, lo que ayuda a desarrollar el mapa y los sectores de la aplicación. Realizar pruebas con una persona en mente ayuda a garantizar que el cliente objetivo esté satisfecho con la solución. Puede que no sea posible o prudente probar si la aplicación funciona bien para todos, pero las pruebas deben evaluar si las personas objetivo pueden usar la aplicación fácilmente y si la nueva funcionalidad se adapta a sus procesos actuales o se suma a ellos sin tener que intervenir el camino.

Historia Mapas—un Pruebas herramienta después Todo

A primera vista, el story map no parece ser un activo obvio para las pruebas, pero tras una inspección más cercana, demuestra su valor en cualquier caja de herramientas de prueba. El mapa en sí es una forma confiable de probar que se está construyendo el sistema correcto antes de escribir cualquier código. El mapa también proporciona una ayuda visual para realizar pruebas en sectores de aplicaciones horizontales, lo que permite una confirmación temprana de que un proyecto va por el camino correcto.

Como señala Steve, aunque el mapeo de historias es esencialmente una herramienta de recopilación de requisitos, también es una excelente manera de comenzar a realizar pruebas antes de que se haya escrito el código. Mientras busca ideas para realizar pruebas, recuerde considerar los atributos de calidad que también necesitan ser probados. Identifique áreas de incertidumbre y riesgo, y busque alternativas más simples que puedan facilitar las pruebas.

Las 7 dimensiones del producto

Los analistas de negocios (BA) pueden enseñarnos muchas prácticas que nos ayudarán a asociarnos con los clientes para obtener la información necesaria para crear productos valiosos. Ya sea que su equipo incluya BA o no, el equipo debe aprender y experimentar con prácticas de análisis. Muchas técnicas de análisis de negocios ya están en la caja de herramientas de un evaluador y solo necesitan adaptarse ligeramente para usarlas de manera efectiva. Por ejemplo, los diagramas de estado y los diagramas de contexto se utilizan no sólo para crear variaciones de prueba, sino también para determinar qué construir. Si los evaluadores comienzan a aplicar estas técnicas durante las sesiones de preparación de la historia, habrá comenzado el cambio hacia la prevención de defectos, en lugar de concentrarse en encontrarlos después de codificar el software.

Una herramienta de análisis que hemos encontrado especialmente útil son las 7 dimensiones del producto (Gottesdiener y Gorman, 2012), que ayudan a identificar las necesidades del producto en todos los niveles de planificación. Estas dimensiones incluyen:

- **Usuario:** ¿Quién valora, se beneficia y/o utiliza el producto? El usuario podría ser una persona u otro sistema que interactúa con el producto. Las personas son una forma de describir a un típico (humano) usuario.
- **Interfaz:** ¿Qué mecanismos se necesitan para conectar a los usuarios con el producto? ¿Cómo enviará y recibirá el producto datos o

mensajes? Los mapas de relaciones, los diagramas de contexto, los prototipos y los wireframes son formas de visualizar las interfaces.

- Acción: ¿Cuáles son las capacidades del producto? como son las acciones
¿Se activa y en qué secuencia? ¿Cómo responde el producto?
¿Cómo afectan las acciones a los datos? Los diagramas de procesos, mapas de historias y mapas de flujo de valor visualizan el flujo y la secuencia de las acciones.
Piense en posibles escenarios en los que se omite un paso o se realiza en la secuencia incorrecta.
- Datos: ¿Qué datos recibe el producto, de dónde y cómo se validan y almacenan? ¿Qué datos necesitan los usuarios, en qué contexto y durante cuánto tiempo son válidos? Los modelos de datos se utilizan para visualizar las relaciones de datos; Los diagramas de estado muestran las transiciones entre estados de datos.
- Control: ¿Qué restricciones necesita el producto para hacer cumplir, por ejemplo, políticas, regulaciones, reglas comerciales? ¿Cuál es el riesgo de incumplimiento?
Las tablas y árboles de decisiones son útiles para organizar conjuntos de reglas.
- Medio ambiente: piense en las propiedades físicas del producto: ¿Dónde se utilizará? ¿Cómo se instalará, configurará, otorgará licencia y operará? ¿Qué pasa con el entorno de desarrollo? ¿Qué software, hardware y estándares se utilizarán? Una empresa puede tener múltiples entornos, así que identifique cuáles se aplican al producto que se está planificando.
- Atributo de calidad: ¿Cuáles son los niveles de servicio para cualidades operativas como disponibilidad, recuperabilidad, seguridad, usabilidad, etc.? Para las cualidades de desarrollo, piense en las definiciones de atributos como eficiencia, modificabilidad y capacidad de prueba. Piensa en cómo vas a medir estos atributos.

Nos ha resultado útil considerar las siete dimensiones durante la planificación del lanzamiento, las funciones y las iteraciones. A menudo te darás cuenta de que se ha pasado por alto una dimensión y se necesitan historias o recursos adicionales. Por ejemplo, cuando se centra en la dimensión de datos, se da cuenta de que su aplicación recibirá datos de un tercero. ¿Se conocen las reglas de validación? ¿El tercero participa en las pruebas? ¿Cuándo deberían comenzar estas pruebas? ¿Cómo se manejarán los datos no válidos? ¿Recibirá el remitente los mensajes de error adecuados? ¿Existe una forma segura para que el tercero pueda

enviar los datos y ¿se requiere un “apretón de manos” cuando se reciben? ¿Tiene datos de muestra del tercero para fines de prueba? ¿Se aplica alguna regulación y es necesario revelar alguna información a los usuarios? ¿Dónde se conservarán los datos? ¿La cantidad de datos afectará el rendimiento del sistema? Estos son ejemplos de preguntas muy poderosas que es mejor formular temprano. ¡Trae tu mentalidad de probador al juego!

Conversaciones estructuradas utilizando las 7 dimensiones del producto

María Gorman y Ellen Gottesdiener, autores de *descubrir a Entregar* (Gottesdiener y Gorman, 2012), describen una gran transformación ágil y cómo las eso fue pasando por a conversaciones estructuradas ayudaron a construir el “derecho” al poder.

La organización necesitaba mejorar rápidamente sus prácticas de prueba y análisis. Los líderes imaginaron transformar la forma en que obtuvieron, comunicaron y probaron los requisitos ágiles. Con más de 40 evaluadores y analistas, muchos de los cuales eran nuevos en las disciplinas de calidad y análisis empresarial, el desafío era grande.

Tanto el director de pruebas como el director de análisis de negocio no ocultaron el hecho de que tenían, en el mejor de los casos, prácticas ad hoc. Incluso con una capacitación ágil básica, usando kanban para visualizar el trabajo de los equipos, escribiendo historias y empleando una forma aproximada de dado_cuándo_entonces para especificar los requisitos, los equipos todavía actuaban en modo de traspaso. Esto significaba que no estaban teniendo conversaciones oportunas y justo a tiempo sobre los requisitos ni en el nivel de iteración (Now-View) ni en el de lanzamiento (Pre-View). Los equipos estaban encontrando tarde defectos que se habrían evitado con un buen análisis. Estaban algo desmoralizados porque luchaban por entregar historias “terminadas” dentro de una iteración.

El equipo de liderazgo necesitaba que los evaluadores y analistas desempeñaran un papel de liderazgo al ejemplificar a las comunidades de desarrolladores y empresas cómo colaborar eficazmente en torno a los requisitos ágiles.

Mary Gorman guió a todos los evaluadores y analistas de negocios en el aprendizaje y aplicación del marco Discover to Deliver. Un proyecto crítico fue el foco de una clínica que involucró a todo el equipo ágil en la identificación de características e historias de alto valor para la próxima versión e iteración. El equipo descubrió en colaboración opciones para las 7 dimensiones del producto (consulte la Figura 9-3).

Interfaz de usuario	Acción	Datos	Control medioambiental			Calidad Atributo
Usuarios interactuar con el producto	El producto conecta a los usuarios, sistemas, y dispositivos	El producto proporciona capacidades para usuarios	El producto incluye un repositorio de datos y útil información	El producto hace cumplir restricciones	El producto se ajusta al físico propiedades y tecnología plataformas	El producto tiene cierta propiedades que califican su operación y desarrollo

Figura 9-3 Las 7 dimensiones del producto
 Fuente: Ellen Gottesdiener y Mary Gorman. 2012. Descubrir a Entregar Análisis y planificación ágil de productos. Sudbury, MA: EBG Consulting. Usado con permiso.

El equipo utilizó el tablero de opciones para mostrar y analizar sus hallazgos. (El Tablero de Opciones cuelga de una pared larga con los 7 símbolos de Dimensión del Producto colocados en la parte superior. Debajo de cada dimensión, el equipo enumera las opciones y esboza los modelos de análisis apropiados). Emplearon la conversación estructurada en tres partes para guiar su trabajo:

- Explorar opciones.
- Evaluar opciones dentro y entre las 7 dimensiones e identificar soluciones candidatas de alto valor.
- Confirmar la solución definiendo criterios de aceptación. Hubo actividades para todo el grupo, así como trabajo en subgrupos multifuncionales.

Al final de la sesión, la gente reflexionó y señaló: "Llegamos a un entendimiento compartido de las historias; todo el equipo participa activamente y contribuye; nuestra conversación tenía un enfoque; Pudimos desarrollar muchos detalles realmente buenos". Una semana después, el propietario del producto y el líder tecnológico compartieron sus ideas.

Propietario del producto: "En el último día y medio siento que nuestro equipo ha progresado mejor o más que toda la semana pasada (iteración 1). Como resultado, nuestro equipo no se siente frustrado por la falta de progreso o comprensión, sino que se siente alentado por el progreso que se está logrando, lo que a su vez conduce a un ambiente de trabajo más positivo".

Líder tecnológico: "Solo quería transmitirles algunos comentarios positivos sobre las sesiones de descubrimiento de nuestro equipo esta semana. Creo que todos sintieron que hemos tenido más éxito esta semana. Hice una encuesta rápida sobre por qué la gente pensaba que era así. Algunos comentarios:

- "No estamos tratando de hacerlo perfecto".
- "No estamos tan concentrados en simplemente completar la tarjeta de la historia; más bien el La tarjeta de historia es un subproducto de la discusión.
- "Estamos más centrados en comprender los requisitos, no en intentar adaptar la historia a una solución esperada".
- "El enfoque de las 7 dimensiones del producto está ayudando a centrar el debate".

Mary y Ellen recientemente hicieron un seguimiento con los líderes de pruebas y análisis de negocios. Ellos compartieron:

- "Los equipos utilizan las 7 dimensiones del producto en todas sus conversaciones sobre requisitos. Cuelga las imágenes de las 7 dimensiones cerca de sus tableros kanban y las utiliza para guiar el descubrimiento y el análisis. Los resultados son conversaciones más eficientes y efectivas y requisitos de mayor calidad".
- "La clave fue hacer de estas conversaciones sobre requisitos una colaboración entre todas las partes interesadas. Dado que el análisis y la mentalidad del evaluador son responsabilidad de todos, esto ha tenido un gran impacto en las comunidades de desarrollo y propietarios de negocios".
- "Utilizan una lista de lo que llaman 'preguntas de sondeo' para cada una de las 7 Dimensiones. Las preguntas combinan las 'Preguntas de enfoque para la conversación estructurada' proporcionadas en Ágil Planificación de productos y Análisis con pruebas heurísticas".
- "Los equipos de proyecto están cada vez más cerca de realizar realmente análisis justo a tiempo y lo suficiente".

Los equipos de pruebas y análisis de negocios actúan como líderes en el uso de las 7 dimensiones del producto para la planificación de pruebas, especificaciones y una colaboración más profunda. Los éxitos logrados hasta la fecha en la realización de conversaciones estructuradas sobre requisitos oportunas, enfocadas y holísticas han sido fundamentales para impulsar el proyecto de transformación.

Experimente con prácticas de expertos en análisis de negocios ágiles, como las 7 dimensiones del producto y conversaciones estructuradas, para ayudar a su equipo a identificar qué funciones desarrollar. Recuerde hacer preguntas y apuntar a esa comprensión compartida de lo que su equipo está construyendo.

Más herramientas o técnicas para explorar temprano

Eric Ries (Ries, 2010) habla de la mentalidad que necesitan las empresas que están innovando y probando nuevas ideas. Las ideas que recomienda para las startups lean se pueden utilizar en muchas situaciones. Construir-medir-aprender (BML) se trata de construir pequeños fragmentos, medir y aprender obteniendo comentarios tempranos y validando ideas. Suena a prueba, ¿no?

Los evaluadores pueden agregar valor en estas situaciones haciendo preguntas y ayudando a determinar qué se puede medir. A veces podemos hacer cosas sencillas como revisar maquetas en papel que han creado programadores o diseñadores de UX.

Las pruebas A/B son una técnica que se puede utilizar para probar una hipótesis. La idea es desarrollar dos implementaciones separadas y ponerlas a disposición de los clientes para que las utilicen activamente y luego medir los resultados. Los usuarios son dirigidos a diferentes implementaciones. La empresa supervisa estadísticas como, por ejemplo, qué clientes hacen clic o compran algo y cuáles se van inmediatamente. De esta manera, las empresas pueden decidir qué experiencia les gusta más a los clientes basándose en evidencia real. Uno de nuestros colaboradores ha compartido

su historia sobre las pruebas A/B en el Capítulo 13, "Otros tipos de pruebas".

Como mencionamos en el Capítulo 7, "Niveles de precisión para la planificación", visualizar diferentes opciones con herramientas como los mapas mentales es una manera poderosa de descubrir suposiciones ocultas.

Invierta para construir lo correcto

El futuro es impredecible; sólo podemos hacer nuestra suposición más fundamentada. Sin embargo, al hacer buenas preguntas y visualizar posibles respuestas, podemos ayudar a nuestros clientes a identificar las funciones más valiosas para desarrollar a continuación. Empiece por conocer el propósito de cada característica propuesta. Piense en lo que sucedería cuando la función se implemente en producción. ¿Cómo sabremos que es exitoso? ¿Cómo juzgaremos si

proporcionó el valor correcto? ¿Cómo mediremos o monitorearemos? Responder esas preguntas temprano puede ayudar a evitar perder tiempo en cosas equivocadas.

Hay muchos aspectos de la calidad y nos damos cuenta de que técnicas como las 7 dimensiones del producto apenas arañan la superficie. Tenemos que equilibrar la funcionalidad con la confiabilidad, la seguridad, el rendimiento y otros atributos valorados por nuestros clientes. Tenemos que sopesar el costo de retrasar un lanzamiento mientras trabajamos para crear una función que sea lo suficientemente buena frente a la necesidad de deleitar a nuestros clientes y hacer que quieran comprar nuestro producto.

Invierta tiempo para ayudar a los clientes a identificar la característica más valiosa para desarrollar a continuación e implementar el producto mínimo viable. Si puede evitar perder el tiempo entregando algo incorrecto, su equipo tendrá más tiempo para actividades de prueba críticas.

En el próximo capítulo, entraremos en más detalles sobre las sinergias entre las pruebas y otros conjuntos de habilidades que nos ayudan a comenzar el ciclo de cada nuevo producto enfocándonos en construir lo correcto.

Resumen

Es difícil para los clientes saber lo que quieren y aún más difícil explicárselo al equipo de entrega. Podemos ayudarlos a identificar las características más valiosas para construir y ayudarlos a articular qué características quieren para que tengamos una comprensión compartida de qué construir.

En este capítulo, analizamos los siguientes conceptos:

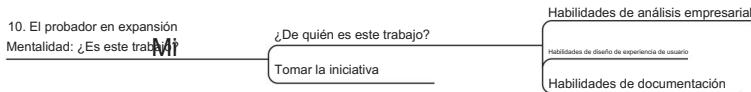
- Comience con el “por qué”, el propósito del software que estamos desarrollando.
- Pruebe algunas herramientas diferentes para ayudar a los clientes a expresar lo que quieren, incluidas
 - Mapeo de impacto
 - Mapeo de historias
 - 7 Dimensiones del producto
 - Técnicas de puesta en marcha ajustada
- Experimente con diferentes herramientas que ayuden a sus equipos de desarrollo y de clientes a colaborar para crear lo correcto.
- El proceso necesita ciclos de modelado, hipótesis, experimentos, descubrimiento y aprendizaje, justo lo que proporciona una mentalidad de prueba.

Esta página se dejó en blanco intencionalmente.

Capítulo 10

El probador en expansión

Mentalidad: ¿Es este mi trabajo?



En el capítulo anterior, mencionamos la superposición entre el análisis empresarial y las pruebas. Tanto los analistas de negocios como los evaluadores trabajan con los clientes para comprender sus deseos para cada característica, pero también intervienen otras funciones. Por ejemplo, los expertos en diseño de experiencia de usuario (UX) buscan comprender cómo las personas usarán el producto y el valor que la empresa espera obtener de las nuevas capacidades propuestas. Los redactores técnicos deben comprender cómo los diferentes usuarios utilizarán el sistema para poder documentar cómo satisface sus necesidades comerciales o técnicas. Sin embargo, un equipo u organización puede carecer de esas habilidades especializadas, por lo que cuando surge la necesidad, los miembros del equipo pueden aprender y utilizar algunas prácticas y técnicas valiosas de esas otras especialidades.

¿ De quién es este trabajo ?

Como mencionamos en el Capítulo 3, “Roles y competencias”, el desarrollo ágil fomenta la generalización de especialistas con habilidades en forma de T. Sin embargo, a menudo necesitamos miembros del equipo con formación especializada y experiencia en determinadas áreas para comprender las necesidades del negocio.

Habilidades de análisis empresarial

Desde que escribimos Agile Testing, nos hemos dado cuenta cada vez más de que los BA aportan algunas habilidades especializadas importantes al equipo ágil. Facilitan conversaciones estructuradas con los expertos en negocios. Recuerdan hacer preguntas sobre todas las dimensiones del producto. ellos entienden el todo

dominio empresarial, no sólo las partes que están automatizadas con software. Históricamente, los analistas de negocios ayudan a los expertos en negocios a comunicar sus necesidades a los equipos de entrega. Bernice Niel Ruhland nos contó sus experiencias en la gestión tanto de evaluadores como de analistas de negocios. A menudo las mismas personas desempeñaban ambos roles. Tener ambos conjuntos de habilidades en el equipo funcionó bien porque siempre estaban pensando juntos en las pruebas y los requisitos.



Al igual que los expertos en pruebas, los expertos de BA saben cómo hacer buenas preguntas y comenzar con los "por qué". Muchos BA saben cómo especificar pruebas basadas en ejemplos de clientes y probablemente tengan algunas herramientas en su conjunto de herramientas que están más allá de la experiencia de la mayoría de los evaluadores. Primero debemos pensar en el problema del negocio y, desde una perspectiva de prueba, es posible que nos centremos en la solución demasiado pronto. Eso significa que, como evaluadores, es posible que debamos pensar de manera diferente al probar las habilidades de BA.

Poniendo en el NO Tiene

Mike habla, de ~~equilibrio~~ de Nueva Zelanda, cuando se centró en la solución y el diseño de problemas de evaluación en su propia práctica con los clientes, cuando intentó escribir por primera vez

Algo que realmente anima a la gente a hacer es apoyar el ideal neozelandés de "probar" algo fuera de su zona de confort dentro de su equipo. Esto se remonta a los primeros días de Nueva Zelanda como colonia pionera, donde los recursos y las habilidades eran escasos y, en general, se fomentaba una cultura de especialistas generalizadores. ¿Suena familiar?

Un proyecto en el que estaba necesitaba un importante análisis empresarial inicial y había probado suficientes requisitos empresariales en mi vida para pensar que sería fácil. Fui por ahí, tuve mis consultas con la empresa y escribí todo. Desafortunadamente, lo que documenté no describía muy bien los problemas y era demasiado detallado.

Probablemente aprendí más en un mes haciendo el trabajo de Licenciatura que en años de estar al otro lado de los requisitos comerciales. También me dio mucha más empatía con las personas en el rol de BA.

Devolviéndome el favor, nuestro analista de negocio del proyecto me ayudó con las pruebas y, como yo, también recorrió el camino torcido antes de salir bien. Esa visión y empatía nos ayudaron a trabajar más estrechamente y a comprender cómo nuestros trabajos se integraban en el trabajo de los demás. En mi experiencia de probar el rol de BA, lo más difícil fue aprender a lograr el nivel correcto de detalle requerido. Concéntrese en las necesidades y trate de no forzar una solución de diseño; de lo contrario, estará limitando la capacidad de su equipo para satisfacer esas necesidades.

Si su equipo tiene evaluadores pero no BA, o viceversa, busque formas de adquirir las habilidades faltantes que podrían ayudar a su equipo a aprender mejor qué construir. El equipo de Lisa, al no poder contratar un BA, formó una comunidad de práctica de BA, donde los miembros del equipo se reunían para compartir las habilidades de BA aprendidas en libros, artículos y sesiones de conferencias. Aprendieron mejores formas de hablar con las partes interesadas del negocio y ampliaron su perspectiva para incluir más dimensiones de calidad. Para conocer la historia completa, consulte el Capítulo 5, “Conciencia técnica”.

Consideraciones sobre los requisitos y el propósito de las pruebas

Pete Walen comparte sus pensamientos sobre el entrelazamiento de la definición de requisitos y las pruebas.

Es bueno saber cuáles son los requisitos, qué pruebas existen, qué pruebas están configuradas y cuáles se han ejecutado. Sin embargo, poco de eso importa si no conocemos el problema que el proyecto pretende abordar. Es fundamental conocer el propósito comercial que el sistema pretende cumplir.

Cuando uno se sienta y habla con gente de negocios en una conversación, puede aprender cosas que de otro modo no se discutirían. Cuando tu actitud es “Por favor, ayúdame a entender para poder ayudarte mejor” (como sirviente, no como superior), las paredes se derrumban y, a menudo, sucede algo extraño: las personas tienden a compartir información libremente.

Cuando los programadores, diseñadores o evaluadores se presentan como expertos que saben cómo solucionar los problemas que tienen los clientes comerciales, puede parecer condescendiente. Cuando abordamos el problema como colegas, podemos encontrar formas de fundamentar nuestras decisiones.

Qué pasa ¿Analistas de negocios?

Un buen analista de negocios puede obtener información y transmitirla al equipo. Sin embargo, con cada capa introducida en el flujo de información, la información transmitida se alterará. A veces se pierde información; en otras ocasiones se puede agregar material. La “aclaración” puede cambiar críticamente parte del mensaje. Encuentro que cuando me acerco a las personas que hacen el trabajo, a menudo aprendo cosas que son importantes para mí como evaluador y que otras personas ignoran como sin importancia.

Si puedo comprender mejor las necesidades del cliente, puedo utilizar el software de manera más eficiente. Si las pruebas que escribo y ejecuto no cumplen explícitamente los requisitos, ¿son realmente necesarias? ¿Son reales o mi interpretación de algo que alguien dijo?

Por supuesto, las ideas importantes se identificarán en los requisitos documentados. Puede haber otras cosas como "No corromper la base de datos" o "Esto debe reflejarse en el sistema ziggidy-splat" o "Este valor debe coincidir con el que aparece en esta otra pantalla". ¿Estas cosas forman parte de la lista de expectativas de los usuarios? ¿Las necesidades comerciales se reflejan completamente en los requisitos documentados? Sólo en estas conversaciones encuentro la respuesta.

Mi consejo: ejercita tanto los requisitos como las historias. Si no puede ejercitarse ambos, practique las historias y luego pida a la gente del negocio que analice los resultados con usted.

Habilidades de diseño de experiencia de usuario

Empresas como Apple han demostrado la importancia del diseño de productos. Como ha señalado Andy Budd (Traynor, 2011), las "atracciones encantadoras" como deslizar el dedo para desbloquear el iPhone "hacen que la gente se enamore de un producto". Los diseñadores de UX son miembros integrales y clave de los equipos de software ágiles de hoy.

Los evaluadores y diseñadores de UX colaboran productivamente desde las primeras etapas del diseño de funciones. Los evaluadores, especialmente aquellos que tienen contacto directo con los usuarios finales o que usan el producto ellos mismos, brindan comentarios útiles sobre las maquetas de la interfaz de usuario, el flujo de trabajo y otros aspectos del diseño. Los buenos diseñadores de UX saben cómo obtener valor de las pruebas de usabilidad y están felices de asociarse con evaluadores para realizar esas actividades. Algunas actividades centradas en la usabilidad que son comunes tanto para los diseñadores como para los evaluadores de UX son las pruebas A/B, los grupos focales e incluso la creación de prototipos en papel.

Las técnicas de planificación, como el mapeo de impacto, pueden ayudarlo a comprender mejor las necesidades y deseos del negocio. No sabemos cómo era el iPhone diseñado, pero podemos imaginar un ejemplo de mapa de impacto para ello:

- Por qué (objetivo): obtener una participación de mercado del 70%
- Quién (Personas): usuarios de iPhone actuales y potenciales
- Cómo (Comportamientos): Desbloqueo de teléfono sencillo que enamora a los usuarios
- Qué (Actividad/Función): Diseñar un gesto de deslizar para desbloquear

El equipo (incluidos los evaluadores, por supuesto) podría luego intercambiar ideas sobre otros comportamientos y actividades que podrían cumplir el objetivo.

La historia de Lisa

Nuestro equipo tenía un diseñador de UX que trabajaba de forma remota y estaba demasiado disperso en nuestro equipo de alrededor de 20 desarrolladores. Cuando contratamos a un segundo diseñador de UX en nuestra oficina de Denver, se abrieron nuevas oportunidades para nosotros, los evaluadores.

Otro evaluador de mi equipo tuvo la idea de organizar una reunión de intercambio de ideas con los tres evaluadores y ambos diseñadores. El tema era "visibilidad del diseño antes de la producción".

Hemos estado probando algunas de las ideas de acción resultantes, que incluyeron:

- Repasen juntos las características de diseño de las historias. Tenga en cuenta los cambios posteriores (por ejemplo, debido a limitaciones de implementación) en los comentarios de la historia.
- Los diseñadores colaboran con los evaluadores para aceptar historias que incluyan implementaciones de diseño de funciones.
- Los diseñadores involucran más a los evaluadores en las pruebas de interacción y diseño de flujo.

Estos experimentos han ayudado a reducir el tiempo perdido en reelaborar el diseño de la interfaz de usuario (UI) después de comenzar la codificación. Las pruebas de aceptación son más fluidas cuando nos emparejamos, ya que los diseñadores de UX pueden detectar de inmediato problemas de implementación, como una fuente o color incorrectos. Incluso cuando no nos emparejamos, los diseñadores están disponibles para responder preguntas de inmediato, por lo que nunca nos bloquean.

Los evaluadores ayudamos a los diseñadores a perfeccionar su script de prueba de usabilidad. Gracias a la experiencia de nuestros diseñadores y a la experiencia en pruebas de usabilidad, recibimos comentarios útiles de los usuarios de nuestro producto en las primeras etapas del proceso de desarrollo. Probar nuestro nuevo diseño de interfaz de usuario fue más fácil con la ayuda de los diseñadores.

Una vez que nuestro nuevo diseño de interfaz de usuario se lanzó en versión beta, los comentarios de los clientes fueron abrumadoramente positivos.

Incluso si no cuenta con especialistas en aspectos como diseño de UX y pruebas de usabilidad en su equipo, puede aprender lo suficiente para obtener comentarios valiosos de los usuarios. Consulte el Capítulo 13, "Otros tipos de pruebas", para ver ejemplos de pruebas de usabilidad.

Habilidades de documentación

Muchos equipos todavía necesitan documentación de usuario, ya sea electrónica o en papel. Si forma parte de uno de los equipos afortunados que pueden trabajar en estrecha colaboración con redactores técnicos, probablemente ya sepa lo valiosos que pueden ser. En Agile Testing (págs. 208-209), dimos algunos ejemplos basados en nuestras experiencias. Los redactores técnicos se encuentran en una posición única para desafiar ideas; Cuando tienes que articular un concepto, a veces no se traduce tan bien como crees.

Cuando no tenga un redactor técnico en su equipo pero aún sea responsable de la documentación del usuario, considere agregar la tarea "Crear contenido" a cada historia. Esta tarea puede ser completada por cualquier miembro del equipo. A nivel de función, puede haber una tarea para "Completar la documentación del usuario". Para entonces, la interfaz de usuario es estable, por lo que si necesita tomar capturas de pantalla, puede hacerlo. Luego, cualquier miembro del equipo o quizás el redactor técnico de la empresa puede recopilar y editar el contenido.

Otra idea es emparejarte con otros miembros de tu equipo. Por ejemplo, Lisa se asoció, no con un redactor técnico, sino con los diseñadores de UX de su equipo para probar una extensa documentación en línea para su API.

Tomar la iniciativa



¡La alegría de tener roles borrosos en equipos ágiles es que hace que nuestros trabajos sean más interesantes! Si siente que su equipo y sus clientes están luchando por definir una característica, póngase otro sombrero. ¿Qué prácticas de una profesión diferente puedes probar?

La historia de Lisa

Trabajé durante muchos años en un equipo que producía software de servicios financieros. Nuestro software gestionó todos los aspectos de los planes de jubilación 401(k). El sistema se diseñó originalmente para que solo una persona de cada empleador con un plan 401(k) pudiera iniciar sesión y administrar el plan de ese empleador. Este "patrocinador del plan" realizó actividades tales como presentar contribuciones de nómina, inscribir nuevos empleados en el plan y aprobar distribuciones y préstamos de cuentas 401(k).

El término "patrocinador del plan" se utilizó en todo nuestro sitio web. En un momento, iniciamos una función para permitir que varias cuentas de usuario realicen actividades de "patrocinador del plan". Sin embargo, por ley, el patrocinador real del plan es el empleador. Tuvimos varias discusiones en grupos pequeños sobre terminología. Todos tenían una opinión diferente.

Finalmente programé una reunión con todas las partes interesadas para tomar una decisión final sobre la terminología. ¿Reunir a la gente de esta manera es trabajo de probador? ¿Por qué no? Nuestro ScrumMaster estaba enfermo y nuestro propietario de producto

siempre estuvo inundado. El desarrollo de la función había comenzado y estábamos perdiendo el tiempo con indecisiones sobre la palabrería.

En 15 minutos, los distintos gerentes comerciales acordaron el término "administrador del plan" para todos los usuarios de un empleador determinado que iniciarían sesión para realizar lo que solía denominarse actividades de "patrocinador del plan". Seguiría habiendo un solo patrocinador de plan por plan cuyo nombre se utilizaría en todos los documentos legales.

Como el propietario del producto estaba demasiado ocupado para hacerlo, hice un simulacro de todos los cambios para las muchas páginas del sitio que, hasta ahora, decían "Patrocinador del plan" (consulte la Figura 10-1). Los publiqué en la wiki para que nuestro programador remoto (que trabaja en India) pudiera verlos y agregué una tarjeta de tarea a la fila de historia relevante en nuestro tablero. A la mañana siguiente, nuestro programador remoto había completado la mayoría de los cambios y otro programador completó el resto. Verifiqué los cambios, se los mostré a los clientes y actualicé una prueba de regresión automatizada.



Figura 10-1 Maqueta simple

Sí, hubiera sido preferible tener la terminología elaborada antes de comenzar la historia. Después de esta y otras experiencias similares, comenzamos a retroceder en el negocio para tener todas las reglas comerciales y ejemplos listos al inicio de la iteración. Si nos faltaba la información necesaria, posponíamos la historia para la siguiente iteración. Sabemos que las preguntas y los cambios durante el desarrollo son inevitables. Pero siempre que podamos encontrar formas de avanzar, ahorrar tiempo y mejorar la calidad, ¡quien tenga la iniciativa puede liderar el camino!

Muchas de las habilidades que practican los analistas de negocios y los diseñadores de UX son en realidad lo que llamamos "pruebas de valor comercial". Desafiar ideas y probar esas ideas y suposiciones nos acerca a las características que nuestros clientes valoran. En el próximo capítulo, hablaremos sobre cómo obtener ejemplos para expresar un entendimiento compartido. Se trata de desarrollar lo "correcto".

Resumen

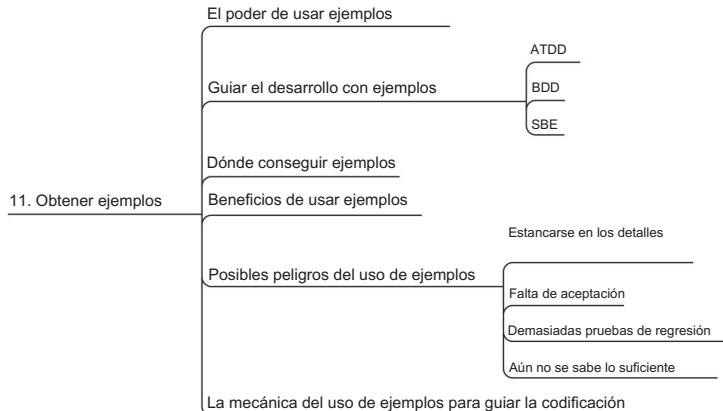
Desdibujar las distinciones entre roles puede hacer que nuestro trabajo sea mucho más interesante. Este capítulo trataba sobre ampliar su forma de pensar y aprender de los demás.

- Haga una lluvia de ideas con el equipo para identificar qué habilidades pueden faltar.
- Pruebe experimentos creativos; aumente sus habilidades en forma de T para llenar los vacíos de habilidades. Use un "sombrero" especial diferente cuando sea necesario.
- Las habilidades de los analistas de negocios son sinérgicas con las de los evaluadores. Si su equipo tiene evaluadores y BA, pueden trabajar juntos para obtener una mejor calidad del software.
- Los evaluadores y diseñadores de UX pueden asociarse para actividades como pruebas de usabilidad, creación de prototipos y obtención rápida de comentarios de los usuarios. Esto ayuda al equipo a ahorrar tiempo y mantener el rumbo.
- Todos los miembros del equipo pueden ponerse el sombrero de redactor técnico si es necesario y crear contenido, pensar en cómo un concepto podría traducirse en palabras y tal vez incluso editar y probar la documentación técnica y de usuario.

- Aprenda prácticas de otras profesiones, como análisis de negocios y diseño de UX, para ayudar a incorporar múltiples dimensiones de calidad en las funciones del software.
- Cuando vea un problema, tome la iniciativa de reunir a las personas adecuadas y discutir posibles soluciones.

Capítulo 11

Obtener ejemplos



Según nuestra experiencia, obtener ejemplos de los clientes es una forma poderosa de guiar el desarrollo tanto con la tecnología como con las pruebas comerciales.

Los ejemplos forman el corazón de los cuadrantes 1 y 2, ya sea un prototipo en papel, un diagrama de flujo dibujado en la pizarra o una hoja de cálculo con entradas y salidas esperadas. En Agile Testing, hablamos de ejemplos en casi todos los capítulos, y eso también es cierto en este libro.

Una de nuestras preguntas favoritas que hacemos si no estamos seguros de lo que se está discutiendo es: "¿Puedes darme un ejemplo de eso?" Le sugerimos que haga lo mismo.

El poder de usar ejemplos

La primera vez que aprendimos sobre el desarrollo basado en ejemplos fue de Brian Marick en 2003. Como explicamos en Agile Testing (págs. 378–80), hemos dependido del trabajo con los clientes para obtener ejemplos de comportamientos deseados y no deseados del sistema para que sabemos las cosas correctas para construir nuestros clientes.

Ejemplos en la vida real

La siguiente historia muestra cómo usar Sherry Heinze, de la vida cotidiana. Desde Canadá, prueba ejemplos en nuestro

Estaba trabajando en el sitio de un cliente que cerró durante cuatro días completos y dos días parciales durante las vacaciones. La gerencia envió el calendario de vacaciones en formato de tabla. Un gerente que tenía tanto personal como contratistas envió una copia del cronograma con una nota para ayudar a las personas a reservar su tiempo libre correctamente.

Nota: El horario habitual de la empresa era de 7,5 horas diarias, de lunes a jueves, y de 7 horas los viernes.

Al ingresar el tiempo de los días festivos, consulte los cuadros [consulte la Figura 11-1] a continuación.

Para cierres de oficinas (que no sean Stat), reservetiempos con este código de administrador; de lo contrario, el tiempo será suyo según los códigos ingresó comisiones

si eres un contratista, solo ingrese el tiempo en la oficina y no el código de administrador anterior. En los días y horas en que la oficina esté cerrada, tiempo ingresado para los contratistas debe ser 0.

Fecha	Si está de vacaciones			Si no está de vacaciones		
	Vacaciones	estadística Día festivo	Administración	Regular Facturación	estadística Día festivo	Administración
Lunes 23 de diciembre (Horario de oficina habitual)	7.5	0	0	7.5	0	0
Martes 24 de diciembre (Cerrado a las 2:00)	4.5	0	3	4.5	0	3
miércoles 25 de diciembre (Cerrado - Estadísticas)	0	7.5	0	0	7.5	0
jueves 26 de diciembre (Cerrado - Estadísticas)	0	7.5	0	0	7.5	0
viernes 27 de diciembre (Cerrado)	0	0	7	0	0	7

Figura 11-1 Gráfico de tiempo

Miré este cuadro y luego hice algunas preguntas aclaratorias ya que empiezo a trabajar a las 7:00 am y soy contratista.

- ¿Significa esto que no puedo facturar las 6 horas que normalmente tendría el 24 de diciembre?
- Si no, ¿debería salir a las 11:30 para mantener las 4,5 horas?
- ¿Quieres que todos lleguemos a las 9:00 el 24 de diciembre?

- ¿Significa esto que no puedo trabajar en casa ese día, como suelo hacer si
¿Las escuelas están cerradas?
- ¿Qué pasa si soy un empleado que trabaja en un grupo de apoyo y debo comenzar
antes a contestar los teléfonos?

Lo que descubrí fue que la persona que creó este gráfico supuso que todos los demás comenzaban a trabajar a las 9:00 a. m. y trabajaban sólo en la oficina.



La historia de Sherry nos muestra cómo un ejemplo tan simple como un gráfico puede brindarnos un punto de partida para hacer preguntas concretas, trabajando hacia una comprensión compartida del objetivo de una característica.

Las herramientas de prueba funcionales han madurado en los últimos años para ayudarnos a capturar ejemplos y convertirlos en pruebas automatizadas. Esto es bueno, pero es posible automatizar muchas pruebas y aun así no lograr entregar lo que el cliente realmente desea.

En su discurso de apertura de Agile Testing Days 2013, JB Rainsberger señaló que muchos equipos todavía no aprovechan la simple idea de "hablar con ejemplos" (Rainsberger, 2013). Su mensaje se hace eco del de Liz Keogh (Keogh, 2012a):

tener conversaciones
es más importante que capturar conversaciones
es más importante que automatizar conversaciones

Del mismo modo, muchos equipos utilizan un lenguaje específico de dominio (DSL) legible por la empresa, a menudo en un formato dado _ cuando _ entonces, para describir el comportamiento de las funciones sin entrar en detalles de implementación. Puede utilizar esta técnica en scripts de prueba automatizados, pero puede tener más valor como herramienta analítica. Colaborar con los clientes para crear ejemplos de estilo "dado _ cuándo _ entonces" ayuda a definir historias útiles y de tamaño adecuado que, cuando estén terminadas, proporcionarán la característica deseada. La automatización se convierte en un efecto secundario útil del uso de la herramienta.

Si su equipo aún no está trabajando con las partes interesadas para obtener ejemplos de comportamiento del sistema deseado y no deseado, comience a experimentar con

Este enfoque. Vea si le ayuda a comprender más rápidamente lo que quieren sus clientes y a entregar lo correcto con menos rotación y pérdida de tiempo. Janet ha descubierto que esta es una regla general útil: si tiene más de un ejemplo de comportamiento deseado, es posible que necesite más de una historia.

Decidimos centrar todo un capítulo de este libro en obtener ejemplos porque es una práctica clave y todavía no se ha popularizado en muchos equipos. Desafortunadamente, el nombre "desarrollo basado en ejemplos" tampoco tuvo éxito. Los nombres comunes que se utilizan actualmente para describir el proceso son

- Desarrollo basado en pruebas de aceptación (ATDD)
- Desarrollo impulsado por el comportamiento (BDD)
- Especificación por ejemplo (SBE)

En Agile Testing (págs. 99-101), nos referimos a estas pruebas como pruebas orientadas al negocio o al cliente que impulsan el desarrollo de manera similar al desarrollo basado en pruebas (TDD), pero a nivel empresarial. Ayudan a definir la calidad externa al definir las características que desean los clientes.

En este libro nos referimos genéricamente a esta práctica como "guiar el desarrollo con ejemplos". No dudes en cambiar el término por el que utilices en tu equipo. Intentamos explicar las diferencias aquí, pero en realidad se trata de comenzar con ejemplos y conversaciones.

La elección de la herramienta puede determinar qué práctica adoptar, por lo que le recomendamos que sepa lo que quiere antes de elegir una herramienta. Por ejemplo, en aplicaciones de tipo flujo de trabajo, puede ser aplicable describir el comportamiento. Si la aplicación requiere muchos cálculos, una hoja de cálculo o un formato de prueba tabular con ejemplos específicos podría ser más apropiado. Damos ejemplos de ambos formatos en el Capítulo 16, "Patrones y enfoques de diseño de automatización de pruebas".

Guia el desarrollo con ejemplos

No se distraiga con la jerga o las palabras de moda. Ya sea que elija llamar a su proceso ATDD, BDD o SBE, el objetivo es el mismo: un entendimiento común para construir lo correcto la primera vez. Todos ellos siguen un flujo similar al mostrado en la Figura 11-2 con ligeras variaciones en el

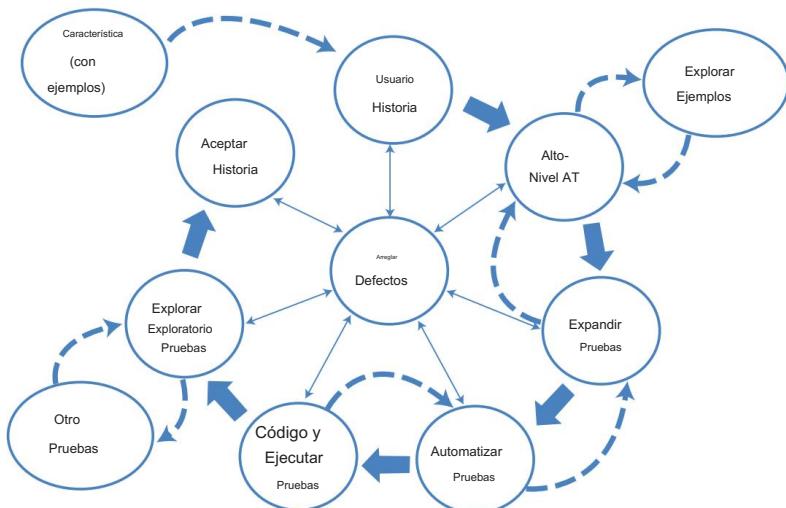


Figura 11-2 Desarrollo de guías con ejemplos

nombrar. Lo más importante es recordar empezar con el final en mente.

ATDD

Jennitta Andrea ha descrito a ATDD como (Andrea, 2010)

... la práctica de expresar los requisitos funcionales de la historia como ejemplos concretos o expectativas antes del desarrollo de la historia. Durante el desarrollo de la historia, se produce un flujo de trabajo colaborativo en el que: se escriben ejemplos y luego se automatizan; se desarrollan pruebas unitarias automatizadas granulares; y el código del sistema se escribe y se integra con el resto del software probado y en ejecución. La historia está "terminada" (se considera lista para pruebas exploratorias y de otro tipo) cuando pasan estas comprobaciones automatizadas que definen el alcance.

Un problema con la etiqueta “desarrollo impulsado por pruebas de aceptación” es que “prueba de aceptación” es un término vago que significa cosas diferentes para diferentes personas. Se puede confundir con las pruebas de aceptación del usuario (UAT),

donde un usuario final o proveedor externo “acepta” el producto. Esta aceptación podrá estar ligada a pagos contractuales.



Definimos las pruebas de aceptación como las pruebas que describen la intención comercial que debe cumplir cada historia. Dependiendo de cómo su equipo implemente la práctica, las pruebas de aceptación pueden incluir solo el comportamiento esperado de alto nivel y un par de ejemplos de mal comportamiento. Sin embargo, también pueden incluir una amplia gama de pruebas que abarcan todo excepto TDD a nivel de unidad y componente. Pueden incluir atributos de calidad como usabilidad y rendimiento, aunque preferimos pensar en ellos como limitaciones que se aplican a todas las historias. La mayoría de la gente se refiere a pruebas funcionales cuando habla de ATDD.

Entregando valor a través de conversaciones en Paddy Power

Augusto Evangelista , a profesional en desarrollo de software en Irlanda, comparte sus experiencias con conversaciones en su organización.

Me uní a Paddy Power como ingeniero de pruebas principal en 2012 y mi tarea estuvo clara desde el primer día que conocí a mi jefe. De hecho, ese primer día me dijo algo como "Gus, necesitas mejorar la calidad sin afectar negativamente el rendimiento". Aunque el cometido era desafiante, ya había estado en circunstancias similares antes y no estaba realmente asustado (tonto).

Me puse a observar a los equipos para poder entender dónde se podían realizar mejoras. Los equipos ya estaban compuestos por excelentes desarrolladores que seguían muy buenas prácticas. Tenían una cobertura de pruebas unitarias muy alta, realizaban revisiones de código y programaban en pares, y tenían un sistema de integración continua (CI) realmente bueno.

Incluso realizaron un desarrollo basado en pruebas de aceptación (ATDD). ¡Ahora tenía miedo! Seguí observando y comencé a profundizar más para encontrar lo que faltaba. La primera sugerencia concreta que hice fue incorporar a los analistas de negocios en los equipos. Habían estado operando como un equipo separado hasta ese momento. Fue un pequeño cambio que ayudó un poco, pero todavía no sabía qué más podría estar pasando.

Pasaron unas cuantas semanas más y poco a poco comencé a comprender dónde podría radicar el problema y por qué teníamos tanto problema.

pocos incidentes en UAT e incluso en producción. La clave fue observar cómo la gente practicaba el desarrollo basado en pruebas de aceptación.

En el tablero kanban, había una columna llamada "En espera de la aprobación de BA", y después de eso, comenzó el desarrollo. Básicamente, las historias de usuario fueron entregadas por los analistas de negocios (BA) a los desarrolladores a través de una aprobación formal, momento en el cual los analistas comenzaron a trabajar en el siguiente conjunto de historias de usuario. No hubo conversaciones. Los desarrolladores tomarían las historias de los usuarios y las transformarían en lo que pensaban que eran las pruebas de aceptación y el código correspondiente.

Tuve mucha suerte de encontrar una gran aliada en Mary, otra ingeniera de pruebas que compartía el mismo deseo de mejorar las cosas. Trabajamos juntos para dimensionar el problema y utilizamos una sesión de mapeo de impacto para identificar una posible solución. El enfoque que identificamos fue crear y facilitar talleres sobre ATDD "real" con los equipos de desarrollo.

Adaptamos nuestro enfoque ATDD a nuestro contexto específico, utilizando parte del enfoque ATDD de Elisabeth Hendrickson y las especificaciones de Gojko Adzic por ejemplo. La clave fue enfatizar la importancia de

Conversaciones sobre la automatización

Les dijimos a los equipos: ATDD se trata de personas, comunicación, colaboración y entrega de valor comercial.

Movimos todo el énfasis en este aspecto y explicamos cómo el conjunto de regresión automatizada que se crea durante el proceso no es más que un resultado natural. El valor está en definir de manera colaborativa las historias de los usuarios, compartir la comprensión de la aplicación y entregar el valor comercial a nuestros clientes.

Para definir de manera colaborativa las historias de los usuarios, eliminamos la columna "Esperando la aprobación de BA" del tablero y la reemplazamos con una columna llamada "Discutir", donde los "3 Amigos" se sientan juntos, hablan sobre la historia del usuario e identifican ejemplos. que luego se convertirán en pruebas de aceptación y, a su vez, en código y, finalmente, en valor comercial. Pensé que este era un ejemplo perfecto de

Colaboración del cliente sobre la negociación del contrato.

El enfoque del taller realmente ayudó a desencadenar un cambio en el comportamiento del equipo y, en cuestión de semanas, los miembros del equipo de desarrollo estaban implementando el nuevo enfoque, con un impacto inmediato y tangible.

Después de un año, los resultados han sido sorprendentes. Los problemas en la UAT son prácticamente inexistentes y lo mismo se aplica a la producción. Los equipos realmente entienden el producto que están creando y los analistas de negocios, ingenieros de pruebas y desarrolladores trabajan como una sola unidad sólida. Por último, pero no menos importante, incluso mejoramos el rendimiento al eliminar la repetición de problemas como resultado de la mala calidad. Los malentendidos que solían surgir en la etapa UAT ahora se identifican y resuelven antes de que se escriba una línea de código.

Ah, casi lo olvido: ¡también nos divertimos mucho!

BDD

El desarrollo impulsado por el comportamiento (BDD) utiliza lenguaje natural para capturar ejemplos de clientes en un lenguaje específico de dominio. El objetivo es mantener esas conversaciones importantes con los clientes y crear escenarios dado_cuándo_entonces que expresen el comportamiento de una característica, incluidas las expectativas para una condición y acción particular, en pruebas que todos los miembros del equipo comprendan.

La descripción de Dan North de BDD (North, 2006) es un poco más complicada, pero la interpretamos como el uso de la palabra comportamiento en lugar de prueba, y el uso de las pautas dada_cuándo_entonces para describir la condición previa, el desencadenante y la condición posterior de lo que es. esperado, antes de que comience la codificación. En una publicación del grupo Yahoo Agile Testing en agosto de 2010, Liz Keogh explicó (Keogh, 2010):

El enfoque de BDD está en el descubrimiento de cosas que no sabíamos, particularmente en torno a los contextos en los que tienen lugar los escenarios o ejemplos. Aquí es donde entra en juego el uso de palabras como “debería” y “comportamiento”, en lugar de “probar”, porque para la mayoría de las personas “probar” presupone que sabemos cuál debería ser el comportamiento. El “debería” nos permite preguntar: “¿Debería?” ¿En realidad? ¿Hay algún contexto que nos falta en el que se comporta de manera diferente?

Originalmente una respuesta a TDD, BDD ha evolucionado hacia análisis y pruebas automatizadas en el nivel de aceptación. Feature injection es un marco de proceso de análisis empresarial que centra BDD y TDD en la entrega de valor empresarial (Matts y Adzic, 2011). Los equipos que utilizan la inyección de características comienzan definiendo y comunicando el valor comercial, crean una lista de características para impulsar la entrega de ese valor comercial y luego amplían el alcance de esas características, investigando ejemplos de alto nivel de uso por parte del cliente. Incluimos ejemplos de pruebas de estilo BDD más adelante en este capítulo.

SBE

Como resultado del taller del grupo de herramientas de prueba funcional de Agile Alliance (AA-FTT) en Agile 2010, Declan Whelan, Gojko Adzic y algunos otros idearon un diagrama (Figura 11-3) para tratar de obtener una comprensión común. respetar las especificaciones con el ejemplo.

Mire atentamente el diagrama. Puede notar que no hay una sola mención a las pruebas. SBE comienza con el objetivo en mente de derivar el alcance. Aquí es donde técnicas como el mapeo de impacto nos brindan un gran comienzo al identificar primero los objetivos correctos. Para llegar a ejemplos clave que expliquen suficientemente el contexto, el equipo especifica de manera colaborativa, tal vez en un taller de especificación o actividad similar. Estos ejemplos clave se convierten en nuestras pruebas de aceptación de alto nivel. Refinamos esos ejemplos, extrayendo un conjunto mínimo para especificar una regla de negocio adecuadamente; estos se convierten en las pruebas de la historia. En este punto, automatizamos sin cambiar el significado, y esas pruebas automatizadas se convierten en comprobaciones para brindar información rápida y regular. El resultado: especificaciones ejecutables o documentación viva que describe la funcionalidad del sistema en cualquier momento.

Gojko Adzic ha explicado cómo los equipos pueden utilizar talleres de especificación para crear ejemplos, logrando que las personas adecuadas colaboren para comprender bien qué construir (Adzic, 2009). Consulte la bibliografía de la Parte IV, “Prueba del valor empresarial”, para obtener más enlaces para obtener más información sobre SBE.

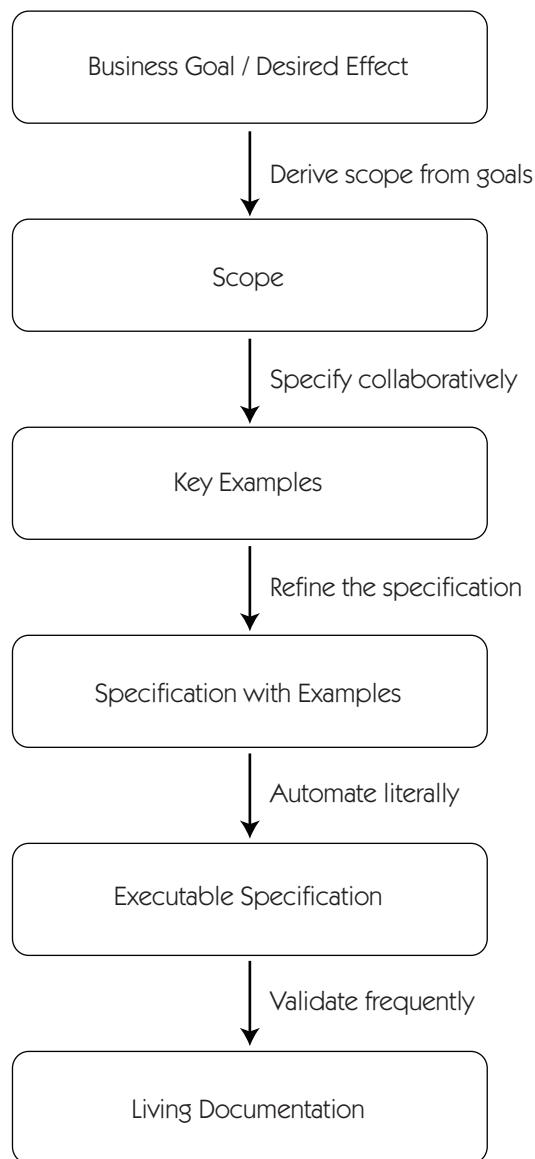


Figure 11-3 Specification by example (Adzic et al., 2010)

WHERE TO GET EXAMPLES

Whatever we call the process, we want to minimize rework and misunderstandings of what to build. In *Agile Testing* (pp. 378–80), we gave some simple ways to collaborate with customers to specify examples. We'll explore a few others here. The best way to get examples of how a particular feature or story should work is to sit with your customers and watch as they do their jobs. Try to imagine where the new feature will help or hinder them.

Informal brainstorming sessions where participants draw on whiteboards (physical or virtual) usually work well. More formal specification workshops are another option; however, getting the right people to participate is key (Adzic, 2009).

Examples can be elicited as part of an impact-mapping session, and they naturally emerge in story-mapping workshops. See Chapter 9, “Are We Building the Right Thing?,” for details on these practices. Whatever you use for the format of your meeting to obtain examples, set a time limit, and keep the meeting short and focused. People can stay fresh during an hour-long meeting, but if you need more time, schedule another session for a few days later. That gives everyone time to ruminiate on what they want and come up with useful examples.

Support tickets and community forums can be sources of examples of what people need to do with your product. The frequency with which a particular feature is requested indicates what's most important to users.

Lisa's Story

We decided to upgrade our product's search engine. As this involved re-indexing everything in our database, it was a good opportunity to add some search functionality. We didn't have a lot of time to spend, but we could go for the “low-hanging fruit.”

Our testing and customer support manager already had a good sense of what customers wanted. She looked through feature requests and complaints on our community forum as well as past customer support tickets and listed the most-requested search functionalities. Our delivery team was able to add a surprising number of them. It was a big win that really pleased our customers.

Ellen Gottesdiener and Mary Gorman use the template shown in Figure 11-4 to get examples in a given_when_then format (Gottesdiener and Gorman, 2012). It ties the story to the scenario, along with its data. It also makes the writer consider some of the dimensions we talked

Story	As a supervisor, I can search in the employee database so that I find a specific employee's information.
Scenario	Valid name search returns results.
Business rule	Search returns only employee names who report to the requesting supervisor.
Given	
Precondition(s), state	A supervisor exists with direct reports.
Fixed data	Supervisor: Kant Employee Last Name: Smith
When	
Action	Kant navigates to the employee name search page and enters the value "S."
Input data	Search criteria – "S"
Then	
Observable outcome: Message, output data	Kant sees a search result that includes Smith.
Post-condition(s), state	Smith is displayed in search result.

Figure 11-4 Given_when_then template (Gottesdiener and Gorman, 2012)

about in Chapter 9, “Are We Building the Right Thing?” Not all stories need a template like this, but it helps draw out the ideas in new areas of the code.

Use business rules, use cases, business expertise (customers, business analysts, domain experts, etc.), the expertise of the delivery team (programmers, testers, DBAs, etc.), along with any other supporting information you might have to get examples.



Make time to learn your business domain as much as is practical, so you can help stakeholders create and articulate useful examples. This knowledge enables you to zero in on the purpose of each feature and strip out unneeded functionality or simplify the approach.

BENEFITS OF USING EXAMPLES

We can get examples of desired system behavior from everyone with a stake in the solution. This is especially important for internal stakeholders such as those in the accounting and legal departments, who are often overlooked in the requirements-gathering process. Thinking in terms of concrete examples helps business experts articulate business rules and engages stakeholders more fully in the development process. If possible, sit down with a business expert, observe how they do their work, and create examples together.



Concrete examples make it much easier for the delivery team, including programmers, testers, operations, user experience designers, and database experts, to understand the needs of the different customers. Rather than having a philosophical discussion of what the best implementation might be, people draw on whiteboards and fill spreadsheets with example inputs and expected outputs to express business rules in terms everyone can understand. We bring all these examples together and turn them into tests to define system behavior, thereby reducing the uncertainty about what needs to be built.

A DSL is one way customers can express their examples in a way that they and the development team can use. A `given_when_then` style like the examples earlier in this chapter may suit your organization. If you’re testing calculations and algorithms, you might prefer a tabular approach, such as a spreadsheet with defined inputs and expected

outputs. Many test frameworks allow examples in a DSL format to be automated directly so they not only are readable by non-programmer team members but can also be executed as part of a continuous integration process.

Programmers can use the DSL examples to help with TDD as well, although TDD is as much about designing the code as creating the unit tests. The broad and deep understanding of the feature that examples provide helps programmers know what code to write.

Preventing Defects with Examples

Cory Maksymchuk, a programmer in Winnipeg, Canada,
explains how test-first coding helps cover a range of example
scenarios and prevents defects.

I was once working on a section of a screen with a dynamically generated drop-down “select box.” The contents of the select box would be different based on which user from which organization was viewing it. Although there were quite a few scenarios for what it should do, I had a good handle on them in my head and in the requirements document. At the time, we had quite a few developers and were low on analyst and tester availability, so I was working ahead of our test cases. I finished the code, complete with unit tests, sent it for code review, and committed it. A day later I received the test cases, of which there were 52. My work passed 14 of them and failed the other 38.

Had I only worked through a requirements document and my code made it to production, there would have been a defect log with a pile of work in it. By writing test cases first, we make sure all cases are covered before committing code. Without defects making it to production or even being seen by a business user, we increase confidence, eliminate months of rework and maintenance, and look like stars. After we recommitted to our test-first strategy, this project went live seven months later, on time and on budget. Four developers, two analysts, one tester, and a lead worked on it, and a year later we have received only one small defect report.

Test-first helps us cover all viable scenarios in a way that the users can read and understand. It allows us to understand what “done” means and to not waste our time writing code that is not covered by tests. Couple this with good coding style, thorough code reviews, and good coverage in unit and integration testing, and you are on your way to a no-defects world.

POTENTIAL PITFALLS OF USING EXAMPLES

As with any testing practice, there are risks to guiding development with examples. As your team learns to collaborate with customers to elicit examples and turn those examples into potentially executable tests, be aware of the ways this can go wrong.

Getting Bogged Down in the Details

Gathering any type of requirements before coding begins is a slippery slope toward a requirements phase. When you provide customers with an easy way to create examples, they may tend to get carried away. Detail-oriented business experts also may be overeager and provide every edge case they can think of. When programmers start working on the feature or story, they may be so overwhelmed with detail they can't see the main purpose of what they need to code.

Needing a large number of examples, or extremely complicated examples, may be a sign of a deeper problem, such as incorrect software design or inadequate models. If examples seem too complicated, there may be a flaw or some fundamental gap in the design or the domain model. The more complexity there is in a feature, the more conversation there needs to be to reduce the uncertainty.

Experiment with how much detail is appropriate for your team. If you're planning a feature similar to work your team has done before, you may not need to spend much time discussing examples. If your customers have prioritized a big, risky feature set, hold a time-boxed meeting a few weeks in advance of when development will start, with both development and customer teams, to start discussing examples. If you need more time, schedule another meeting.

In our experience, it works best to limit examples to a few happy path, sad path, and ugly path scenarios when working this far in advance. In story readiness meetings before the iteration, go through the stories again with the business experts to ensure a shared understanding of the feature. When coding begins on a story, it's time to get into the pickier details and make sure all viable cases are covered. Programmers can start by writing code to make the happy path tests pass, then move into the boundary conditions and negative and edge cases. In *Agile Testing*,

we called this “working with steel threads”—work on the core, and then add complexity.

Chapters 8 and 18 of *Agile Testing* gave more information on the appropriate level of detail for examples throughout the incremental and iterative development process. Each situation is unique, so use retrospectives to see if tests based on examples are providing the right amount of information at different times in the coding and testing cycle.

Lacking Buy-in



Customers and delivery teams need to collaborate to get examples and use those to guide coding. This requires buy-in on all sides. Business stakeholders may feel they’re too busy or that it’s someone else’s job to create requirements for features. We have to show them what’s in it for them—what the benefits are.

Start by proposing an experiment. Explain the benefits of using examples, and suggest trying them for a new feature that’s going to be developed soon. Schedule short meetings to create examples, and don’t run over time. If you don’t finish, schedule another meeting in a few days. Encourage customers to draw flow diagrams, mind maps, or other visuals as they think of examples. Use formats that are appropriate to the domain, such as spreadsheets for financial applications.

As you discuss more detailed examples, show business experts how you turn appropriate ones into executable tests in the agreed-upon DSL. Demo the resulting software early and often. Even seeing only the happy path behavior work will help customers feel their time is well invested.

Programmer buy-in can also be tricky. Programmers might feel they don’t have time to bother with ATDD, especially if they’re already doing TDD at the unit level. If the DSL used for acceptance-level examples is different from the format of their unit tests, it might mean too much task switching for their comfort level.

Testers might even need convincing if they are used to defining everything abstractly, such as, “Verify the name is valid,” rather than using an example to say that, “Janet Gregory is an example of a valid name, but Janet#Gregory is not.” Be prepared to iterate through different

approaches to express and automate examples before finding the one that works for your team.

Too Many Regression Tests

Today's test frameworks make writing test cases in a DSL easy and fast. You can even use them for automated exploratory testing, especially at the API or service level, cranking through a wide variety of inputs and trying all the crazy edge cases. This is great, but beware of keeping all these executable tests in your automated regression suites. Automated regression tests provide a safety net, but they also need to give fast feedback, and we can't spend more time maintaining them than the value they deliver warrants. Once a test is passing, evaluate whether it adds enough value to include it in a regression suite. If it tests an edge case that, if it failed, would be low risk, you might prefer to leave it out or put it in a suite that runs only once per day or right before release. If unit tests cover the same area of the code fairly well, you may not need the higher-level test unless you need it for business visibility.

It's a trade-off between risk and cost. Over eight years, Lisa's previous team had two regression failures in production that required rolling back the release. They had created automated tests that would have caught both of the regressions, but the tests weren't in the suites that ran in the CI. They had made a conscious decision to leave them out, thinking they were edge cases that would never happen. On the one hand, this seems bad, but really, this was only two failures in eight years. That's offset by a lower test maintenance cost by having fewer automated regression tests. This is another area where lots of experimenting is needed to find the right balance.

Not Enough Is Known Yet

If your team starts on a new feature or capability that you've never done before, and nobody on the team has experience in that area, it may be too early to specify concrete examples. As Liz Keogh writes (Keogh, 2012a):

When you start writing tests, or having discussions, and the requirements begin changing underneath you because of what you discover as a result, that's complex. You can look back at what you end up with and understand that it's much better, but you can't come up with it to start

with, nor can you define what “better” will look like and try to reach it. It emerges as you work.

When there are too many unknowns, or you don’t even yet know what you don’t know, it’s more useful to spike solutions and get quick feedback to see if you’re going in the right direction. Programmer-tester pairing to explore potential behavior may be useful at this time. The product ideas and technical implementations have to emerge.

THE MECHANICS OF USING EXAMPLES TO GUIDE CODING

Example-driven development, whether you use ATDD, BDD, or SBE, is a core practice for agile teams, and it takes work and practice to learn how to do it effectively. Please see the bibliography for Part IV for books and other resources that teach ATDD/BDD/SBE through examples.

SUMMARY

For years now, we’ve followed this motto coined by Brian Marick: “An example would be handy right about now.” If you find yourself embroiled in an unending team discussion about how a particular feature should work, grab a marker and a whiteboard or flip chart (or the virtual equivalent) and start asking for concrete examples. You’ll quickly make progress toward defining what that feature will look and act like once it’s “done.” Key points in this chapter are

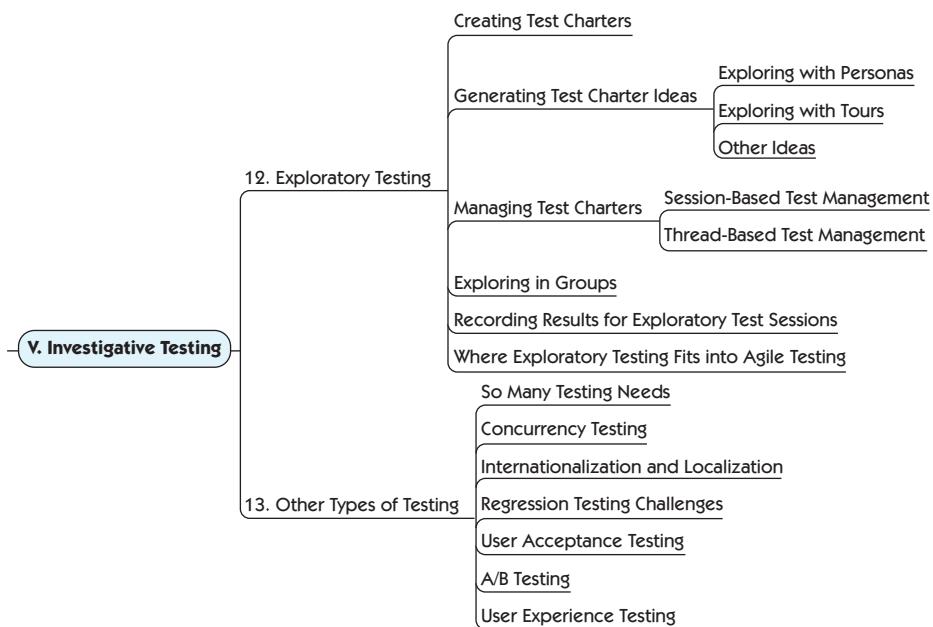
- The power of using examples
- Overviews of the popular approaches to guiding development with business-facing examples:
 - Acceptance-test-driven development (ATDD)
 - Behavior-driven development (BDD)
 - Specification by example (SBE)
- Where and how to get examples from business experts
- Benefits of using examples
- Potential pitfalls of guiding coding with examples
- The mechanics of example-driven development

Part V

INVESTIGATIVE TESTING

In Part IV, “Testing Business Value,” we looked at ways to help customers set goals that bring value to the business, expressed examples of how software will help achieve those goals, and turned those examples into tests that guide development. These activities give us confidence that we can build the right software for our business. However, we can’t know for sure that the testing we do based on the initial ideas about how features should behave will actually ensure that all our customers’ needs are met. We need to learn more about the features as we build them, working in a series of small experiments and building in short feedback loops to refine requirements.

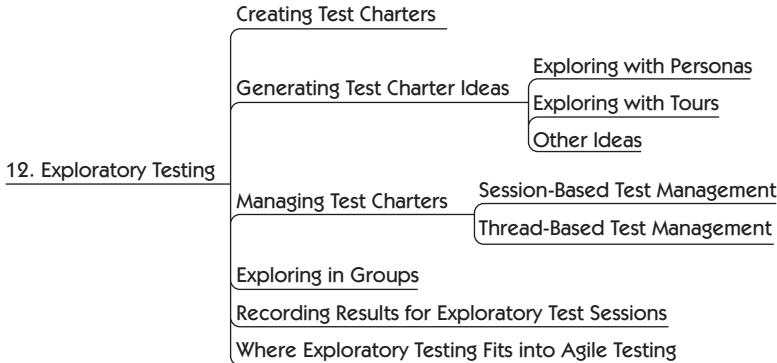
As programmers commit new and updated code, we look at the product to see if it meets expectations based on the experience of the people testing it. We investigate the results of each experiment to inform the next. Exploratory testing is one way to investigate. Because agile teams deliver new code frequently, they may tend to focus on functional testing. However, there are many other kinds of tests that question the code and how the team interpreted the business needs. In Part V, we’ll consider a variety of quality attributes that agile teams may need to investigate as code is completed.



- **Chapter 12, “Exploratory Testing”**
- **Chapter 13, “Other Types of Testing”**

Chapter 12

EXPLORATORY TESTING



Exploratory testing combines test design with test execution and focuses on learning about the application under test. It requires a mixture of thinking processes: logical, calculating, and conscious, along with fast and instinctive. In Chapter 4, “Thinking Skills for Testing,” we discussed how different styles of thinking are needed for different types of testing. Exploratory testing is one area that particularly benefits from applying different ways of thinking. In this chapter we will share how exploratory testing has developed and how you can practice it in your agile teams.

Exploratory testers must consider many aspects of the product, including its users, how the feature they’re testing relates to the company’s business goals, what ripple effects implementing the feature might have on other parts of the system, and what the competition is doing. In our experience, testers who are skilled at exploratory testing bring enormous value to their teams.

Lisa's Story

My current team hired its first full-time tester less than two years before I joined as the third tester. Working with testers was new to the company

culture. The team developed everything test-first and did a great job of test automation at all levels. After we three testers had been working together with the team for more than a year, I was surprised and gratified when one of the programmers noted on a company forum that he thought that no amount of automated tests could replace skilled, dedicated testers who know how to do exploratory testing well.

Exploratory testers do not enter into a test session with predefined, expected results. Instead, they compare the behavior of the system against what they might expect, based on experience, heuristics, and perhaps oracles. The difference is subtle, but meaningful.

James Lyndsay explains the subtle difference between scripted testing and exploratory testing in his paper “Why Exploration Has a Place in Any Strategy” (Lyndsay, 2006). He notes:

An automated test won't tell you that the system's slow, unless you tell it to look in advance. It won't tell you that the window leaves a persistent shadow, that every other record in the database has been trashed, that even the false are returning true, unless it knows where to look, and what to look for. Sure, you may notice a problem as you dig through the reams of data you've asked it to gather, but then we're back to exploratory techniques again.

In another paper, “Testing in an Agile Environment” (Lyndsay, 2007), James suggests that one of the roles that testers can play is the “bad customer.” A bad customer goes off the happy path and may even try to break the system. Using your knowledge of potential issues, you can play the part of the “bad customer,” whether that’s someone who’s malicious, in too much of a hurry, or simply incompetent. Here are some examples of actions you might try as this persona:

- Invalid parts of input—characters, values, combinations
- Time changes
- Unusual uses
- Too much—long strings, large numbers, many instances
- Stop halfway/jump in halfway
- Wrong assumptions

- Making lots of mistakes, compounding mistakes
- Using the same information for different entities
- Triggering error messages
- Going too fast



You may not consider yourself to be a skilled exploratory tester, but you may have tried typing in all the special characters on the keyboard or blowing out the buffer of an input field. You can use these instincts in a more mindful way. Elisabeth Hendrickson’s “Test Heuristics Cheat Sheet” (Hendrickson, 2011) is a great place to start looking for ideas. Exploratory testing gives you the opportunity to use your ability to critique, assess, and challenge your understanding of the product in a purposeful manner, in order to provide critical information to your stakeholders. Later in this chapter we will suggest ways to give constructive feedback to stakeholders.

Exploratory testing and automation aren’t mutually exclusive but rather work in conjunction. Automation handles the day-to-day repetitive regression testing (checking), which enables the exploratory testers to test all the things the team didn’t think about before coding. As you explore, you may find additional tests that need more investigation or should be automated. You will likely use automation to set up exploratory scenarios, to monitor log files, or perhaps to explore scenarios that are not accessible through manual means. For complex new features where there are many “unknown unknowns,” testers and programmers can explore together as they spike potential solutions to learn enough about the feature to start writing stories and tests to guide development.

There are multiple ways to explore. You may explore alone, or in pairs or groups. In this chapter, we’ll discuss a few techniques that have worked well for us, and we’ve included stories from those who have had other experiences. We encourage you to experiment with these approaches as well, using lightweight strategies to manage them. The bibliography for Part V, “Investigative Testing,” contains many books, articles, and other resources for learning more about this powerful testing approach. Make exploratory testing an important part of your toolkit when testing stories, features, and the system as a whole.

CREATING TEST CHARTERS

A test charter outlines a goal or mission for your exploratory session. There is no one right way to create a test charter, but we'll look at a few different methods. As always, it's best to experiment and find the one that works for you. You may find that one style works best for session-based test management (SBTM), while another works better for testing with thread-based test management (TBTM). Charters may range from happy path functional validation (although this is perhaps less valuable) to exploring failure modes, and performance and scalability.

Elisabeth Hendrickson's *Explore It!* (Hendrickson, 2013) has a section on creating good charters. It takes practice to find the right level of detail for your purposes. Too specific means that you don't have enough room to wander off the beaten path to make unexpected discoveries. Too vague or broad doesn't give enough focus and may cause you to waste time.

Experiment with how you word charters, as well as with the number of charters you create. Elisabeth notes that "a good charter is a prompt: it suggests sources of inspiration without dictating precise actions or outcomes" (Hendrickson, 2013, p. 16). One template Elisabeth has found that works well and gives enough guidance and focus is

Explore ... <target>
With ... <resources>
To discover ... <information>

It's better to have multiple charters, each of which is concise and focuses on a single area, for example:

Explore editing profiles
With real usernames
To discover if there are instances where username constraints are not enforced

Another way to create a test charter is a mission statement and areas to be tested, for example:

Analyze the edit menu functionality of Product X

And report on areas of potential risk in Operating System Y

The simpler you keep a charter, the easier it is to stick to it. However, James Lyndsay reminds us that the broader you make a charter, the easier it is to consider distractions within the charter (Lyndsay, 2014). The breadth and specificity of the charter are tools to guide exploration. For example, a charter that says “Performance Test X” really gives no guidance.

Let’s use an example of a web-based toy store to write a charter for the end game during a delivery cycle. *Note:* The feature would have been explored for the workflow as soon as it was completed.

Explore shopping for a new toy

With a real live user

To identify potential bottlenecks and unexpected disadvantages

Again, we’re not suggesting that one way is better than another. On agile teams, we move at a fast pace, so we want to keep focused on the stories and features we’re currently developing. At the same time, we have to keep the big picture in mind and make sure new code doesn’t cause unintended effects elsewhere in the application.

About Sessions and Charters

Matt Heusser, who writes and consults about testing and is based in the United States, shares his experiences with charters and session-based testing.

A couple of years ago I was consulting for a large corporation that used an agile method. The business domain was incredibly complex, with legacy systems composed of many layers added by different people over decades. Each team contained everything needed to ship software, including support and operations, and each team

determined its own work process independently. To try to share information, the company had a monthly “tester community of practice” meeting.

The “charters” the team adopted were one to three pages long in MS Word and were relatively prescriptive. It was an improvement from what they did before, but not what I think about when I say “test charter.”

For me, a charter is a mission for testing. In an agile environment, where we assume regular functional testing as part of story work, a charter might address a risk we want to investigate or invest time to mitigate. Here are some examples:

- Test <feature> under <new browser version>
- Test <system subset> with tablets
- Test for failures with <third-party web services that are usually <up>
- Test multi-user and race conditions with <feature>
- Additional exploratory penetration testing for the new web front end as a system
- Customers see <X> but we are unable to reproduce the problem; can you troubleshoot?

Let’s take another look at these charters. The examples are the intersection of a specific risk and a feature, but they don’t have to be. I like my charters to be less than 200 characters long—most of the time, they fit into a tweet. That tweet is the title. If you have specific concerns or spend five minutes brainstorming, you might create a list of test ideas that become the “body” of the charter.

You can use charters in many ways. I’ve made heavy use of charters to help manage the release process with limited time. I tend to think of charters as small chunks of work, usually 30 to 60 minutes. Of course, if we identify a risk that can be investigated during the iteration, doing it earlier is better. When the charter makes sense to tie into a story, we can just put it in the story notes. When the charter addresses a crosscutting concern or is unrelated to current stories, I might suggest creating a new story.

By taking notes and debriefing a fellow team member on what we’ve done after the session, we have a chance to increase team knowledge about the system and test practices. This serves as a sort of mini retrospective for testing.

Start with the style of charter that seems most workable for your team. Trying different formats is a nice way to shake things up and help you see your software with a fresh perspective. You can think about the templates we've provided as training wheels until you find the one that works for you.

GENERATING TEST CHARTER IDEAS

There are a few techniques that can help generate test ideas that we'll share in this next section. It is not an exhaustive list, but we hope it will trigger some of your own ideas.

Exploring with Personas

Personas are a way of understanding your end users, and many companies create personas as part of their marketing strategy.

Jeff Patton (Patton, 2010) and David Hussman (Hussman, 2011) are among the many practitioners who create pragmatic personas to identify who actually uses a product. Personas are a good way to imagine different ways people will use an application. We explained how we use personas for usability testing in *Agile Testing* (pp. 202–4), but we think the concept can be taken beyond usability. If you currently don't have defined personas, conduct a quick workshop with your team to discover at least some of them to give you a start. We've mentioned James Lyndsay's “bad customer,” and we've shared two personas (see Figures 12-1 and 12-2) that Mike Talks has used for testing login account functionality and security.

Concentrate as a user on the following chunks of functionality:

- Suspend your account.
- Deactivate the RSA token linked with an account.
- Delete your account.

Potentially this user will need help from the help desk.

The Security Worried User: This user has concerns about using the login service and wants to be protected from anything bad happening. After all, you hear such terrible things in the news about people's identities being stolen.



Figure 12-1 Security Worried User persona

Try to support the Help Desk admin with the following:

- Search user.
- Authenticate identity.
- Deactivate RSA token.
- Delete account.

Flows to follow:

- Create new account.
- Set account to be Help Desk User or Help Desk Admin.
- Delete account.
- Get reports of activity.

By its nature you will also be touching upon

- Monitoring activity
- Validation

Personas are a great way to look at your application from different angles.

The Help Desk Admin: The administrator can give permissions to other users to make them Help Desk Users or Help Desk Admins. The Help Desk Admin goes beyond the Help Desk User in what the person can support and change in people's accounts.



Figure 12-2 Help Desk Admin persona

Lisa's Story

My team experimented with creating a separate project for system testing a complete rewrite of our agile project-tracking product. A tester, a designer, and a marketing expert teamed up to identify various personas representing our users. Developer Denise and Product Paul were two representatives of our product's users. We wrote charters as user stories so we could track them in our online tracking tool. This was a good way to track the testing we felt was needed, but not a good way to capture the results of our exploratory testing sessions. See the section "Recording Results for Exploratory Test Sessions" later in this chapter on how we did that.

Persona: Paul is the project manager for Agile Toys. In his weekly iteration planning meetings (IPMs) with the team, he goes through stories in the upcoming iteration, answers questions, updates the stories with point estimates, and rearranges stories in the backlog according to priority. He typically uses his iPad in the IPM to make the changes. At his desk, he uses Chrome on a MacBook Air. Paul's main usage of the tracking tool is keeping the backlog organized and prioritized.

Charter: Explore as Paul, the project manager, in an IPM to discover any issues with concurrent updates to stories from different devices.

Scenarios:

- Pre-IPM backlog prioritizing
- Iteration planning meeting—updating stories

Each of us sets aside an hour a day for exploratory system testing for what we call “mini group hugs,” where each tester chooses a persona and a browser and tests concurrent usage of the system. We recorded our test results on our team wiki.

Using this process, we’ve found important bugs that weren’t found while doing exploratory testing on the individual user stories. For example, an exploratory testing session with a charter of updating stories as Paul would during an IPM turned up several new bugs related to concurrent usage.

If you use personas, make sure your whole team understands them and how they can help with testing. Make them visible, perhaps by pinning their pictures and profiles on the wall. It is a good way for the programmers to get a better understanding of the users, and it also helps raise awareness of the value of exploratory testing.

Exploring with Tours

Tours can be another useful tool to generate ideas for exploratory charters or to get familiar with a new product or capability. This technique uses a metaphor of tourism and can add useful variation to your explorations, uncovering issues you may not see otherwise. James Whittaker has described some unique exploratory testing tours (Whittaker, 2012).

For example, as a tourist, perhaps you want a strategy for seeing the most important sights in London. Who you are, or what your goal is, will determine your strategy. Whittaker suggests that visiting students would approach this situation much differently from a group of flight attendants who are there for a weekend. A similar approach can help you explore your software features. Check out Whittaker’s defined tours, such as the Guidebook Tour (looking for bargains, shortcuts) or the Landmark Tour (hopping through an application’s landmarks). In the

Landmark Tour, you would identify a set of software capabilities (the landmarks) and then visit those landmarks, perhaps randomizing the order. Changing the sequence of events may cause an unexpected error to occur. These are good places to start, but make them your own. For example, try combining personas with a tour to visit your most important landmarks.

Tours can be done at any time, but Janet has had great success defining tours for the end game, when you think your new release is ready to ship. Often this will give your team extra confidence that the most important features in your product work as expected. Be creative in how you document these touring sessions. Perhaps you can create a visual map to show where you've been. See the bibliography for Part V for links to explore some of the possibilities.

Markus Gärtner (Gärtner, 2014) recommends debriefing after completing each tour. As you debrief, you'll identify more charters, which will take you deeper into areas you briefly touched on in the tours.

Other Ideas

Some teams base their charters on their story acceptance criteria. For example, you may want to explore error handling, perceived response time, and complementary features.

If you have identified risks during story elaboration, you may create risk-based charters that will highlight likely problem areas or areas of uncertainty. A conversation with your programmer is an excellent source for identifying architectural risks and makes a great driver for test charters.

David Hussman (Hussman, 2013) suggests creating journeys to take the personas you've identified someplace interesting. Once you've learned more about the personas through story mapping, imagine where they might like to go. These journeys might also be a useful way to explore your system after it is built. For example, if we refer back to the story

map in Chapter 9, “Are We Building the Right Thing?” (refer to Figure 9-2), one way to test it might be to describe a charter as a possible journey:

Journey: Search by keyword, select an email, add sender to contacts, and reply.

Charter:

Explore journey

With different folders, different senders

To discover if flow hangs together

In Chapter 13, “Other Types of Testing,” we will look at some ways to use exploratory testing for several quality attributes beyond the scope of functional tests.

MANAGING TEST CHARTERS

By now you should have some great ideas for creating your exploratory test charters. The question now is, How do you keep them straight? There are a few different ways, and we’ve shared a couple of stories from exploratory testing practitioners about how they manage their charters.

Session-Based Test Management

Session-based test management (SBTM) is based on the idea of creating test charters or missions for a testing idea, exploring uninterrupted for a specific time period, recording the results, and following up with a debriefing session. We mentioned this technique in *Agile Testing* (p. 243) but will share a few other ideas we’ve gathered. For example, Bernice Niel Ruhland told us she uses SBTM to help train testers. The debrief session is a great way for her to provide immediate feedback to the new testers.

Managing Regression Tests with SBTM Charters

Matt Heusser shares his experiences with SBTM and how he uses it to manage a manual regression suite.

SBTM fits well in an agile development process. It radically improves the performance of the test process so that it can fit into tight time boxes in a way that stands up to scrutiny.

SBTM uses a cognitive, thinking, and investigative approach, which also makes it work well in an agile setting. Using SBTM, I've managed to compress regression testing to fit within an iteration for teams that have almost no test tooling and no customer-facing automated testing infrastructure. We do use automation to help, for example, when we generate test data and make build environments faster.

To make it work, the software has to be reasonably high quality before you get to the "final shakedown" test. Otherwise, the find/fix/retest loop kills you. Jon Bach's SBTM paper (Bach, 2000) gives more specifics, but here's one way to adapt the method in an agile context. Consider this exercise for a whole agile team:

1. Make a list of risks for the product in charters, 30 to 60 minutes apiece.
2. Put all the lists on a wall, whiteboard, or projected page.
3. Sort the list by risk. Dot vote if you'd like to determine priority.
4. Put the list on a kanban board.
5. Team members pull one charter at a time and test, reporting and fixing bugs as a whole team as they go.
6. At any time, anyone can change the order of the cards or add new cards.
7. When you run out of time, stop testing. Look at the known issues and the risks not tested, and ask yourself, "Are we ready to ship?" Conduct a Roman vote, thumbs up or down.

SBTM is a whole-team, people-centered approach to testing. It can help with designing a test strategy for a legacy application. If your application has complicated rendering code or business logic in the graphical user interface (GUI), consider using SBTM in combination with high-level GUI smoke tests, rather than trying to cover everything with automated GUI tests. This combination can provide better coverage at a lower cost and let your team make better use of their test time.

Also consider SBTM when you see risks in platform or third-party compatibility. For example, SBTM can help you catch failures, resulting in an upgrade to jQuery or a third-party open-source framework.

SBTM is useful if you need to support a new platform. For example, if your team is unexpectedly faced with supporting mobile devices, the touchscreen platform poses usability and compatibility risks. You can manage these well through session-based testing.

Complex business domains, high cost of failure, and other scenarios where early bug detection is critical are all motivating factors to use SBTM. By all means, tweak the process to improve the quality of the code before it gets to final test—but you can also add a layer of SBTM as a compensation mechanism. When the consequences of a mistake are high, you can reduce risk with SBTM.

Exploratory testing provides a safety net. Managing exploratory testing through sessions provides one more layer of protection and includes a feedback mechanism to help the team learn to improve in its exploration.

Try SBTM with your team. As with any testing technique we mention, there's no one perfectly correct way to do it. Conduct a session, see how it works for you, inspect, and adapt. Check the Part V bibliography for resources to learn more, including James Lyndsay's "Adventures in Session-Based Testing" (Lyndsay, 2003).

Thread-Based Test Management

Thread-based test management (TBTM) is less rigid in terms of time-boxing a session than SBTM. It works on the idea of organizing tests around threads of activity, rather than test sessions or artifacts. A thread does not imply a commitment of a specific time box as SBTM does. Its flexibility may lead the tester in different directions. TBTM may work better in some situations with rapidly changing priorities or frequent interruptions.

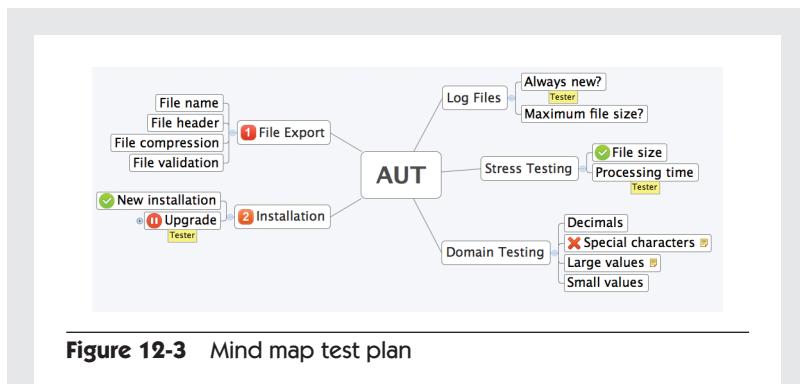
Converting from Session-Based to Thread-Based

Christin Wiedemann, a tester, trainer, and speaker from Vancouver, Canada, shares the challenges she faces with SBTM and how she adapted the process to TBTM with mind maps.

Over the years, I have repeatedly found myself in environments that are too volatile for more rigid test approaches to work well, especially as deadlines draw nearer. One example is a small team of four developers and two testers, working on a product for an external client. We had adopted a hybrid test approach, mixing scripted manual test cases and exploratory testing, organized in sessions structured according to the principles of SBTM.

As we got closer to the launch date, the pace increased drastically. As a result of increased pace and the rapid changes, we abandoned the scripted tests and ran only exploratory test sessions. However, we would typically start a test activity but quickly get interrupted and not complete the task. Working in this manner caused a great amount of stress, since it felt as if we never finished anything and never made any progress. We also came to realize that in some cases we felt hemmed in by the actual session. Even if the conditions changed or something urgent came up, we still felt obliged to finish the session before we started a new activity. Something had to change, so we decided to give TBTM a try.

The first step was to make a mind map containing all test ideas as threads (see Figure 12-3). We grouped the test threads in different ways. Most groups represented function areas (e.g., installation), but other groups were formed based on the type of testing (e.g., stress testing). After a while we realized that we wanted a third type of group that corresponded to issues we encountered frequently—potential problem patterns or heuristics such as log files. Each group corresponded to a node in the mind map. In some cases, short notes were needed to explain the thread, and we simply added those notes to the mind map. We also prioritized the function areas based on risk and added this information to the mind map by labeling the different nodes with numbers. This simple, yet powerful, test plan took us less than an hour to draft.



During the test period, we were constantly updating the mind map, and it always gave an accurate picture of the current status of testing (see Figure 12-4). As soon as someone started working on a thread, we would make a note in the mind map. Threads grew by the addition of short notes summarizing our testing and our observations, and we added new threads as we came up with new ideas. When we found defects, the corresponding thread was marked with a red cross, and the identification number from our defect-tracking system was added together with a short description (see Figure 12-5). Threads for which we felt sufficient testing for delivery had been done were checked off in green.

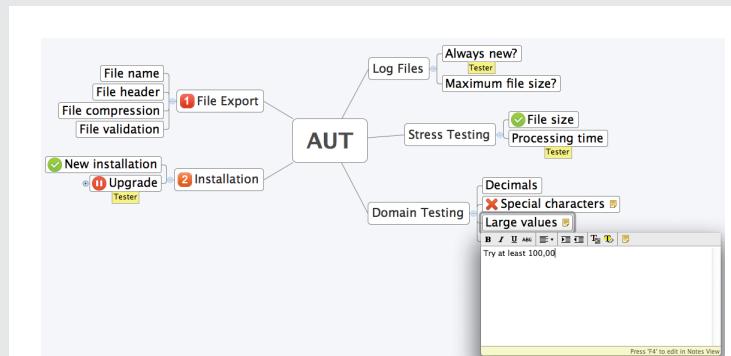


Figure 12-4 Test results with notes in a mind map

Initially our goal was to also write daily status reports containing a few short notes on what had been tested, but in reality we kept this up for only four days. With constant updates to the mind map, we really

didn't feel a need for additional reporting. At the end of our test period, we used the latest version of the mind map as our test report, which was sent to the customer.

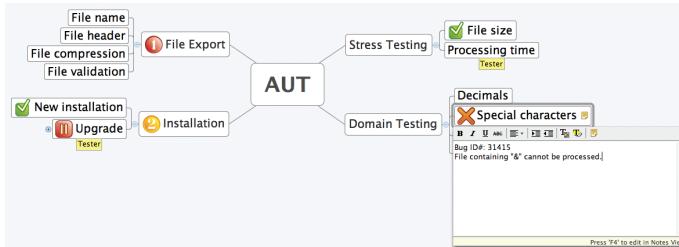


Figure 12-5 Defect logged in a mind map

What was the main difference between this project and previous projects I had worked on? The first thing that comes to mind is that this time we actually used the test plan to guide our work! In addition, the test report was continually read and reread, both to review our test results and to plan the testing on a subsequent project. Keeping the mind map up-to-date was much easier than updating any other kind of status document, which meant that it actually got updated. As it turned out, even the developers liked it—they preferred looking at the mind map over using our bug-tracking tool!

TBTM works very well under some circumstances, but I've found that mind maps are great tools regardless of the test approach. Nowadays I always use mind maps to plan testing. This not only allows me a higher degree of control without causing additional overhead, but it also makes it easier to initiate discussions and obtain feedback within the project team.

Christin's team grouped threads by functionality or type of testing, but threads can also be based around a feature or a story. They can be organized based on common resources—for example, small-scale functional data threads versus large-scale performance threads, or threads that focus on failure modes versus threads that focus on happy path workflow.

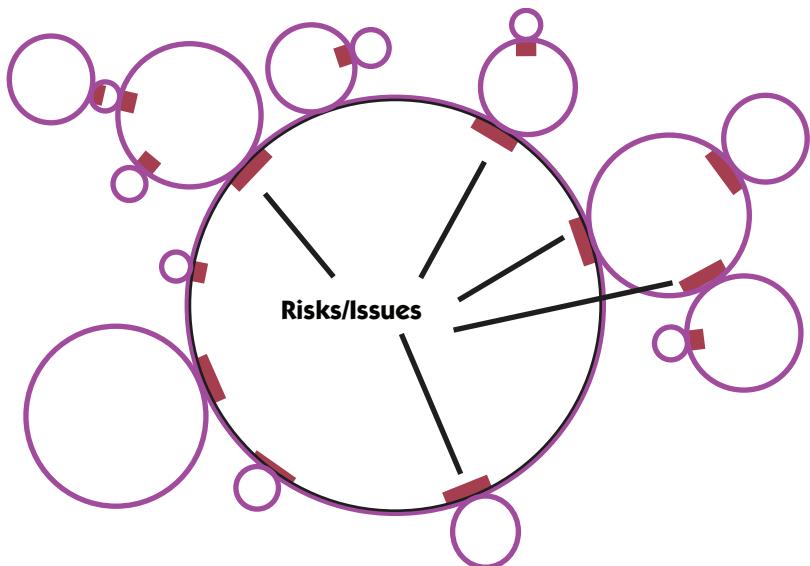


Figure 12-6 Fractal representation of TBTM



Adam Knight (Knight, 2011 and 2013) tackles his organization's large-scale data testing using thread-based testing, which allows testers to work on threads in parallel defined by test charters. Figure 12-6 is a representation of how he explains the benefits of an exploratory testing approach using threads to new testers in his company. They start with a feature area, idea, or risk in the center. As a flaw is discovered, they expand a new set of tests on that discovery. For most discoveries, this is done under the scope of the thread. If discoveries are made that are too large to be considered within the scope of the thread, a new thread is created to explore that area.

That mini exploration will result in a more targeted testing exploration around that feature area and can be represented as a circle off the original. In this way, the testing effort within each thread naturally focuses on the problem and risk areas as they are discovered.

Markus Gärtner (Gärtner, 2011) uses a slightly different tactic, which he calls Pomodoro Testing. He uses shorter sessions of 25 minutes and continues developing his testing mind map during the debriefings. You

can find links to more about Adam Knight's experiences using TBTM and Markus's Pomodoro Testing in the bibliography for Part V.

EXPLORING IN GROUPS

Generally, teams think about exploring as an individual activity, or maybe an activity done by a pair. However, group exploring provides unique opportunities to discover issues or missing features in a new product or major update. We have facilitated exercises in shared exploration, and the results demonstrate the same thing—diversity creates different ideas.

Bernice Niel Ruhland (Ruhland, 2013b) uses this approach once in a while for more complex, riskier areas of a product, especially when time is working against the team. She had the testing team, programmers, and business analysts (BAs) participate in the testing. She recounts:

We used an Excel spreadsheet to define the tests as we had specific test paths based upon coding risks to explore. In some cases critical bugs were fixed before we even finished the testing session. I received positive feedback from this approach. And of course how much or little documentation we provided changed based upon what we were testing and the testers' experience level with the functionality.

Lisa's Story

Spread the Testing Love: Group Hugs!

When our team is preparing a major new release, we sometimes organize “group hugs,” where the whole team, or a subset of it, joins in for testing. Sometimes it’s only testing team members, sometimes the entire product team, or something in between, but it’s always useful. Some people refer to this type of activity as a “bug hunt,” but we feel it’s a positive activity that demonstrates our passion for building quality into our product; our focus isn’t about finding bugs, but about consistency and confidence.

The iOS Group Hug

Recently we released some new features to our iOS app. We asked the team for volunteers for a group hug. Programmers, testers, and marketing folks joined in. We used a shared Google doc for the session, where we noted which device and version each participant was using, as well as already-known problems.

Our team is geographically distributed, so we used a videoconference meeting to communicate during the testing. People in each of our office locations gathered in one room, and remote people joined individually. We find it's helpful to be able to physically talk to each other. For example, if more than one person finds the same bug, we can avoid duplication in reporting it. Also, talking through what we've tried gives other team members ideas for interesting tests.

During the group hug, we found some new bugs, as well as some usability issues. Generally, the feedback was positive, and we were able to release within a few days.

Involving multiple team members in one testing session is expensive. However, we've needed to do it only for major, risky new features where concurrency is critical, and generally one group hug is enough to provide necessary feedback.

The Place for Group Hugs

We build quality into our product with test-driven development (TDD), acceptance-test-driven development (ATDD), and constant pairing and collaboration. We have multiple suites of automated regression tests providing continual feedback in our continuous integration system. We do have automated tests for concurrent changes, but they don't cover every possibility. In addition to the extensive exploratory testing we do on each new feature, the group hugs provide a quick way to get information that we can't get in our normal process.

Consider group exploring (see Figure 12-7) if concurrency is a vital feature of your product, and your automated tests and exploratory testing by individual or paired testers can't cover every scenario. Use these sessions judiciously, and add only as much structure as you need. Sharing a document where everyone records the results is helpful, but in some cases it's too heavyweight. The same goes for preparing charters in advance.

Another option is exploring with a trio (programmer, tester, BA) aimed for fast feedback. Julian Harty calls this "Trinity Testing" (Harty, 2010). The programmer can give feedback on the charters or testing paths based on his or her knowledge of the code, the BA on the business risk. When issues arise from testing, the BA can explain any business impact while the programmer assesses coding risks and time to fix the problem.



Figure 12-7 Exploring in groups

However you choose to run your group session, be prepared to learn from each one, so you can improve your next group testing session and learn even more about the software you're testing.

RECORDING RESULTS FOR EXPLORATORY TEST SESSIONS

It is important to record your exploratory test results and coverage as well as to share your thoughts with others. Some of the reasons to record results or make notes are: it gives you an opportunity to review your results with a peer and have a meaningful conversation about your findings; it allows you to track progress and issues as Christin showed with her TBTM story; it provides the opportunity to review your own results for later testing or if you want to review charters; and last (and our least favorite reason), it demonstrates to management what you've done if problems occur.

Recording can be as simple as taking notes on paper, on a wiki page, as a mind map, in a text document, or on a session sheet. Record issues,

unexpected behavior, or features that seemed to be lacking. Hold a debriefing session to go over what you discovered. You're likely to think of ideas for future sessions as you discuss your most recent one with peers.

Keep your notes brief and simple. If multiple team members are exploring individually or as a group, agree on one format for note taking so you can easily understand each other's findings. Post results in a place where your whole team can see them for reference, and use these notes to improve your testing.

Lisa's Story

When we wrote charters based on personas, we created a chore (a task) in our team's online tracking tool for each charter. This was a good way to track the testing we felt was needed, and we could easily assign ourselves charters and see at a glance who was working on which charter. We could also track which charters were complete, in progress, or not yet started. We tried putting the results of our sessions in these chores, but we didn't find it convenient to refer back to the results this way. Next, we experimented with writing up our test session results on our wiki. Here's an example:

Explore being Paul, the project manager, in an iteration planning meeting.

Browsers: Chrome and Firefox

Initial setup: On Test01, used project 101 populated by the test data fixture.

Observations:

- Deleting stories required a reload.
 - The velocity shown for iterations in the backlog didn't change as I reprioritized stories until I reloaded.
 - Need to do more exploring in projects with custom point scales.
-

Spreadsheets are a simple way to record exploratory test sessions. Adam Knight uses Excel for the top-level charters and exploratory note taking because it provides flexibility to present test results in many forms: text,

graphs, tables, and external links. He then has a set of small macros that provide input boxes to allow very fast input of test notes and tracking of status. He can record status but also can indicate if an issue is “off the charter” and merits the creation of another charter to follow up.

Bernice Niel Ruhland stores her threads in one Excel tab with additional columns for testing notes. It’s like reading a story about the testing. At a glance she knows how many threads are done, in progress, and not started. It also allows her to review any issues quickly.

There are products available that record notes as you do exploratory testing. See the “Tools” section of the bibliography for a link to one of these. Some session-recording tools let you record your keystrokes and the pages you’ve visited so that you can go back and reproduce your steps if you find something worth investigating further. They even let you specify how many pages you want to keep in memory or what type you want to save.

Janet's Story

I decided to try a tool I heard about at a conference called qTrace from QASymphony. (Note: I am not recommending this one over any other.) It is a recording tool for exploratory testing and captures screen shots, notes, and other details. I created a charter for exploring my own website, www.janetgregory.ca:

Explore the blog page

With the search function

To discover if it returns what is expected

I set a time limit of 30 minutes, but with the focus narrowed, I immediately found two issues that I had not noticed when I first checked the search function. The screen prints allowed me to check where I had been without thinking twice. The interesting thing was that I found one issue that bothered me enough that I put in a new request (story)—to search only in blog posts rather than the whole site—and one defect—showing posts authored by Mark (strange, since my name isn’t Mark, and nobody else posts) that weren’t posts, as highlighted in Figure 12-8.

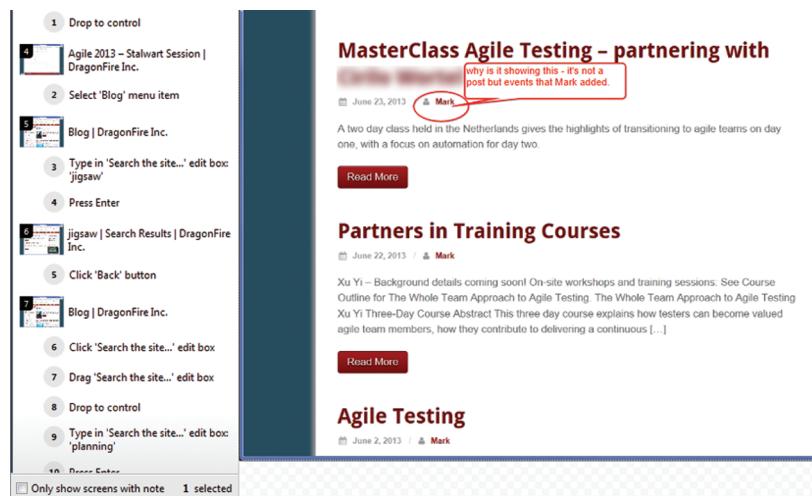


Figure 12-8 Exploratory test session using qTrace

Some recording tools record extensive additional information, such as data about the environment. This helps when you need a reproducible bug so that programmers can write a failing automated test and a fix. You may also use this information to write new user stories and new tests for features you found lacking or to tweak existing features you'd like to improve.

Whatever method you choose to record your results, remember to make them accessible to the whole team. Share the valuable knowledge you gained by exploratory testing.

WHERE EXPLORATORY TESTING FITS INTO AGILE TESTING

We think that exploratory testing is an integral part of agile testing, and it's important to see where it fits in an agile context.

Part IV, “Testing Business Value,” was all about testing for business value. Another way to think of it is testing ideas and assumptions early—before we start coding. That type of testing is about building the right thing. There is another kind of testing designed to answer the question,

“Are we building it right?” We included exploratory testing in Quadrant 3 of the agile testing quadrants because we are usually exploring the workflow to see if we delivered the business value we anticipated. We challenge our assumptions to see what we did not think about earlier when building.

Consider our levels of precision from Chapter 7, “Levels of Precision for Planning”—the product release, the feature, the story, and the task—and what type of exploratory testing might be useful at each level.

- **Product release level:** This is where you would test an integrated product delivery to the integration team, or a release candidate during the end game. This would be a good time to explore dependencies between teams and high-risk workflows and perhaps do tours.
- **Features:** Once all associated stories are “done,” you can explore the complete feature. At this level, good candidates for exploration are feature workflow and interaction with other applications. This might be a good place to try what Lisa calls “group hugs”—more than one person exploring on a charter.
- **Stories:** Once the story meets the expected results—initial coding has been completed and all the automated tests specified before or during coding pass—you can start exploring. Think about the development risks, boundary conditions, more detailed functionality issues, and different variations of formats or states.
- **Tasks:** Exploring at this level would happen during coding. Examples might be programmer exploration on an API, pairing on performance issues with a tester, or maybe even exploring some of the boundary conditions on strings. Consider programmer-tester pairing to create exploratory charters for the code being developed.

Agile teams benefit from continual exploring. When you’re brainstorming new features, you can explore how your different personas might use them. You can explore released software to identify what features may still be missing. Focus your exploratory testing by using charters, tours, heuristics, or session- or thread-based test management. What you learn will help your team improve quality, not only in the impending release,

but as the product evolves in the future. Use the resources in the bibliography for Part V to grow your team's exploratory testing skills.

SUMMARY

Whichever techniques you use, exploratory testing is a powerful way to test new stories and features as they are incrementally delivered.

- Use exploratory testing approaches
 - In conjunction with your automation strategy
 - To give feedback quickly to stakeholders to see if you are building the right thing
 - To prepare for releases with manual regression testing
- Experiment with different methods to create your test charters.
- Experiment with new test charter ideas, such as using personas, tours, or journeys.
- Manage your test charters with SBTM or TBTM or your own approach that meet your needs.
- Practice to build your exploratory testing capabilities.
- Exploratory testing adds value in most, if not all, agile development activities.

Chapter 13

OTHER TYPES OF TESTING

13. Other Types of Testing

- So Many Testing Needs
- Concurrency Testing
- Internationalization and Localization
- Regression Testing Challenges
- User Acceptance Testing
- A/B Testing
- User Experience Testing

In Chapter 8, “Using Models to Help Plan,” we reintroduced the agile testing quadrants. In the section on testing early, we covered Quadrant 1 and 2 tests commonly used by agile teams. In Chapter 12, “Exploratory Testing,” we outlined several approaches to exploratory testing. There’s a seemingly endless list of different types of tests that may be appropriate for your application. For example, some domains require unique approaches, such as the specialized testing techniques needed for business intelligence and data warehousing software, which we will cover along with contexts in Part VII, “What Is Your Context?” In this chapter, we’ll cover a few different types of testing other than functional testing and focus on ones we don’t normally talk about in agile teams but that we think are important.

Remember that the numbering of the Quadrants doesn’t indicate when different types of testing should be done. Q4 tests should be considered as soon as you start discussing a new feature or theme. For example, if there is a constraint that all pages in the application must respond in less than two seconds, make sure everyone on the team is aware of it. Capture that requirement as tests, and keep it in mind during development.

In *Discover to Deliver* (Gottesdiener and Gorman, 2012), Ellen Gottesdiener and Mary Gorman represent the quality attribute dimension in two different areas: operations and development. In Chapter 11 of *Agile*

Testing, we covered some of the basic operational attributes that need testing, such as security, reliability, availability, interoperability, and performance. That is not a complete list, but it gives you an idea of the attributes and constraints we need to consider. The Quadrants can help you brainstorm about the different kinds of testing you need to do on your particular feature or product.

So MANY TESTING NEEDS

Our first Extreme Programming (XP)/agile teams back in the late 1990s and early 2000s were focused on finding out what our customers wanted and then delivering that functionality. We and our teammates tended to be generalizing generalists, and we sometimes neglected areas such as security, reliability, accessibility, performance, and internationalization—at our own peril. It became clear that agile teams must be diligent about all aspects of software quality, even if our business stakeholders don’t mention them.

New types of testing have accompanied the proliferation of platforms, devices, and technology. Many consumer products have software that must be tested. Fitness machines have embedded software. The Tesla automobile has an API! At the same time, outside forces require more test coverage. Security threats to software systems increase all the time, so we must find better ways to test security. As hardware advances in processing speed and graphical display quality, we have to ensure that our products perform well and look good.

Your team may need additional infrastructure to accommodate some types of testing. Let’s say you have a reliability requirement of no more than one failure in 1,000 transactions, or the need to do soak testing. Your team may want a second test environment to be able to drop stable code for reliability testing. You may need extra monitoring tools for watching overnight performance against minimum requirements. If you’re testing an Android app, you’ll need lots of devices in your test lab, and possibly some outside help.

We’ve heard of lists of more than 100 kinds of testing (see the bibliography in Part V, “Investigative Testing,” for an example), although we find

that those lists often contain some duplication. They include the usual ones you would expect to see, such as load testing or browser compatibility testing. However, they also tend to list “types” such as “agile testing,” which really isn’t a type but an approach. When you start thinking about all the different kinds of testing, you might start to wonder when you’ll possibly have the skills, much less the time, to do it all, and it can be overwhelming to new testers. One of the problems we see is the vocabulary misalignment with so many different types. For example, when new people come into the organization and talk about integration tests, everyone else assumes they share the same definition, when perhaps to them, integration testing has a totally different meaning.



We suggest keeping your list of test types as simple as possible and creating a common understanding within your organization. You might start with the types that Wikipedia lists (Wikipedia, 2014l) and differentiate among types, levels, and methods. Bernice Niel Ruhland (Ruhland, 2014) maintains a list of testing types specific to her team, which includes an explanation of how her teams do or do not use a technique. She finds keeping a list helpful for onboarding new testers, and it provides a standard vocabulary shared by everyone in the department. Here’s one example from Bernice’s list of test types:

End-to-end testing is used to test whether the flow of an application or product is performing as designed from start to finish. It typically represents real-world use cases by walking through the steps similar to the end user. The purpose of performing end-to-end tests is to identify system dependencies and to ensure that the correct information is passed between database tables and the application’s modules and feature. We typically perform end-to-end testing for custom-development projects.

Let’s look at a few different types of testing that people often neglect or struggle with on their agile teams. This is not meant to be an all-encompassing list. The goal is to make sure that we remember to do certain important types of testing that often get overlooked in the rush to deliver new features frequently. In Chapter 18, “Agile Testing in the Enterprise,” we will look at how some of these crosscutting concerns affect development organizations with multiple teams.

CONCURRENCY TESTING

Web and mobile applications enhance usability by providing easy ways to make updates, such as dragging and dropping items. When multiple users are looking at the same view of data, we need to verify that when one user makes a change, the other users see that change. If two users update the same piece of data at the same time, who “wins”?

Lisa's Story

The ability for one person to see updates made by another user in real time is an important feature of our product. For our new autosave feature, where changes to an input field should persist as soon as focus moves out of the field, testing concurrent changes was a must. We recognized this as a high-risk area, since users want to feel certain their changes are persisted. We had automated regression tests for concurrent updates but wanted to explore this area in more detail because we weren't feeling confident about the feature quality.

We scheduled what our team calls a “group hug,” where multiple team members test at the same time, as described in Chapter 12, “Exploratory Testing.” We time-boxed this session to one hour and wrote up test charters and scenarios in advance. These charters captured normal use as well as extreme worst-case scenarios.

We typed notes on the bugs we found and other observations in a shared document. Learning about a bug that one person found often inspired me to think up another test to try.

Here's a sampling of issues we noted that would be harder to find while testing individually:

- Changing an epic label when someone else has the epic open, the user who changes sees an error and the label blanks out.
- Starting or moving a story with someone else updating the description causes overwriting and disappearing history.

One interesting discovery was that we couldn't reliably reproduce some problems, which pointed to timing issues. We marked it to explore later to see if there was an issue in the implementation or in the way we tested. It turned out to be the tip of the iceberg of a bad bug that spurred a major code design change.

The group hug confirmed our suspicions that the feature still needed a lot of work. Our team needed to do some redesign, coding changes, and more exploring before we could consider releasing the feature for beta test.

During the group hug, we realized we still had questions about the desired behavior of the autosave feature. This led to further discussions within the

whole team, and within a few days, design improvements and development stories were under way. We knew that there would be more concurrency testing group hugs for this feature.

If your product risks losing updates when simultaneous updates occur or when users need to see updates by other users in real time, concurrency is an important area to think about before coding, but also to cover in your exploratory testing. Solutions for this type of requirement often involve caching updates, which can lead to both performance and transaction integrity issues. Automated regression tests for these scenarios are essential because it is difficult to test them manually; however, timing issues can be subtle and hard to find. Experiment with creative ways to test your features in the same way that customers will use them in production. Take advantage of regular events; for example, check your system's response time while the Olympics are being streamed live.

INTERNATIONALIZATION AND LOCALIZATION

With global markets come global challenges. Many organizations must support multiple languages due to changing business needs, and that creates challenges for agile teams. In traditional projects, the application is built, and then the strings that are to be translated are sent off to experts. The translators have the full context of the application, so the translations are generally correct. In agile teams, we have quick releases with the possibility of releasing to the customer every iteration or perhaps even continuously. Traditional methods do not work in that environment.

All Ýøur ßugs ßeløñg tø Us

Paul Carvalho, a test specialist from Ontario, Canada, shares his experiences with internationalization and localization testing in an agile way.

A Roman walks into a bar, holds up two fingers, and says, “Five beers, please.” Ba-dum-cha. This old joke relies on knowledge of a culture that is not your own. If you already know the answer to the joke, are

interested in finding out, or are looking for new cool ways to bring value to your software systems, internationalization (i18n) and localization (L10n) testing may be right for you.

I'll start with a couple of definitions. i18n lays the groundwork or framework for software and systems to be used in a particular language or region. L10n testing (on a properly internationalized base product) involves checking that the system is translated correctly and looks and works as expected according to some language or culture.

In simpler terms, i18n means the system is *ready* to be translated into another specific language. It's like having a box of Legos with the right pieces to build something—like a fire truck or *Star Wars* set. Meanwhile, L10n means the system *is* translated into another language. Continuing the Lego analogy, you have now built the Lego set into the desired product for someone.

Think about the end product and users' desires. You don't expect a *Star Wars* set to satisfy a medieval castle fan. While there might be some similar pieces, you will need to reexamine the requirements and put the fundamental (i18n) pieces into the box to be successful with different audiences.

This example illustrates one of the advantages to developing internationalized systems in agile ways. An agile team focuses on delivering working software frequently and satisfying the customer through early delivery of valuable software. That is, don't try to solve all your i18n and L10n problems at once. Make different customers happy as you go.

As a team, before you start building anything, get together and come up with specific examples of how the system should look and work for people in a particular target region or country. Remember the Lego example—**BEWARE OF ASSUMPTIONS**. Not all languages are the same everywhere.

Take English, for example. In my travels, I have heard English spoken, spelled, and used in many different ways. English from England (EN-GB) is different from English in the United States (EN-US) is different from English in Australia (EN-AU). If you say your software supports English, I will ask, "Which one(s)?" For example, if a child asks for a *Star Wars* Lego set for his birthday, you'd better be sure you know what that means to him. There is a big difference between an Ewok village set and a *Millennium Falcon*. Guessing wrong may lead down the path to suffering.

Use an iterative approach to building localized systems. For instance, in the first iteration, you may settle on EN-US (if you work for an American company or market) in order to complete a particular feature that provides some value to your target market. When you complete that feature and decide to take on your next target region or audience, always make sure you consult with people who are local to that particular region. You are *loco* (aka crazy, aka nuts, aka . . .) if you don't use locals for localizations.

For example, let's say a U.S. company wants to localize software for its Canadian neighbors (EN-CA). A Canadian could easily tell you that there are more than just spelling differences in some of the words. Yes, the dictionary of terms for labels and outputs is an important piece. However, units of measure require more than just label changes.

Once, before a presentation on i18n and L10n, I used a web-based translation program to translate the following sentence from English to French:

Before: "We need **40,000 lbs** of fuel."

After: "Nous avons besoin de **40.000 kg** de carburant."

Eek! No! Translation requires conversions, too! Let's say it's a hot day and your EN-US weather app says it's 100°F outside. If the user switches to EN-CA, changing *only* the unit label to now say it is 100°C is so many levels of wrong. "People are dying" kind of wrong. We never want to make mistakes that threaten people's lives.

So now you need to add code to your software that takes an input value and makes the appropriate conversion before applying the appropriate unit label. This is where test-driven development allows you to safely make changes to the system to account for new and changing requirements while remembering the requirements on the base features.

Internationalization and localization need to be designed into the system. As with other important quality requirements, we iterate on the design and seek feedback frequently. To be successful, you need access to a specialist on your team, just as with security, performance, or user experience.

Some languages and regions have specific needs that you might never guess on your own, such as capitalization rules or name sorting. Even when translating into European languages that use the same base character set, you need more than just support for the extended ASCII character set.

Does the delivery team need to immerse themselves in a particular culture to be successful? I think it might be fun to bring the team closer to their target audience by getting out of the office and exploring multi-cultural multimedia opportunities to enrich their appreciation for other cultures and regions.

I'm convinced that Hawaiian English needs more exploration and that my employer should support a research expedition. With my luck, I'll probably get a DVD copy of the Elvis movie *Blue Hawaii*.

If your product has a global clientele, do what you can to avoid frustrating customers who use other languages and character sets. Support globalization (g11n), which includes internationalization and localization, with tests that guide development, exploratory testing, and perhaps other types of testing, such as linguistic testing. Internationalization is a constraint that developers must consider as they are coding and incorporate into every story. For example, it would include encoding, formatting, and externalizing strings across the code base. If you are working on legacy code, perhaps you have stories specifically to address some of the i18n requirements.

Localization is the translation piece, not only of the language itself, but often of nuances for specific locales. Terminology needs to be established, but perhaps all of it doesn't need to be done up front. The point when teams break features into stories might be a good time to think of the terms that should be used. Maybe the first story is for an analyst to determine what words will be used for consistency across the product. Frameworks to support localizing and translating can speed up the process, but they don't eliminate the need to verify the translations and suitability for specific cultures and regions.

In Figure 13-1, Lisa shows the terminology for the parts of a donkey harness. To Janet, it seems like a foreign language, but at least now we have common terms and can point to the same piece of harness and mean the same thing. Even writing this book, we had to compromise between EN-US and EN-CA spellings. Since our audience is global, we tried to avoid using slang and colloquialisms.

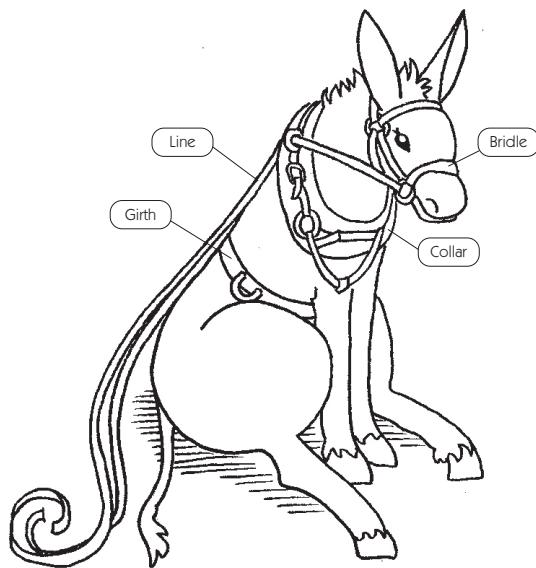


Figure 13-1 Donkey harness vocabulary

Your organizational culture may have a large impact on how you approach localization on your team. As Paul Carvalho mentions in his story, having local support for languages is ideal. However, that may not always be possible, since it is expensive to have many experts available at all times for translations. There are alternative solutions that may allow you to take advantage of fast (or faster) feedback than is usual on a phased-and-gated-style project. Perhaps you can use machine translations, which are getting better all the time. The trade-off is the need for significant post-editing to make sure you catch the errors in the automated conversions.

Another approach might be a hybrid, where agile teams create a “drop” for translators. This drop would include a whole feature, rather than specific strings to translate from each story, and would allow translators to have some context for their translations. If your release cycle is six months long, perhaps a drop of every four weeks could give you fast enough feedback to correct any mistakes before the end game, while minimizing the overhead of translations. Figure 13-2 shows this alternative. It is all about balancing speed with quality, time, and functionality.

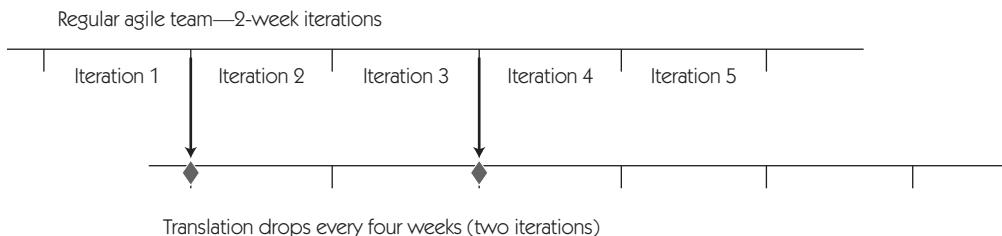


Figure 13-2 Possible schedule for translation drops

Remember, if you are using third-party vendors, have agreed-upon time frames and expectations, and take advantage of automation in file drops for consistency in the process.

It is a different mindset to embrace rapid release cycles, and we encourage teams to think about how they can make changes to their process to get fast feedback. Perhaps, just perhaps, the local experts exist on some of your global teams.

REGRESSION TESTING CHALLENGES

First, we give a quick definition of what we mean by regression testing because it can be a contentious term. To us, regression tests are those tests that run regularly to give you confidence that changes made to the code do not affect existing functionality unexpectedly. We believe that automated checks can do this with the fastest feedback. At a minimum, these should run nightly.



Now that many teams release once a week, several times per week, or several times per day, regression testing is an even bigger challenge. Even if you automate your regression tests, they may not all run fast enough to complete in time for the next release.

Many companies faced with this problem turn to “testing in production.” Seth Eliot has written and presented extensively on this subject. He defines testing in production this way (Eliot, 2012):

Testing in production (TiP) is a set of software testing methodologies that utilizes real users and production environments in a way that both

leverages the diversity of production, while mitigating risks to end users. By leveraging the diversity of production we are able to exercise code paths and use cases that we were unable to achieve in our test lab, or did not anticipate in our test planning.

Lisa's team has good coverage from automated test suites running in the continuous integration system at all levels: unit, functional, and user interface (UI). They also spend lots of time doing exploratory testing, but it's hard to cover every scenario. For new major versions, they use a TiP approach. They enable the new version for a small percentage of users and monitor production logs to see if users are experiencing errors. As they identify new defects, they fix them as needed. They have a rollback plan to disable the feature if there are unacceptable results. When the new features appear to be stable, the team enables them for more users, continuing to watch logs carefully, until all users are able to use them.

As much as we automate regression tests, there may be tests that are difficult to automate in a way that provides proper feedback, and other tests that are too costly to automate. It's also a challenge to do manual regression tests for quality attributes such as look and feel when releasing so frequently. The testers on Lisa's team created wiki pages with manual regression test checklists for different areas of the product based on risk. Often, programmers use the checklists to do the manual regression testing, or at least help with it. The team regularly audits the checklists to see if any tests can be or have been automated. They keep the lists as concise as possible, focused on the high-risk areas. In Chapter 12, “Exploratory Testing,” there were a couple of examples of managing regression testing with session-based test management or thread-based test management if you do not have automation. How you manage your regression suite will depend on your context, the risks within your system, and how often you release to production.

USER ACCEPTANCE TESTING

User acceptance testing (UAT) is part of Quadrant 3, business-facing tests that critique the product. We describe user acceptance testing as “making sure the actual users can do their job.” UAT may be performed by product managers, but preferably it's done by the actual end users of

the product. As we explained in *Agile Testing*, UAT is often done as part of a post-development testing cycle (pp. 467–68), but this does not mean it has to be left to the end game. Janet encourages companies she works with to think of ways to bring it earlier into the development cycle.

Janet's Story

I started with a team that was on three-month delivery cycles, which should have meant that they released to the customer every three months. However, the UAT testing took another six weeks, so new features weren't actually in production for almost six months. I found out that the reason for such a long UAT cycle was that the person (I'll call her Betty) doing the testing had to do it in addition to her own job. She had to "fit it in" around her regular duties, and six weeks was how long it took.

I suggested that we have Betty sit with the team at the end of each two-week iteration and play with the new features we were developing. I paired with her for a while to show her the features and then let her be. At the end of the three months when we were ready to deliver the new features, Betty asked for only three weeks (instead of the usual six) for UAT.

This was a substantial improvement, but I wanted it to be even better. We got her a workstation in the team work area, and Betty started coming in every Friday afternoon. She gained confidence in our work and what we were delivering. At the end of that release, we included one full dedicated day of UAT and were able to put the release into production within the three-month delivery cycle.

Lisa's current team develops a software-as-a-service (SaaS) product that is also used internally by the entire company. This provides a great opportunity to release new features for internal-only beta and get feedback from actual users who happen to be in the same company. Problems with real-life use are identified and fixed before the new features are made available to paying customers.

Understand your customers, your real users, and brainstorm ways to get them to use the system to make sure they can do their job and use the system appropriately. Customers are the ultimate judges of software value. End users are probably the best people to critique your product.

A/B TESTING

A/B or split testing is often used in lean startup products or in existing products that are changing their look. It is a different type of testing in that it validates a business idea, so in some ways it is a Q2 type of test. However, it is done in production by real customers, so it's really a way of critiquing the product.

The idea is to develop two distinct implementations, each representing a different hypothesis about user behavior, and put them out for production customers to use. For example, you can move UI elements around or change the steps of a UI wizard. The company monitors statistics on which customers “click through” and which leave right away. In this way, companies can base decisions on real results and can improve their applications based on continued A/B experiments. When appropriately done, A/B testing can help companies make decisions about everything from user experience (UX) design to pricing plans.

Small Changes Can Have Big Impacts

Toby Sinclair, a tester in the UK, shares his experiences at a retail company that did extensive A/B testing.

Whilst I worked with a large online UK retailer in 2011, they introduced the concept of A/B testing. The usability and customer experience teams wanted to have more confidence in the design decisions being made, and they saw A/B testing as a way to do this.

The initial tool used was Campaign Optimizer, as it integrated with our existing Oracle ATG e-commerce platform. It allowed the business to set up A/B tests without code deployments. This meant they could easily switch tests on and off without contacting the development teams. The tool also provides the administration facility (reporting, switching tests on/off, etc.). Some initial development and integration testing was required to get it up and running.

The first A/B tests the team implemented were very simple, for example, a 50/50 split between these two choices:

- **Existing version:** 20 items on shopping gallery page by default
- **The challenger:** 50 items on shopping gallery page by default

The winner was decided based upon a number of metrics. These included

- Pages viewed by a user
- Particular items on a page viewed by a user
- Products added to a shopping cart by a user
- Purchases made by a user

Deciding how to split your tests and how long to run them is quite an art. Some of the factors include:

- How many visits did you get to your site? You need a high count in order to have confidence in your results.
- How experimental is the challenger? You may want to send only a small number of customers to the challenger if the impact is unknown.

Generally most of our A/B tests ran for two weeks, as we found that gave us a good statistical result based upon the average customer visits. To validate the results from the A/B tests, we also compared information from our analytics partners (Coremetrics and Speed-Trap). This ensured that we had full confidence in the winning version.

We also developed the capability to run tests against specific customer segments. This made the results obtained even richer and provided further insights to the targeted content team.

Today, every day, there are tests running across the website providing rich information to the teams about how best to design the website. The capabilities of and focus on A/B testing have grown considerably over the past few years. There is now even a separate A/B test team that recruits people with both a technical and marketing background.

A/B testing is definitely not limited to agile projects, but it fits agile values well. The goal of A/B testing is to identify which changes to your website will have the greatest impact. This type of testing is all about iterating with fast feedback to select a design or achieve and maintain other quality characteristics that help the business achieve its goals. If you think your team might benefit from A/B testing, check the links in the bibliography for Part V to learn more.

USER EXPERIENCE TESTING

User experience (UX) designers have many ways to get feedback on functional website design and designs in progress. They use methods commonly found in industrial design and anthropology to test designs and design concepts before producing anything on screen or even on paper.

User Research

Drew McKinney, a product designer on Lisa's current team, shares some of his experiences with user research and user experience testing.

One of the projects I led was to develop a project planning and management system for Disney Animation Studios artists and technical staff. My team conducted ethnographic studies with software developers, technical directors, technology managers, and animation technologists to understand their collaborative work processes. In a process known as contextual inquiry, the designer or researcher observes the user perform day-to-day activities and discusses them with the user while taking part (Wikipedia, 2014b).

In our case, these predesign sessions revealed several problems with the artist software delivery system and technical project management. Ongoing communication issues coupled with a poorly designed software indexing system resulted in numerous headaches for both technical and artist staffs. As a result of these early predesign exercises, it was easier for our team to design an appropriate solution to Disney's software (and people) management problem.

Prototype Testing and Paper Prototypes

Design testing can be a lengthy process if done incorrectly. Testing too late in the process can reveal fundamental flaws with a new system's design, requiring costly redesigns and rewrites. To provide early feedback on product designs, UX designers use a number of prototyping methods throughout the product development process. These prototypes can range from full-color printouts to simple sketches.

The easiest way to achieve quick usability feedback is also the cheapest: paper prototypes. Paper prototypes are paper representations of interfaces. On a different project, my team designed an iPhone app for a weight-loss program. Again, we used early paper prototypes to see how users would respond to an app concept. We noticed that

users were confused about the meaning of representations on a map, so we improved the graphical design to make the meaning clearer. We observed that the iPhone users were more attracted to applications that provided a fun and fresh experience, so we designed their system to accommodate that need.

Sometimes paper prototypes are not enough, and a more functional example needs to be used. In another example from Disney, our team was tasked with building a collaborative canvas interface to allow senior animators to critique remotely located artists, a process that still relied on fax machines and phone calls. This was a clear opportunity for a functional prototype: a working example that artists could use as if it were final. Rather than build a prototype, we used off-the-shelf hardware and software to accomplish this. We attached a Teradici remote workstation to a Cintiq monitor and placed a MacBook above the workspace with a FaceTime call to the other artist. While by no means a final product, this quick prototype allowed us to vet the design to understand the benefits and limitations of such a system. Two Disney animators worked together remotely to test a design prototype. Figure 13-3 is a representation of this collaboration.

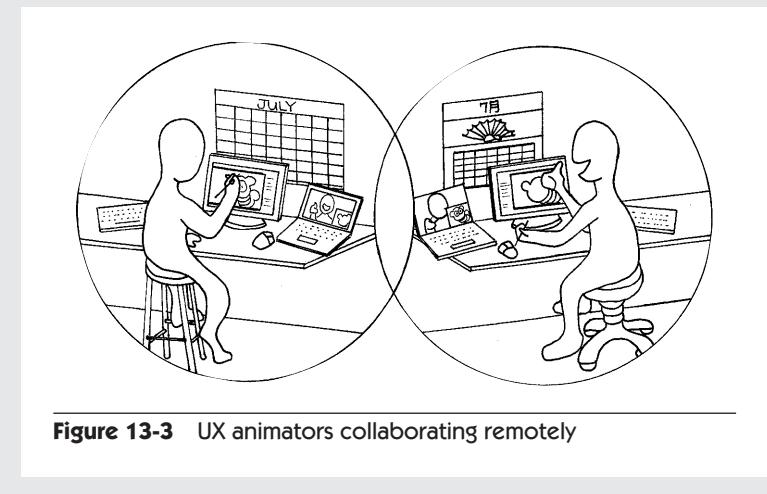


Figure 13-3 UX animators collaborating remotely

User experience testing is an example of a type of testing that can fit into more than one of the agile testing quadrants. You can get feedback from users before any coding is started, using paper prototypes and other techniques such as the ones Drew describes. This is a form of Quadrant 2 testing, creating customer-facing examples and information

that will guide development. You can also do usability testing after coding is complete or monitor the product already in production to learn whether the current design and functionality are adequate. Those are examples of Quadrant 3 activities, evaluating the software from a business and user perspective. Feedback from this may result in new features and stories to be done in the future.

Nordstrom Labs recorded its in-store innovation testing efforts for an iPad app that would help customers choose eyeglass frames (Nordstrom, 2011). The video shows testing activities ongoing throughout iterations that lasted minutes rather than days. Designers and testers on Lisa's team have done usability testing with both internal company users of a product and people outside the company, taking advantage of user group meet-ups. Look for ways to test with real end users rather than speculate about how they will use a particular feature.

Sit with your product's existing users and see where they struggle. Take your new design to your local coffee shop and see what people think of it. Involve current and desired customers early and often. Chapter 20, "Agile Testing for Mobile and Embedded Systems," has a bit more on user experience but focuses more on how it relates to mobile apps. Check the Part V bibliography for more links on these different types of testing.

SUMMARY

Don't repeat the mistakes of our early XP/agile teams and focus exclusively on functional testing. In this chapter, we discussed some of the types of testing that are outside the scope of what is generally known as functional tests. Many of these tests can be done with an exploratory approach.

- Use the Quadrants to think about all the different types of testing your product requires.
- Talk with your business stakeholders to learn their expectations for attributes such as stability, performance, security, usability, and other "ilities."
- Concurrency testing is key for products whose users may update the same data simultaneously. Use both automated and

exploratory tests to ensure that updates are reflected correctly and in a timely manner.

- Internationalization and localization testing requires looking at cultural differences as well as languages and character sets. Specialists in this field are needed, just as many software products require security, performance, or UX testing experts.
- Testing (monitoring) in production is one approach to finding defects that are missed by checking and exploring during development.
- Completing user acceptance testing during development, rather than after, shortens the UAT cycle needed during the prerelease end game.
- A/B testing is one way to get fast feedback from production users about aspects of the application design, using a series of experiments, each one building on what was learned from the last.
- Usability testing can be done simply and productively with paper prototypes and conversations with users to help your team refine your designs and features.

Part VI

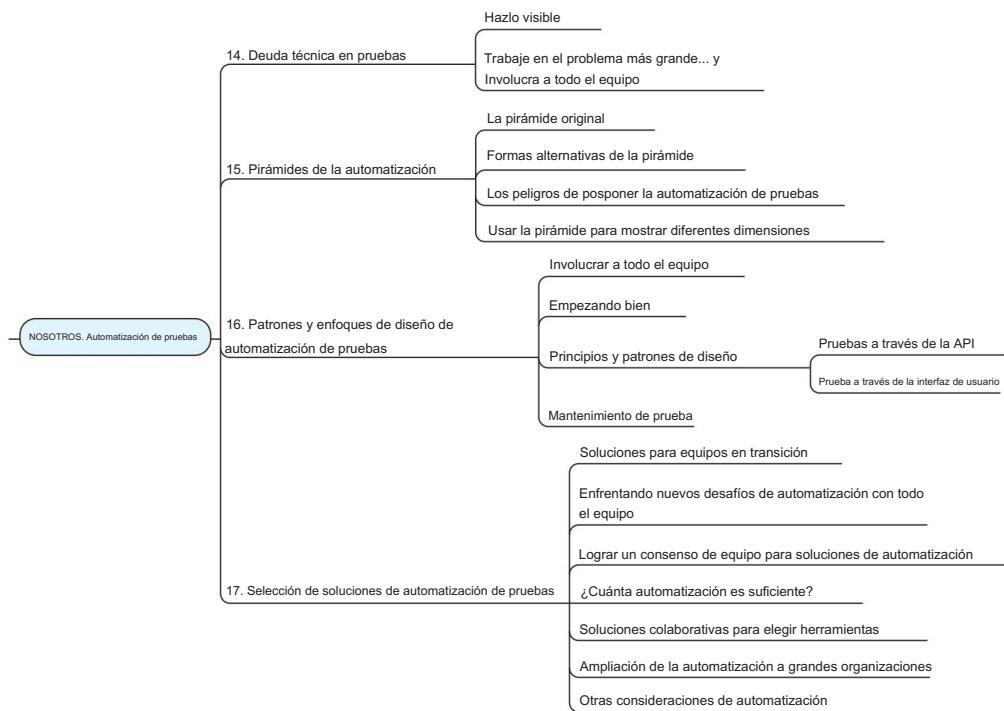
TEST AUTOMATION

Test automation is necessary. We need to automate to manage technical debt, make time for critical testing activities such as exploratory testing, and guide development with customer-facing examples. Automated regression tests tell us quickly if we've broken existing production code. As well, our automated tests provide excellent living documentation for our application.

Test automation is hard. Some types of automation are fairly painless and can even be rewarding once you get past the “hump of pain” of learning how. Some present ongoing challenges, such as testing JavaScript and logic-heavy user interface pages and experiencing sporadic failures due to timing or automatic browser upgrades.

Agile Testing included a large section about automation. To learn more about the reasons to automate tests, barriers that get in the way of automation, and how to put together an agile test automation strategy based on the test automation pyramid and agile testing quadrants, please refer to it.

Here in Part VI, we will explore more ways that testers and agile teams can succeed over the long term in creating maintainable test automation. Automated tests can help teams achieve a sustainable pace with manageable levels of technical debt. But this works only if your tests provide quick feedback at a low-enough cost. We'll look at alternative interpretations of the test automation pyramid and how those can inspire us to do more experiments involving the whole team in solving automation problems.



- Capítulo 14, “Deuda técnica en las pruebas” ■
- Capítulo 15, “Pirámides de automatización” ■
- Capítulo 16, “Patrones y enfoques de diseño de automatización de pruebas” ■
- Capítulo 17, “Selección de soluciones de automatización de pruebas” ■

Capítulo 14

Deuda técnica en pruebas

[14. Deuda técnica en pruebas](#)

Hazlo visible

Trabaje en el problema más grande... y
Involucra a todo el equipo

Ward Cunningham acuñó el término deuda técnica (Cunningham, 2009) para representar la entrega apresurada de una característica o historia de usuario sin tener que volver a refactorizar el código para expresar lo aprendido. Podemos incurrir en una deuda técnica abrumadora en nuestro código de prueba automatizado, así como en nuestro código de producción. Los equipos a menudo cometen errores en sus intentos iniciales de automatización de pruebas funcionales, en parte porque los marcos y controladores actuales facilitan la automatización rápida de las pruebas. Sin embargo, si no prestamos tanta atención a los principios y patrones de diseño del código con el código de prueba automatizado como lo hacemos (idealmente) con el código de producción, mantener los scripts de prueba puede convertirse en una carga. Es complicado refactorizar el código de prueba automatizado; ¿El fallo de una prueba es un error en la refactorización o un error de regresión real en el software? Una de las primeras referencias a la deuda técnica en las pruebas que encontramos es la publicación del blog de Markus Gärtnert “Deuda técnica aplicada a las pruebas” (Gärtnert, 2009).

Si la automatización de pruebas es inadecuada, o si las pruebas automatizadas requieren demasiado tiempo para su mantenimiento, hay menos tiempo para otras actividades de prueba cruciales. Si el tiempo es corto, los equipos ceden a la tentación y se saltan las actividades de prueba de los cuadrantes 3 y 4. (Consulte el Capítulo 8, “Uso de modelos para ayudar a planificar”, para obtener más información sobre los cuadrantes de pruebas ágiles). Si el equipo siente presión para cumplir con un fecha límite y las pruebas automatizadas están fallando, pueden racionalizar que necesitan arreglar el código de producción, pero arreglar las pruebas puede posponerse para más tarde. Cuando los evaluadores constantemente se ponen al día probando el código que se verificó en una iteración anterior, no tendrán tiempo para colaborar con los clientes para obtener ejemplos y especificar pruebas comerciales para guiar el desarrollo de las historias actuales. Esto se convierte en un círculo vicioso; no hay tiempo, por lo que nos saltamos actividades

da como resultado más fallas de regresión y menos probabilidad de que desarrollemos las funciones adecuadas para el cliente. Cuando las pruebas y la codificación se valoran por igual, podemos evitar este problema.

Puede hacer un trabajo bastante bueno automatizando pruebas en los diferentes niveles de la pirámide de automatización de pruebas (consulte el Capítulo 15, "Pirámides de automatización") y aun así descubrir que no fue suficiente. A medida que cambia un producto, algunos equipos descubren que se vuelve más difícil actualizar las pruebas de regresión automatizadas en consecuencia. Al igual que con el código de producción, el código de prueba puede ser difícil de leer y comprender o contener duplicaciones innecesarias. Necesita los mismos cuidados y alimentación que el código de producción.

Veamos formas de gestionar la deuda técnica en sus pruebas.

Hazlo visible

Como ocurre con muchos impedimentos que enfrentan los equipos ágiles, puede comenzar a reducir la deuda técnica de prueba de su equipo a un nivel manejable dándole primero visibilidad. Para cada historia en el trabajo pendiente que implique cambiar el código, piense en qué pruebas automatizadas existentes podrían verse afectadas. Si es necesario, eche un vistazo rápido a las pruebas automatizadas para tener una idea del posible esfuerzo necesario. Asegúrese de que la estimación refleje el tiempo necesario para actualizar las pruebas y escriba una tarjeta de tarea para hacerlo (consulte la Figura 14-1).



Figura 14-1 Ejemplo de tarea para abordar pruebas automatizadas existentes

Escriba historias para cualquier refactorización de código de prueba importante que necesite realizar, tal como lo haría para el código de producción.

Realice un seguimiento de cuánto tiempo dedica al mantenimiento de las pruebas y hágalo visible. Vea dónde pasa su tiempo. ¿Se trata de pruebas de depuración que fallan o de actualización de pruebas que dan resultados falsos o que de otro modo estaban mal diseñadas? Quizás pueda escribir tarjetas de tareas o historias para cada falla que deba investigarse y para mantener o refactorizar guiones de prueba para crear visibilidad.

De manera similar, asegúrese de que las estimaciones y tareas reflejen la cantidad de esfuerzo que se dedica a las pruebas de regresión manual de cada versión. Recuerde que todo el equipo es responsable de todas las actividades de prueba, incluidas las pruebas de regresión. Es común que los programadores y otros miembros del equipo ayuden con las pruebas de regresión manual. A esto lo llamamos "compartir el dolor" y permite al equipo tomar mejores decisiones sobre cómo seguir adelante.

Reducción de la deuda por defectos: tolerancia cero

Augusto Evangelista comparte cómo su equipo redujo su deuda por defectos y ganó más disfrute.

Descargo de responsabilidad: Esto no funcionará para todos; todo lo que digo es: "Funcionó en mi contexto", y puede intentar usarlo bajo su propio riesgo.

Un día, hace unos años, tuve una conversación con un inspirador hombre; él era mi CTO en ese momento. Habló de tolerancia cero hacia errores y lo beneficioso que es eliminar las molestias de la gestión de errores en el desarrollo de software. Escuché, pero en ese momento No entendí el concepto completamente; Nunca había visto o imaginado una situación en la que un equipo de desarrollo podría producir cero o para eso importa incluso cerca de cero errores. Desde entonces han pasado algunos años. y he trabajado duro en su visión. Hoy puedo decir que tenía razón; un equipo de entrega puede ofrecer valor sin errores conocidos. Incluso aprendí que el equipo del proyecto no tenga que pasar la mitad de su tiempo jugando con una herramienta de gestión de errores para archivar, priorizar y cambiar los residuos.

Permitanme darles un poco de contexto sobre mi proyecto y mis prácticas actuales. Contamos con dos equipos ágiles interfuncionales ubicados, cada uno con cuatro desarrolladores, un evaluador, un analista de negocios, un propietario de producto, un chico de DevOps y un líder de equipo que funciona como facilitador de kanban, para un total de 18 personas.

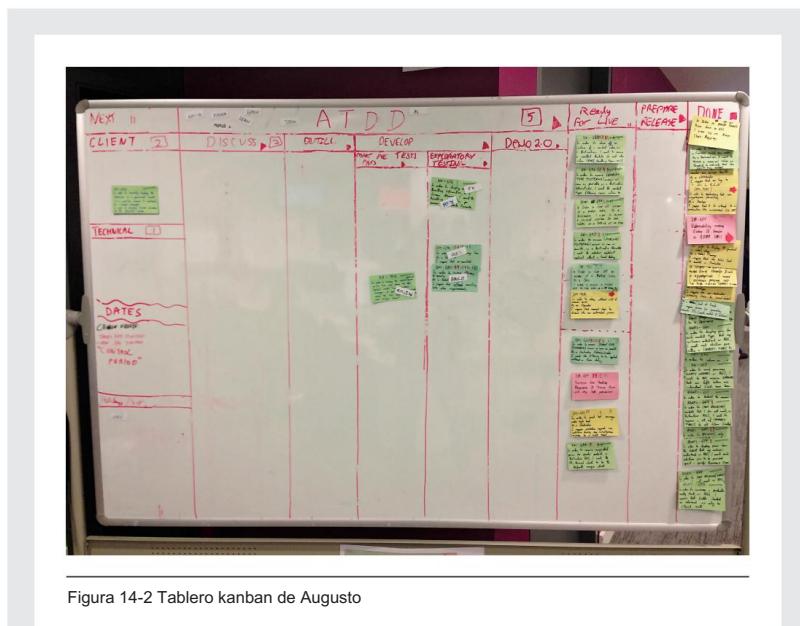


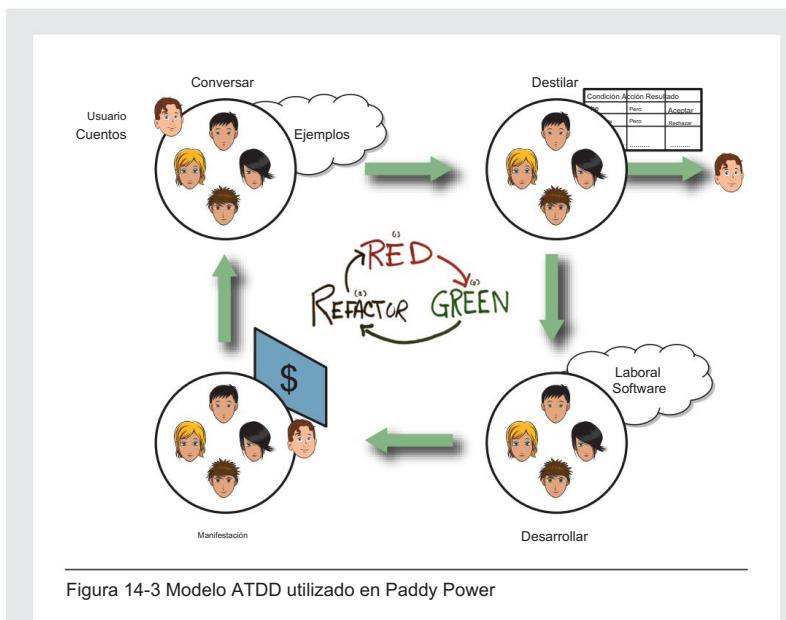
Figura 14-2 Tablero kanban de Augusto

Usamos kanban para visualizar nuestro proceso y tenemos la capacidad de entregar cada vez que completamos una historia de usuario si así lo queremos. Nuestro Kanban El tablero (ver Figura 14-2) visualiza nuestro proceso. En el centro superior, hay una sección "ATDD" (desarrollo basado en pruebas de aceptación) con encabezados de columna tomados prestados del trabajo de Elisabeth Hendrickson: "Desmaldecir", "Destilar", "Desarrollar" y "Demostrar".

Mantenemos nuestras historias de usuario pequeñas, para poder entregarlas en menos de tres días. Las historias son cortes verticales, lo que significa que no hay integración. Se requiere entre equipos. Cada equipo trabaja en la base del código completo, que abarca múltiples aplicaciones.

Usamos un híbrido de ATDD y especificación por ejemplo (SBE) [autores nota: consulte el Capítulo 11, "Obtención de ejemplos", para obtener más información sobre ellos], que Lo he desarrollado específicamente para adaptarse a nuestro contexto. Revisión de código de programadores cada línea de código antes de cada implementación; también practican algún par programación y desarrollo basado en pruebas (TDD) (Hendrickson, 2008).

Automatizamos cerca del 100% de nuestras pruebas de aceptación, utilizando jBehave y Tucídides. Consulte la Figura 14-3 para obtener una imagen de nuestro ciclo ATDD. Es basado en un modelo de James Shore, con cambios sugeridos por Grigori Melnick, Brian Marick y Elisabeth Hendrickson y el concepto SBE de Gojko Adzic. También contiene la imagen "Rojo, Verde, Refactor". de (Ottie Águeda Langr, 2009).



Contamos con un proceso de compilación e implementación de "estilo plataforma bajo demanda" desarrollado internamente que ejecuta todas nuestras pruebas funcionales, así como pruebas de rendimiento/carga que verifican las variaciones con respecto a una línea de base conocida. Cada prueba automatizada se ejecuta después de cada envío al troncal de la tubería.

Realizamos pruebas exploratorias en una historia una vez que pasan todas las pruebas de aceptación automatizadas. Una vez que se completan las pruebas exploratorias, hacemos una demostración a nuestros propietarios de productos, quienes posteriormente realizan pruebas de aceptación del usuario (UAT) antes de aceptar la historia.

Si se encuentra un error durante las pruebas exploratorias o las pruebas de aceptación del usuario, los programadores que trabajaron en esa historia de usuario abandonan todo lo demás que podría estar haciendo y corren el error de inmediato. La tarjeta progritará si no se soluciona un error válido. La construcción siempre debe ser ecológica. Si falla (se vuelve rojo), el desarrollador que causó la inestabilidad descarta todo lo demás y soluciona el problema de inmediato.

Cuando un programador corrige un error, escribe pruebas automatizadas que cubren su ruta. Quizás hayas notado que mencioné errores. Sí, por supuesto, somos humanos y cometemos errores. Esto no significa que debamos celebrar los errores y hacerlos visibles registrándolos en herramientas de seguimiento de errores o que debamos mantenerlos junto con los desechos almacenados en ellos, cuando el pobre insecto tiene la vida de un desafortunado mariposa.

Cuando encuentro un error mientras hago pruebas exploratorias, simplemente voy al desarrollador y diga de manera amigable: "Creo que podría haber un problema; Ven a mi escritorio y te lo mostraré". Si el programador necesita ir casa y no puedo arreglarlo inmediatamente, no hay problema; Pego una nota adhesiva roja en la tarjeta de historia de usuario en el tablero kanban físico con dos palabras describiendo el error como recordatorio. Esa tarjeta no irá a ninguna parte hasta el error está arreglado y enterrado. La nota adhesiva roja va a la papelera. después de la muerte del insecto.

El enfoque de desarrollo que seguimos nos permite tener una buena comprensión del valor empresarial que ofrecemos porque tenemos discusiones grupales para derivar ejemplos para cada historia de usuario. Cuando usted agregue un grupo de excelentes desarrolladores que siguen una buena ingeniería prácticas a esa mezcla, descubres que los errores que descubres cuando explorar el software son muy pocos. Cuando actúas sobre ellos inmediatamente, sin excusas, los errores realmente no te molestan.

En nuestra situación, registrar y gestionar errores sería simplemente un desperdicio. Nosotros todos vivimos felices y sin errores Ping-pong entre programadores y probadores, sin priorización de errores o reuniones de clasificación, sin estadísticas de errores, y no es necesario utilizar tendencias de errores para identificar las fechas de lanzamiento del producto. y adivina ¿qué? Entregamos software que no tiene errores conocidos y deleita a nuestros clientes.

El equipo de Augusto ha encontrado formas de crear lo que sus clientes quieren y de evitar que los defectos lleguen al cliente. Parte de su estrategia es poner una nota adhesiva roja en la tarjeta de la historia cada vez que encuentran un error mientras prueban la historia, lo que brinda visibilidad. Todos pueden ver de un vistazo cuando hay un error que bloquea la historia. Haga visibles tanto las actividades de prueba como los defectos en su sistema de seguimiento de proyectos, ya sea un tablero físico en la pared con fichas o un tablero virtual en línea. Con esta visibilidad, todos pueden ver si el equipo está asumiendo demasiado trabajo y se está quedando sin tiempo para realizar pruebas dentro de una iteración. A veces es inevitable llevar las tareas de prueba a la siguiente iteración, pero puede ser una señal de alerta de que el equipo ha acumulado demasiada deuda técnica de prueba, si no deuda técnica general.

La historia de Janet

En una organización con la que trabajé, teníamos un gran conjunto de sistemas automatizados. pruebas de regresión, pero empezaron a fallar y a tardar cada vez más. Nosotros investigó y descubrió que la deuda técnica en ese conjunto de pruebas era

enorme. Las pruebas no sólo eran frágiles y complejas, sino que también muchos que probaron lo mismo. Eso podría ser manejable en una cascada. proceso, pero se convierte en una carga en ágil. Nadie había estado mirando o prestando atención a los diseños de prueba. Medimos cuánto tiempo pasó en mantenimiento y tomó la decisión de que una persona pasara medio día durante un mes para refactorizar para eliminar duplicaciones y simplificar las pruebas. Este enfoque redujo los costos de mantenimiento; no sólo fue más fácil de encontrar y solucionó cualquier problema, pero también redujo el tiempo para ejecutar las pruebas y mejoró la calidad y cobertura de las pruebas. Al hacer visible el problema, el El equipo (incluido el propietario del producto) pudo elegir cómo arreglarlo.

Trabaje en el problema más grande y consiga el todo Equipo involucrado

Una vez que tenga señales visuales de deuda técnica, como tareas de prueba que no se completaron al final de la iteración, compilaciones de integración continua (CI) rojas (rotas) o demasiado tiempo dedicado al mantenimiento de las pruebas, tome medidas. La deuda técnica de prueba es algo que todo el equipo debe gestionar. Recuerde que todo el equipo es responsable de todas las actividades de prueba, desde convertir ejemplos en pruebas automatizadas que guíen el desarrollo hasta pruebas exploratorias y comprobaciones de regresión.

Las retrospectivas de equipo son una oportunidad perfecta para sacar a relucir áreas de deuda técnica de pruebas que deben reembolsarse. El equipo puede discutir los factores que contribuyen a cada fuente de deuda y utilizar la votación por puntos o una técnica similar para identificar el problema más grande que debe abordarse primero. Piense en pequeños experimentos que podría intentar solucionar ese problema.

Por ejemplo, digamos que sus mayores problemas con las pruebas son que las pruebas de regresión automatizadas tardan demasiado en ejecutarse, las fallas tardan mucho en identificarse y las pruebas constantemente obtienen falsos pases o fallas (Janet las llama “pruebas inestables”). Una forma de acelerar las compilaciones es agregando esclavos de compilación en una máquina virtual y múltiples conjuntos de pruebas en paralelo. Las pruebas inestables se podrían separar en un conjunto separado y se podrían volver a intentar automáticamente una vez antes de investigar la falla. Se podrían medir para ver con qué frecuencia fallan o aprueban, y luego se podrían tomar decisiones sobre si corregirlos o reescribirlos para que eventualmente puedan regresar al conjunto de regresión regular. Tal vez ya no valga la pena mantener algunas pruebas, o cubren el mismo terreno que las pruebas automatizadas de nivel inferior.

pruebas de regresión y deben eliminarse. Quizás se podrían eliminar algunos datos de prueba no utilizados para que la configuración de los datos para las pruebas sea más rápida. Incluso podría ser un proceso más complejo de revisión de pruebas para ver si algunas pueden eliminarse o si las pruebas requieren refactorización.

Mientras su equipo piensa en posibles formas de abordar la deuda técnica de las pruebas, haga visibles las posibles soluciones, tal como hizo con los problemas. Escriba y calcule historias de actividades para comenzar a eliminar su deuda técnica de prueba. El equipo de Lisa utiliza un proyecto separado para rastrear historias y tareas (tareas) para cosas como reducir fallas laborales falsas de CI. Celebran una breve reunión cada semana y las historias y tareas de alta prioridad se trasladan al trabajo pendiente activo del equipo según sea necesario. Si varios experimentos no logran mejorar el problema, un pequeño grupo multifuncional propone más soluciones y los individuos asumen la responsabilidad de probar otras diferentes. Revisan los temas en los que han elegido trabajar durante cada retrospectiva hasta que ya no frenan al equipo.

Reducción de la deuda técnica de pruebas mediante la colaboración

cris george , un probador de Reitano cuenta esta historia de cómo probador y aa fuerzas para hacer lo que muchos pensamiento era "imposible."

La tarea "imposible" era transformar un conjunto de pruebas heredado desde un Tasa de fracaso del 10 % (¡pero no siempre el mismo 10 %!) en 8.000 pruebas que Tomó más de 12 horas ejecutarlo y pasar al 100% en mucho menos tiempo. Hubo muchos intentos de solucionarlo con distintos grados de éxito, pero otros trabajos de proyecto normalmente interrumpían cualquier progreso.

Estas pruebas habían adquirido notoriedad entre los equipos de entrega, y la estimación para arreglar el conjunto de pruebas fue del orden de varios meses. Desafortunadamente, las pruebas no se pudieron eliminar simplemente porque el código que cubrieron fue utilizado por varios productos, uno de los cuales se estaba trasladando a una estrategia de lanzamiento frecuente, y estas pruebas estaban bloqueando su progreso.

Jeff, uno de los desarrolladores, y yo decidimos echarle un vistazo. en las pruebas. Teníamos experiencia en mejorar conjuntos de pruebas y pensamos que Podríamos aplicar nuestras experiencias a este. Nos dieron dos semanas para ver cuánto podemos hacer. Intentos anteriores de arreglar las pruebas involucradas. reescrituras importantes del marco de prueba subyacente. El enfoque que nosotros quería tomar era algo menos invasivo y no implicaba una reescritura.

Comenzamos con cada uno de nosotros tomando una pequeña sección de las pruebas fallidas, analizando y categorizando las fallas, y luego arreglando las fáciles. Mayoría

de las pruebas utilizaron bases de datos y copias de seguridad de bases de datos, y resultó que muchas de las fallas se debieron a problemas de configuración. Estos se solucionaron con bastante facilidad.

¡En el transcurso de dos días, solucionamos aproximadamente el 50% de las fallas!

¡Otros dos días y seguramente habremos terminado!

Mientras investigamos los problemas restantes, descubrimos que las pruebas creaban muchas bases de datos pero no se aclaraban por sí solas.

Ambos hemos trabajado con SQL Server el tiempo suficiente para darnos cuenta de que cuantas más bases de datos tenga en un servidor, más lento se vuelve. Entonces, incluso si una décima parte del número total de pruebas creara una base de datos cada una, esto realmente dañaría el rendimiento del servidor y agregaría un tiempo significativo a la ejecución de la prueba.

Si esto hubiera dependido de mí, habría puesto los comandos de caída de la base de datos en el desmontaje de prueba. Afortunadamente, no dependió de mí y, de hecho, trabajamos juntos en el problema. A Jeff se le ocurrió la idea de aplicar el patrón Dispose a este problema, que se utiliza para manejar la limpieza de recursos. Ya creamos un objeto de base de datos cuando restauramos una base de datos, por lo que simplemente hicimos este objeto desecharlo. El método de eliminación luego llamó al comando soltar. ¡Al hacer esto, el autor de la prueba no necesita acordarse de eliminar la base de datos!

Durante el resto de la semana, Jeff y yo aplicamos este patrón a cada prueba. Nuestra razón fundamental para hacer esto fue reducir los problemas que conocíamos. Una vez que eliminemos las causas obvias de la inestabilidad y el fracaso, nos quedaríamos con un conjunto de fracasos reales que investigar.

Como un descanso de las pruebas de corrección, intentamos perfilar algunas de las pruebas de larga duración. ¡Esto realmente resultó ser muy lucrativo!

¡Descubrimos que una de las principales rutinas de comparación para validar resultados utilizaba un algoritmo $O(N^2)$! (Para obtener información sobre la notación O, consulte Bell, 2014). Esto estuvo bien para un conjunto de resultados con solo unas pocas entradas, pero el tiempo que llevó ejecutar esta comparación aumentó exponencialmente a medida que crecieron los conjuntos de resultados. Esto fue una locura y, utilizando la experiencia de Jeff, refactorizamos la rutina para que fuera simplemente $O(N)$. ¡Este cambio por sí solo redujo horas el tiempo de ejecución!

Algunas de las otras victorias rápidas fueron cosas como actualizar bibliotecas de terceros a las últimas versiones y una refactorización general del código para "hacerlo mejor", aplicando la regla Boy Scout (Kubasek, 2011) de dejar el campamento más limpio que nosotros. Lo encontré.

Arreglamos muchas más pruebas durante el resto de la semana, algunas fáciles, otras difíciles, y cuando necesitábamos, las emparejamos. Utilizamos la experiencia en desarrollo de Jeff, pero también nos apoyamos en mi experiencia en pruebas para determinar la utilidad de muchas de las pruebas.

Al final de las dos semanas, el número de pruebas en realidad había aumentó a 9.000 (después de que "encontramos" 1.000 pruebas que no estaban siendo ejecutar debido a un error de configuración), pero habíamos reducido con éxito las pruebas fallidas a una, y ahora el tiempo de ejecución promedio del conjunto de pruebas por plataforma se redujo a alrededor de dos horas.

Esto no habría sido posible sin las habilidades combinadas de Jeff y yo. Tener dos perspectivas diferentes y habilidades superpuestas.

Los conjuntos nos colocan en una muy buena posición para resolver cualquier problema que encontremos. También demostramos que una tarea aparentemente imposible era en realidad posible si se aplica un enfoque metódico y multifacético.



La historia de Chris muestra el valor de que los miembros del equipo en diferentes disciplinas asuman la responsabilidad de gestionar la deuda técnica de prueba. Lo mismo ocurre al tomar la decisión de contraer esta deuda. Hay momentos en los que es apropiado tomar atajos para cumplir con una fecha límite comercial crucial, pero tome estas decisiones de manera consciente e involucre a evaluadores, programadores, partes interesadas del negocio, personal de DevOps, analistas y otros roles, según corresponda. Explique a los clientes comerciales los costos futuros de aplazar cualquier tipo de prueba y asegúrese de que comprendan que habrá más trabajo más adelante para rectificar los atajos tomados. Documente las decisiones de alguna manera, tal vez a través de historias de usuarios para el futuro o en la wiki del equipo, para que las razones para omitir alguna actividad de prueba esencial no se pierdan en la noche del tiempo o, peor aún, se olviden por completo.

Resumen

Los equipos pueden decidir incurrir en deuda técnica de prueba al diferir la automatización de pruebas u otras actividades de prueba para adaptarse al negocio, pero esa deuda debe gestionarse para que no ralentice al equipo con el tiempo. La automatización inadecuada de las pruebas, en particular, a menudo lleva a dedicar más tiempo a las actividades de prueba y hace que las pruebas se queden atrás de la codificación.

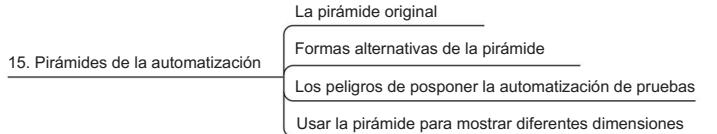
En este capítulo, analizamos algunas formas de gestionar la deuda técnica de pruebas, especialmente en lo que se refiere a la automatización de pruebas.

- Haga visible el tiempo dedicado a depurar fallas de las pruebas y mantener las pruebas automatizadas escribiendo tarjetas de tareas para estas actividades o reflejando el tiempo adicional en estimaciones de historias.
- Hacer visibles los errores y las historias bloqueadas resultantes recuerda al equipo que debe corregir los defectos rápidamente, centrarse en la prevención de defectos y evitar incurrir en deuda técnica en forma de colas de defectos.
- Todo el equipo debe asumir la responsabilidad de gestionar la deuda técnica, tanto de código como de prueba.
- Identificar la mayor fuente de deuda técnica y probar experimentos para reducirla.
- Haga que los miembros del equipo con diferentes especialidades trabajen juntos y apliquen esos diferentes conjuntos de habilidades para reducir la deuda técnica.

Esta página se dejó en blanco intencionalmente.

Capítulo 15

Pirámides de automatización



Una de las preguntas que escuchamos con más frecuencia es: "¿Cómo decidimos qué automatizar?" Nuestro modelo preferido es la pirámide, aunque otros modelos, como los cuadrantes de pruebas ágiles, también pueden ayudar. En los últimos años, los profesionales han adaptado la pirámide de automatización de pruebas de Mike Cohn que presentamos en Agile Testing. Hemos visto algunas extensiones útiles, pero las ideas básicas detrás del modelo siguen siendo las mismas. En este capítulo, exploraremos los beneficios potenciales de las interpretaciones más nuevas.

La pirámide original

Los principios básicos de la pirámide de automatización de pruebas que se muestran en la figura 15-1 todavía se aplican a la mayoría de los equipos y proyectos, y la mayor parte del beneficio se basa en la rapidez con la que se recibe la retroalimentación. El nivel más bajo, las pruebas unitarias, obtiene la retroalimentación más rápida al ejecutarse con cada confirmación de código nuevo.

Cambiamos algunas palabras de la pirámide para aclarar su propósito. La prueba de reglas comerciales en la capa API brinda retroalimentación rápida porque las pruebas en ese nivel no pasan por la interfaz de usuario (UI). La capa superior, en su mayoría pruebas de flujo de trabajo a través de la interfaz de usuario, tiende a brindar la retroalimentación más lenta. El único cambio que hicimos en la pirámide original de Mike Cohn fue la pequeña nube en la parte superior que representa pruebas manuales y exploratorias que no se pueden automatizar por completo. Muchos equipos requieren algunas pruebas de regresión manuales para complementar sus comprobaciones automatizadas; deberían encontrar maneras de mantener esta retroalimentación oportuna.

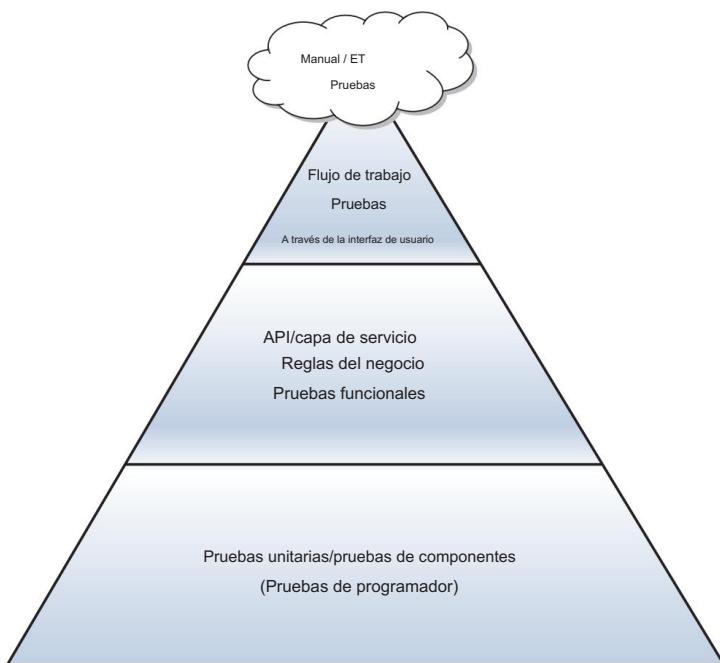


Figura 15-1 Pirámide de automatización de pruebas

Los marcos y herramientas para pruebas en la capa de API/servicio y en el nivel de UI siguen mejorando y nos permiten automatizar pruebas para software más complejo. Vea más sobre esto en el Capítulo 16, “Patrones y enfoques de diseño de automatización de pruebas”.

Formas alternativas de la pirámide

A lo largo de los años, nos hemos dado cuenta de que la pirámide original no se ajusta a todas las situaciones y los avances tecnológicos han cambiado algunas de las suposiciones. Hemos aprendido algunas buenas formas de modificar la pirámide para adaptarla a diversas situaciones, pero aún mantenemos la idea original de dónde automatizar.

La historia de Janet

En una organización que visité, el equipo me había leído que la pirámide Ágil Pruebas Y dijo les era inútil. No podía entender cómo podía ser eso, pero escuché por qué no podían usarlo. Su problema fue que los programadores realmente no hicieron muchas pruebas a nivel unitario, ya que

No hicieron mucho código y no vieron cómo podrían automatizar a nivel de API. En su mayoría, recopilaron datos de muchas fuentes de datos diferentes y luego los manipularon para mostrárselos a los tomadores de decisiones. Entendí su problema y facilité un taller para encontrar alternativas.

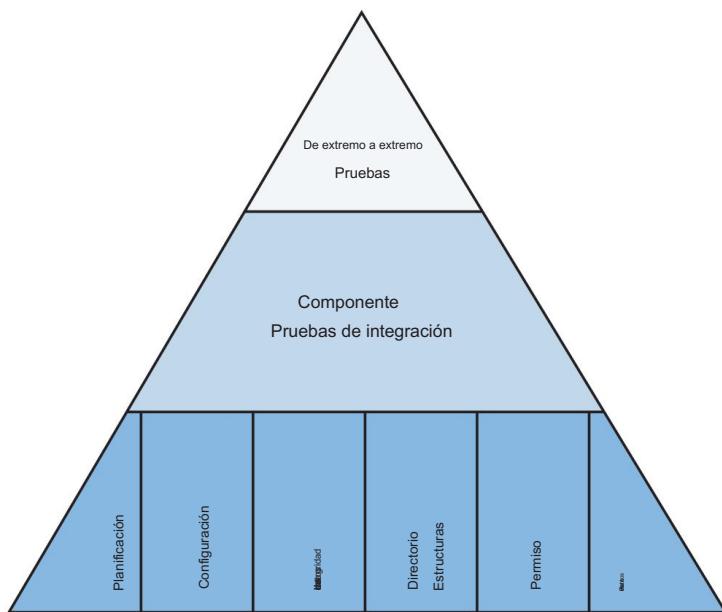


Figura 15-2 Pirámide de pruebas alternativa: sin pruebas unitarias

Primero discutimos los tipos de problemas que estaban experimentando y luego analizamos qué tipos de pruebas podrían ayudar a mitigar esos problemas.

Luego discutimos cuándo podría ser el mejor momento para realizar las pruebas y quiénes podrían ser las mejores personas para realizarlas. El resultado es la pirámide que se muestra en la Figura 15-2. La capa más baja consta de las pruebas de las que los programadores se apropiaron porque querían una respuesta rápida. Discutieron formas de automatizarlos y decidieron experimentar con varias opciones. La capa intermedia comprendía las pruebas de integración de componentes y la respuesta más lenta provino de las pruebas de un extremo a otro.

Los productos de software son cada vez más complejos y los usuarios exigen tiempos de respuesta más rápidos e interfaces más intuitivas. Las aplicaciones web dependen cada vez más de JavaScript para mejorar la experiencia del usuario, por lo que la abstracción entre la lógica empresarial y la capa de presentación se ha desdibujado. Existen herramientas que permiten a los programadores realizar pruebas unitarias de estos

complejidades en la interfaz de usuario, pero en estas situaciones puede ser necesaria una amplia automatización funcional de la interfaz de usuario. Muchos equipos no pueden confiar en pruebas de nivel inferior para protegerse contra regresiones.

La historia de Lisa

El equipo al que me uní en 2003 decidió mantener la interfaz de usuario de nuestra aplicación de servicios financieros basada en web lo más delgada posible. Sin embargo, a medida que la tecnología y las expectativas de los usuarios evolucionaron, necesitábamos ofrecer una experiencia de usuario más rica y a prueba de errores. Más lógica de negocios en el front-end requirió pruebas de UI más sofisticadas. A través de la experimentación y el tiempo dedicado a aprender, pudimos hacerlo de manera mantenible. El nivel superior de nuestra pirámide de automatización de pruebas se expandió, pero el valor superó con creces los costos de las pruebas de UI adicionales (consulte la Figura 15-3).

Las pruebas de mi equipo actual tienen un poco más de forma de reloj de arena, por razones similares. Nuestra interfaz de usuario basada en JavaScript contiene una lógica empresarial muy compleja. Cuando planificamos y estimamos historias de front-end, incluimos tiempo para el desarrollo basado en pruebas a nivel de la interfaz de usuario, así como para agregar o actualizar pruebas de regresión de la interfaz de usuario. Este es un gran ejemplo de por qué las pruebas y la codificación son partes integrales del desarrollo de software. Aunque normalmente hay que realizar pruebas cuando se completa la codificación, la prueba y el código no pueden tratarse como fases separadas. Cuando hablamos de planificar nuestras pruebas, siempre pretendemos que esto sea parte de la planificación general de la historia, la iteración y el lanzamiento.

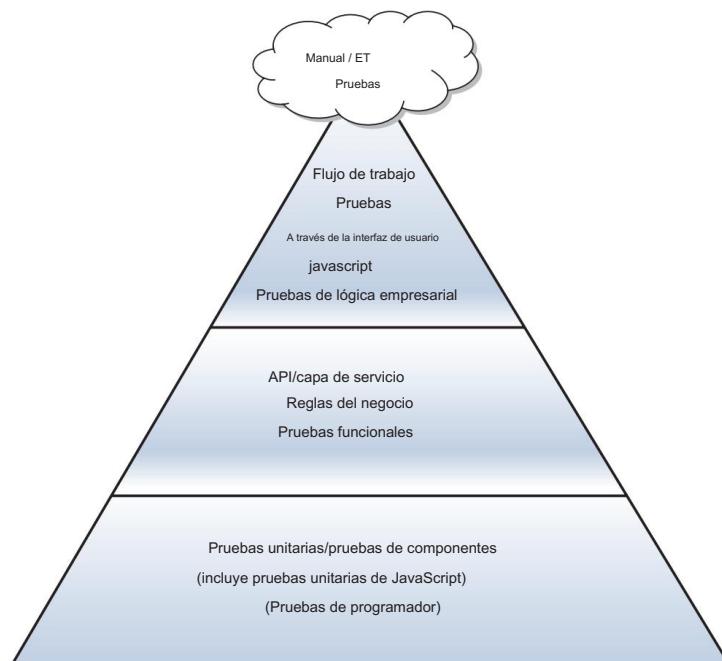


Figura 15-3 Pirámide de automatización de pruebas con complejidades de JavaScript

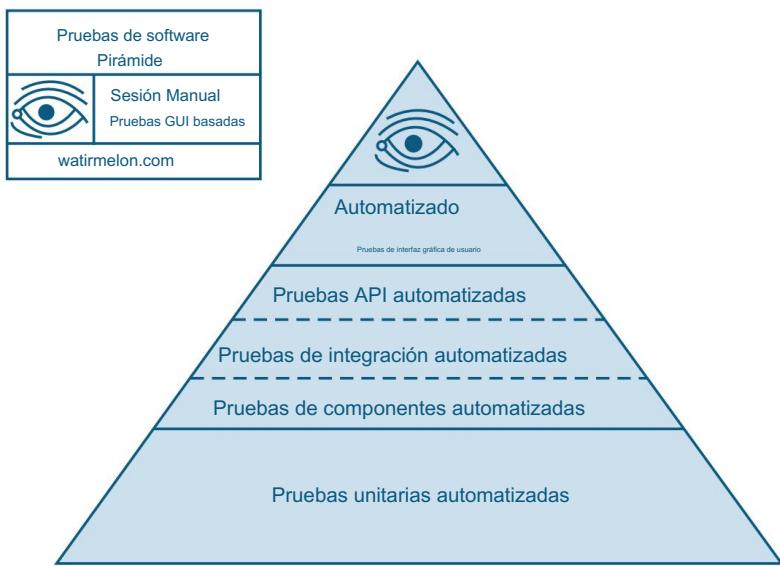


Figura 15-4 Pirámide de prueba automatizada, con el Ojo de la Providencia

Fuente: Scott, Alister, <http://watirmelon.com/2011/06/10/yet-another-software-testing-pyramid>, 2011. Usado con autorización.

Hemos conocido a varios equipos cuya situación simplemente no se ajusta al modelo piramidal original. Está bien. Cada equipo debe desarrollar un modelo que se ajuste a sus necesidades. Lo importante es recordar los principios detrás de la pirámide. Los niveles inferiores representan la retroalimentación rápida que los equipos necesitan a medida que agregan y actualizan código. El objetivo es involucrar a todo el equipo en la automatización de suficientes pruebas de regresión para que el proyecto no quede atrapado en una deuda técnica. Llevar las pruebas a niveles más bajos donde generalmente son más fáciles de escribir y mantener ayuda a lograr el mejor retorno de la inversión.

Alister Scott ha creado una pirámide de automatización de pruebas similar a la original pero que pone un poco más de énfasis en las pruebas exploratorias. En su opinión, las pruebas exploratorias o basadas en sesiones garantizan la confianza en las pruebas automatizadas que se desarrollan y ejecutan. Sin él, una estrategia de pruebas automatizadas es fundamentalmente defectuosa, razón por la cual incluye las pruebas exploratorias como el "Ojo de la Providencia" en su pirámide, como se muestra en la Figura 15-4 (Scott, 2011b).

Los peligros de posponer la automatización de pruebas

La pirámide de automatización de pruebas es un modelo destinado a guiar a los equipos para obtener el máximo valor de su automatización de pruebas por el mínimo.

inversión. Matt Barcomb utilizó la pirámide como base para una fábula sobre lo que sucede cuando no hay un esfuerzo de todo el equipo para implementar una automatización de pruebas adecuada, especialmente en el nivel más bajo.

En La leyenda del volcán de automatización de pruebas (consulte la Figura 15-5), un equipo de desarrollo vive en un pequeño oasis al lado de una pirámide de automatización de pruebas. Los programadores descansan bajo las palmeras y disfrutan de bebidas con sombrilla. Están demasiado ocupados haciendo lo que creen que les gusta más como para automatizar cualquier prueba de regresión para su producto de software. Mientras tanto, los probadores sudorosos trabajan en las entrañas de lo que no es una pirámide de automatización sino un volcán, realizando frenéticamente pruebas de regresión manuales y quedándose cada vez más atrás. Faltan las pruebas dentro del volcán que normalmente lo enfriarían. No hay tiempo que dedicar a ese sol de prueba exploratorio en la cima. La deuda técnica se acumula en forma de lava al rojo vivo. Si el equipo de desarrollo no colabora con los evaluadores y no automatiza suficientes pruebas de regresión, la lava sigue acumulándose.



Figura 15-5 Volcán de prueba automatizado

y calentándose hasta que el volcán entra en erupción, enterrando a los desafortunados programadores en su oasis.

Aunque Matt convierte esto en una historia hilarante, que en su relato incluye vírgenes sacrificadas, el mensaje es serio. Los equipos de la vida real que trabajan bajo la presión de una deuda técnica cada vez mayor sienten un dolor real. A largo plazo, la cobertura insuficiente de las pruebas de regresión automatizadas y la falta de tiempo para actividades esenciales, como las pruebas exploratorias, ralentizarán al equipo y posiblemente podrían detenerlo en seco.

Afortunadamente, escuchamos de cada vez más equipos que pueden aplicar el modelo piramidal de automatización de pruebas para controlar su deuda técnica y acortar sus ciclos de retroalimentación. Veamos algunas historias de éxito.

Cómo volteamos nuestra pirámide

Aquí hay una historia de entusiasmados desarrolladores de software cuyo equipo sufrió una pirámide invertida, sobre cómo lograron darle estabilidad. UN

La primera encarnación de la API de 7digital fue una aplicación monolítica única que comenzó siendo pequeña y se probó principalmente con pruebas completas de extremo a extremo. Como la aplicación era tan pequeña, estas pruebas fueron muy útiles, pero a medida que la aplicación creció, la cantidad de pruebas de un extremo a otro creció con ella. Pronto se llegó a un punto en el que una ejecución completa del conjunto de pruebas tomaría casi una hora y, como puedes imaginar, esto fue extremadamente perjudicial para el ciclo de retroalimentación y la posterior velocidad de desarrollo.

Los desarrolladores comenzaron a emplear tácticas para manejar este conjunto de pruebas de larga duración, por ejemplo, ejecutando solo las pruebas que eran obviamente relevantes para el área en la que estaban trabajando en sus máquinas locales y luego confiando en el servidor de integración continua (CI) para ejecutar la suite completa. Cuando se completó la suite completa, el desarrollador a menudo había pasado a otra cosa, por lo que era probable que se hubiera pasado por alto cualquier falla.

Las propias pruebas también eran propensas a fallos inesperados. Debido a su naturaleza de extremo a extremo, eran muy susceptibles a cambios en cualquier parte de la pila, como una base de datos que dejaba de estar disponible momentáneamente o una fluctuación en la red interna. Estos frecuentes casos de inestabilidad redujeron la confianza en las pruebas, lo que hizo que los desarrolladores estuvieran aún menos dispuestos a ejecutarlas.

El equipo se dio cuenta de que esta situación era insostenible. Estos terminan-
Se habían utilizado pruebas de extremo a extremo para cubrir todos los escenarios, incluidos
todos los casos extremos y todas las variaciones posibles. Era un hábito que comenzó
cuando la aplicación era pequeña y continuó a lo largo de su crecimiento.

Decidimos que necesitábamos abordar este problema directamente, pero era difícil precisarlo:
el alcance era grande y era imposible ver lo que se requería sin profundizar. Adoptamos el
enfoque de que cuando los programadores estuvieran trabajando en un área determinada ,
revisarían las pruebas asociadas, aislarían los principales "viajes del usuario" para que
siguieran siendo pruebas de un extremo a otro y llevarían los casos extremos a pruebas
unitarias y de integración.

Esto extendió significativamente el tiempo para completar cada característica o corrección
de errores, y hubo ocasiones en las que un cambio en las pruebas era tan grande que era
necesario abordarlo por sí solo. En estos casos, redactaríamos una tarjeta (un elemento de
trabajo) para la refactorización, la etiquetaríamos como deuda técnica y la priorizaríamos frente
al resto del trabajo pendiente. Esto no fue difícil de hacer ya que nuestro gerente de producto
apoyó mucho el trabajo.

Pudo ver el efecto que estas pruebas de bajo rendimiento estaban teniendo en nuestra
capacidad para implementar nuevas funciones.

También teníamos una pequeña pizarra en el área del equipo donde grabábamos
algunas ideas y objetivos de refactorización, como "Refactorizar cualquier formulario web para
utilizar un patrón MVP que permita probar los Presentadores". Esto nos mantuvo enfocados
porque estábamos refactorizando como parte de otro trabajo.

El cambio fue gradual y en muchos casos muy frustrante. El estado de la base del código
empeoró mucho antes de mejorar debido al período de transición en el que se utilizaron
diferentes enfoques. La base del código general era confusa a la vista y existía la tentación
de rendirnos y empezar de nuevo, pero perseveramos. El gráfico de la Figura 15-6 muestra el
enorme aumento en el tiempo necesario para completar los elementos de trabajo cuando
comenzamos a abordar el problema. Puede ver claramente en el punto A cuando iniciamos
esta iniciativa a medida que aumentaba el tiempo del ciclo. El efecto positivo en el tiempo del
ciclo se puede ver más adelante en el punto B, cuando el arduo trabajo comenzó a dar sus
frutos y el tiempo del ciclo se redujo, mostrando una tendencia general a la baja.

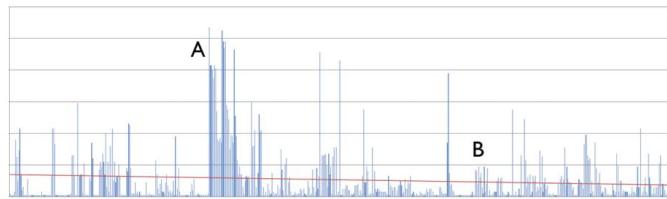


Figura 15-6 Tiempo de ciclo de finalización del elemento de trabajo

El esfuerzo dio sus frutos y el tiempo de ejecución de las pruebas de un extremo a otro se redujo de casi una hora a menos de cinco minutos, con más del cuádruple el número de pruebas unitarias. Los desarrolladores pudieron ejecutar las pruebas unitarias en sus máquinas locales después de cada cambio, confiando en que prácticamente todos los escenarios estaban cubiertos. El conjunto completo de pruebas ahora se ejecuta solo antes de enviar los cambios a la rama principal.

Usar la pirámide para mostrar diferentes dimensiones

Lo llamamos pirámide, pero en realidad es un triángulo. Preferimos modelos simples y usar más de uno cuando corresponda, como la pirámide simple y los cuadrantes de prueba ágil. Descubrimos que cuanto más complejo es un modelo, más difícil es de entender. La gente ha ido añadiendo dimensiones a la pirámide, algunas con más éxito que otras. La siguiente historia es uno de esos éxitos.

La pirámide de automatización de pruebas: ampliada

Sharon Robson le da a la pirámide de automatización de pruebas nuevo algunas dimensiones, reflejando las diversas dimensiones que los equipos de calidad necesitan incorporar en sus productos de software.

Interpreto la pirámide de pruebas ágiles como el comienzo de una discusión sobre la estrategia de pruebas en un equipo ágil. Me gusta mucho la analogía de la pirámide porque una pirámide es una estructura autoportante que no puede existir sin cada uno de sus lados. Cada uno tiene un enfoque (o dirección) diferente, pero cada uno se vincula con la unidad completa. Sigo cuatro pasos para explicar la pirámide a los equipos.

El paso 1 es una explicación básica de la pirámide (consulte la Figura 15-1): qué significa cada capa y para qué está diseñada cada capa. Esto ayuda a diseñar pruebas apropiadas para los miembros del equipo.

El paso 2 es agregar el siguiente lado (derecho) a la pirámide (ver Figura 15-7), que abarca las herramientas que podemos usar para ejecutar estas pruebas. Cada herramienta se elige para el tipo de prueba y el ejecutor de la prueba. Para

Por ejemplo, en la capa de prueba de reglas funcionales y comerciales, analizamos las pruebas del sistema y nos aseguramos de que la funcionalidad fundamental de cada uno de los elementos arquitectónicos funcione tanto individualmente como en conjunto. A nivel de UI buscamos pruebas de aceptación del usuario.

Esto significa que necesitamos tener el tipo correcto de herramienta para que la utilicen los ejecutores de las pruebas en ese nivel.

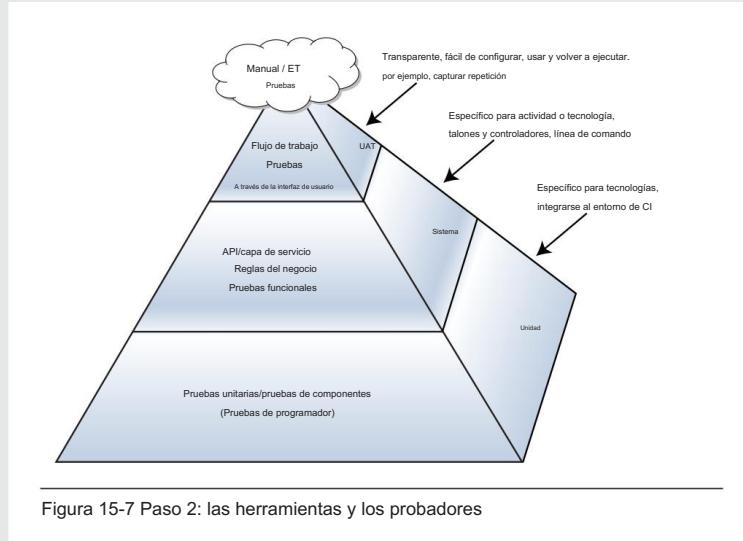


Figura 15-7 Paso 2: las herramientas y los probadores

Esta expansión permite a las personas comprender que no existe una sola herramienta que lo haga todo y que cuando automatizamos debemos hacer coincidir la herramienta con el probador, así como con las pruebas. Este modelo permite tener claridad sobre cómo cambian los tipos de herramientas y los usuarios de las mismas dependiendo de en qué parte de la aplicación o arquitectura nos estemos enfocando.

El paso 3 consiste en extender la pirámide para mostrar tres lados (consulte la Figura 15-8), con la superposición de tipos de pruebas o atributos del sistema. Estos nos permiten estar seguros de que consideraremos todos los aspectos de la solución que deben probarse.

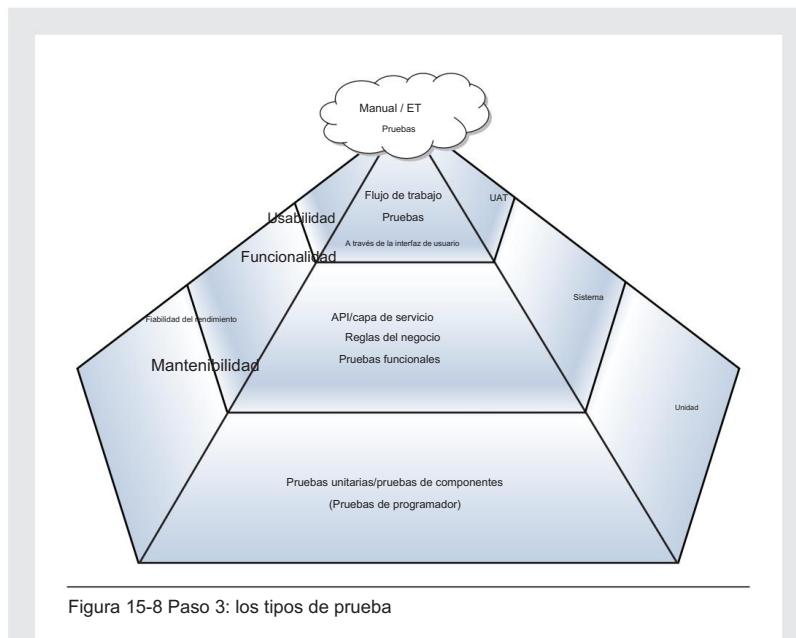


Figura 15-8 Paso 3: los tipos de prueba

Después de estos tres pasos, podemos ver quién se centrará en qué tipo de prueba (por ejemplo, funcional versus no funcional) y por qué estas pruebas deben ejecutarse en los distintos niveles. Esto nos permite entonces

Verifique y vea si la selección de herramientas y las pruebas han sido diseñadas para cubrir el atributo del sistema que estamos tratando de evaluar.

Una vez que se asigna un atributo de prueba al nivel de prueba, las personas del equipo comienzan a comprender cuántas pruebas pueden implicarse y comenzamos a hablar sobre conjuntos de datos, reutilización de datos, pruebas y diseño de pruebas, así como también diseño de pruebas para objetivos específicos. atributos. También animamos a los miembros del equipo técnico a compartir sus conocimientos sobre cómo probar los atributos más técnicos (arquitectura y diseño) en los niveles superiores. Los miembros no técnicos del equipo comienzan a comprender por qué la automatización y el diseño de las pruebas son importantes para la cobertura y la creación de recursos de prueba para la regresión.

Visualmente es bastante difícil representar ese cuarto lado de la pirámide, así que dejo las pruebas de regresión como líneas que recorren el tercer lado,

lo que indica que la regresión puede realizarse sobre cualquier atributo del sistema que se haya considerado como parte de la prueba (ver Figura 15-9).

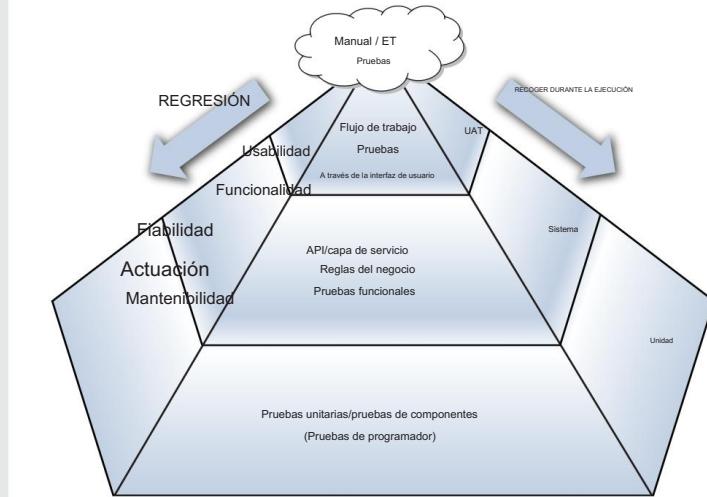


Figura 15-9 Paso 4: pruebas de regresión

Utilizo este proceso de cuatro pasos cuando hablo de automatización en general, no solo en el marco ágil. Es muy útil cuando se consideran las pruebas realizadas por todo el equipo, identificando las pruebas y los conjuntos de datos consistentes y reutilizables que se requerirán, y explicando la necesidad de diferentes tipos y niveles de pruebas para demostrar que cada uno de los componentes y elementos del sistema. , y los atributos se unen en una solución que "funciona" de la manera que esperamos.

Adapte la pirámide de automatización de pruebas para que se ajuste a las necesidades de su equipo, su conjunto de tecnología y su producto. Úselo para automatizar pruebas al nivel óptimo y obtener información rápida sobre fallas de regresión. El modelo piramidal ayuda a su equipo a recordar llevar las pruebas al nivel más bajo que tenga sentido, maximizar el retorno de la inversión en automatización y encontrar formas de incorporar una variedad de características de calidad.

Resumen

La pirámide de automatización de pruebas original es útil, pero no es perfecta para todas las situaciones. Hemos compartido algunas formas en que nosotros y otros profesionales de pruebas ágiles hemos adaptado y ampliado la pirámide para representar estrategias de prueba automatizadas en un modelo visual. Estos diferentes modelos incluyen

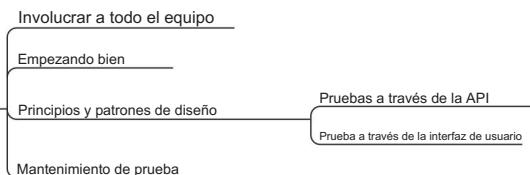
- La pirámide de automatización de pruebas original, probada y verdadera
- Pirámide de pruebas sin pruebas unitarias pero con otras pruebas de programador en cambio
- Pirámide que muestra dónde encajan las pruebas de JavaScript
- La pirámide de pruebas de Alister Scott con el Ojo de la Providencia representando realizar pruebas exploratorias
- **Las lecciones aprendidas en la fábula del volcán de prueba automatizado** de Matt Barcomb
- La pirámide de pruebas ampliada de Sharon Robson que representa múltiples dimensiones de calidad, herramientas y tipos de pruebas.

Esta página se dejó en blanco intencionalmente.

Capítulo 16

Diseño de automatización de pruebas Patrones y enfoques

16. Patrones y enfoques de diseño de automatización de pruebas



Desde que escribimos nuestro primer libro, hemos conocido cada vez más equipos que han logrado sus objetivos de automatización de pruebas. Esta es una gran noticia. Sin embargo, todavía es difícil aprender a automatizar pruebas y probablemente siempre lo será. Los marcos de automatización actuales permiten a los equipos automatizar pruebas en todos los niveles en una sintaxis que los expertos empresariales puedan entender. Sin embargo, siguen atormentándonos dificultades como fallos esporádicos en las pruebas debido a problemas de sincronización en las pruebas de la interfaz de usuario (UI). La “joroba dolorosa” de la automatización de pruebas, como se describe en Agile Testing (p. 266), todavía se cierne como una barrera imponente para los equipos que son nuevos en la automatización de pruebas.

Los equipos que adoptaron la metodología ágil encontraron que los marcos y controladores de automatización de pruebas funcionales existentes eran deficientes, por lo que crearon los suyos propios. Luego, muchos de ellos abrieron esas herramientas. Hoy en día contamos con muchas herramientas de prueba funcionales sofisticadas diseñadas para su uso en proyectos ágiles. Probadores y programadores que realizan desarrollo basado en pruebas de aceptación (ATDD) / La especificación por ejemplo (SBE) y el desarrollo basado en el comportamiento (BDD) ya no tienen motivos para envidiar los fantásticos conjuntos de herramientas disponibles para los profesionales del desarrollo basado en pruebas (TDD).

Necesitamos evolucionar nuestras pruebas y procesos para aprovechar los cambios de herramientas. Nuestros productos cambiarán, lo que probablemente signifique cambios en las estructuras de prueba existentes. Adam Knight (Knight, 2014) sugiere hacernos algunas preguntas, como: “¿Es nuestra estructura de prueba extensible si

¿Necesita admitir nuevas interfaces? A lo largo de los años, y gracias a la capacidad de desarrollar y ampliar los sistemas de prueba, Adam ha podido agregar soporte para la ejecución paralela e incluir escalamiento iterativo de la ejecución de pruebas y múltiples ejecuciones de servidores. Escuchamos acerca de experiencias como la de Adam con más frecuencia cada año que pasa. Cada equipo debe estar preparado para asumir desafíos similares.

Involucrar a todo el equipo

El enfoque de todo el equipo para las pruebas y la calidad es posiblemente el más crítico cuando se trata de automatizar pruebas. Las pruebas automatizadas son código. No solo nos protegen contra fallas de regresión, sino que también nos ayudan a documentar nuestro código de producción, diciéndonos exactamente qué hace nuestro sistema. Como hemos mencionado antes, merecen los mismos cuidados y alimentación que nuestro código de producción.

Cuando los evaluadores poseen la automatización de pruebas, deben dedicar una gran cantidad de tiempo a escribir guiones de prueba para historias en la iteración actual, investigar fallas de prueba y mantener las pruebas automatizadas existentes para que continúen funcionando a medida que se actualiza el código de producción. A menudo queda poco tiempo para actividades cruciales como las pruebas exploratorias. Los programadores que no automatizan pruebas funcionales no tienen ningún incentivo para crear código que se pueda probar porque no sienten el dolor del código que no es compatible con la automatización.

Cuando todo el equipo participa en la automatización de pruebas, los programadores reconocen cómo pueden hacer que su código sea comprobable. Por ejemplo, pueden diseñar el código con diferentes capas, cada una de las cuales puede probarse de forma independiente. Para una aplicación basada en web, simplemente usar identificadores únicos para elementos HTML en lugar de usar nombres dinámicos facilita la automatización de las pruebas de UI.

Es importante que todo el equipo sea dueño de la automatización y vea su valor, para que el trabajo pueda compartirse donde tenga más sentido. Tiene sentido que las personas que saben escribir código escriban el código de prueba. Conocemos a personas que se identifican a sí mismas como evaluadores que también son excelentes codificadores y hacen un gran trabajo diseñando pruebas automatizadas. Sin embargo, en la mayoría de los equipos, las personas con mayor experiencia en codificación son los programadores y sus habilidades pueden usarse para el código de prueba. Crear pruebas y escribir el código para ejecutarlas son dos conjuntos de habilidades diferentes.

Los evaluadores son buenos para saber qué pruebas especificar y qué pruebas cambiar si se cambia la funcionalidad existente. Colaborar entre sí para implementar la automatización de pruebas tiene sentido (consulte la sección más adelante en este capítulo sobre “Pruebas a través de la interfaz de usuario”). Como dijimos en Agile Testing (págs. 300-01), los miembros del equipo que desempeñan otros roles, como administradores de sistemas y administradores de bases de datos, también contribuyen a buenas soluciones de automatización.

Empezando bien



A medida que nos deslumbran más marcos y controladores de automatización de pruebas, es tentador decir: “¡Ooooh! ¡Brillante! ¡Y quedaría bien en mi currículum! ruta. Sin embargo, como aconseja Markus Gärtner (Gärtner, 2012), cada equipo debe decidir primero cómo deberían ser sus pruebas. Esto requiere mucha reflexión y experimentación. Debe responder muchas preguntas, como por ejemplo: “¿Quién necesita poder leer los exámenes? ¿Quién los especificará? ¿Quién los automatizará? ¿En qué herramienta de integración continua (CI) se integrarán? ¿Quién los mantendrá?

Liz Keogh (Keogh, 2013a) sugiere que los equipos implementen ciertas capacidades antes de “tomar el camino de las herramientas”: una mirada al panorama general, la capacidad de cuestionar y explorar, la capacidad de detectar y aceptar la incertidumbre, también como tener excelentes relaciones entre las personas. Aprenda a tener conversaciones para obtener ejemplos de comportamiento deseado antes de decidir cómo encapsularlos en una herramienta en particular. Esto mantiene el enfoque en colaborar con expertos en negocios y ayuda al equipo a encontrar de manera creativa lo que funciona mejor para su situación.

Cree un lenguaje específico de dominio (DSL) que permitirá a su equipo guiar el desarrollo con pruebas orientadas al cliente. Un buen lugar para comenzar es obtener ejemplos de sus expertos en el dominio (consulte el Capítulo 11, “Obtener ejemplos”, para obtener más información al respecto). Entonces, y sólo entonces, busque las herramientas adecuadas que le ayuden a crear pruebas que los usuarios empresariales puedan leer; éstas, a su vez, se convierten en especificaciones ejecutables. Al comenzar con las pruebas del Cuadrante 2 (pruebas empresariales que guían y respaldan el desarrollo), los programadores continuarán usando el mismo lenguaje en todas sus pruebas unitarias y código de producción. Esta continuidad crea un lenguaje común que ayuda a proporcionar documentación viva, documentación que se garantiza que estará actualizada, siempre y cuando las pruebas pasen. Como beneficio adicional, estas pruebas se incorporan a los conjuntos de pruebas ejecutados por su proceso de CI,

donde también protegen a sus usuarios finales de fallas de regresión. Consulte la bibliografía de la Parte VI, “Automatización de pruebas”, para obtener recursos sobre cómo aprender más sobre DSL.

Principios y patrones de diseño

En Agile Testing observamos que los principios de un buen diseño de código se aplican tanto al código de prueba como al código de producción. Principios como No repetirse (DRY) ayudan a evitar la duplicación y garantizan que cuando algo cambie en el sistema bajo prueba (SUT), solo sea necesario actualizar un componente de prueba. El patrón Organizar-Actuar-Afirmar (Ottinger y Langr, 2009a) se usa comúnmente en pruebas unitarias, pero también se aplica a pruebas de aceptación de nivel superior. En este patrón, usted organiza el contexto creando un objeto y estableciendo sus valores, actúa ejecutando algún método y afirma que se devolvió el resultado esperado. La comunidad de software tiene

Tabla 16-1 Reglas simples a seguir para automatizar pruebas

Regla	Razón
Propósito único	Más fácil de depurar; Es más fácil de cambiar si cambian las reglas de negocio.
SECO (No te repitas)	Posibilidad de cambiar pruebas en un solo lugar.
Utilice un DSL (lenguaje específico de dominio)	Facilita la comunicación sobre las pruebas.
Código abstracto de las pruebas.	Hace que las pruebas sean legibles para el negocio
Pruebas de instalación y desmontaje.	Puede ejecutar las pruebas repetidamente
Independencia	Las pruebas pueden ejecutarse de forma independiente y no dependen del orden para ejecutar consistentemente
Evite el acceso a la base de datos (si es posible)	Las llamadas a la base de datos ralentizan las pruebas (tenga en cuenta que en algún lugar puede que sea necesario probar el acceso)
Las pruebas deben ser verdes, todo el tiempo	Confianza en las pruebas; documentación viva
Aplicar estándares de prueba comunes (incluidos convenciones de nombres)	Permite compartir código/propiedad de pruebas y uso común. comprensión de las pruebas
Separar la prueba (el qué) de la prueba ejecución (el cómo)	Abstraer el qué del por qué puede permitir que las capas evolucionar por separado; puede agregar más ejemplos a la especificación legible por humanos (la prueba), o puede cambiar la Automatización subyacente sin afectar las reglas de negocio.

Continuó evolucionando principios y patrones de diseño que reducen el costo de escribir y mantener scripts de prueba automatizados.

Ya sea que esté automatizando a nivel de unidad, API o UI, busque formas de mejorar el diseño de su prueba para mantener al mínimo los costos de mantenimiento a largo plazo y, al mismo tiempo, obtener comentarios rápidos y útiles. Pasos simples, como documentar los patrones de diseño de pruebas para su equipo u organización de desarrollo, pueden ayudar a garantizar la coherencia y la mantenibilidad, además de permitir que otros comprendan la estructura de las pruebas.

La Tabla 16-1 muestra algunas reglas de diseño básicas que creemos que son importantes para que las pruebas sean mantenibles. De ninguna manera es exhaustivo, pero puede brindarle un buen comienzo para experimentar y ver qué funciona mejor para su equipo.

Pruebas a través de la API (a nivel de servicio)

Cuando comience a automatizar pruebas empresariales que guíen el desarrollo, dedique tiempo a las partes interesadas y al equipo de ejecución a decidir cómo desea que se vean las pruebas. Las pruebas deben ser útiles y comprensibles para todos los que necesiten utilizarlas. Recuerde que estos

Las pruebas proporcionarán documentación viva valiosa sobre cómo se comporta el sistema, si continúan aprobándose cuando se realizan cambios.

La Figura 16-1 muestra cómo funcionan generalmente los marcos de prueba a nivel de API. El diagrama no refleja la sobrecarga necesaria para crear bibliotecas de prueba.

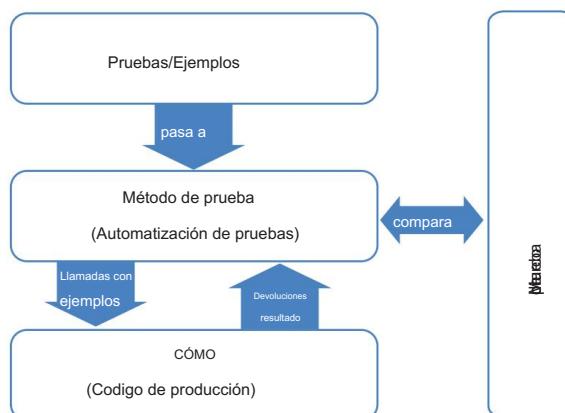


Figura 16-1 Estructura de prueba API

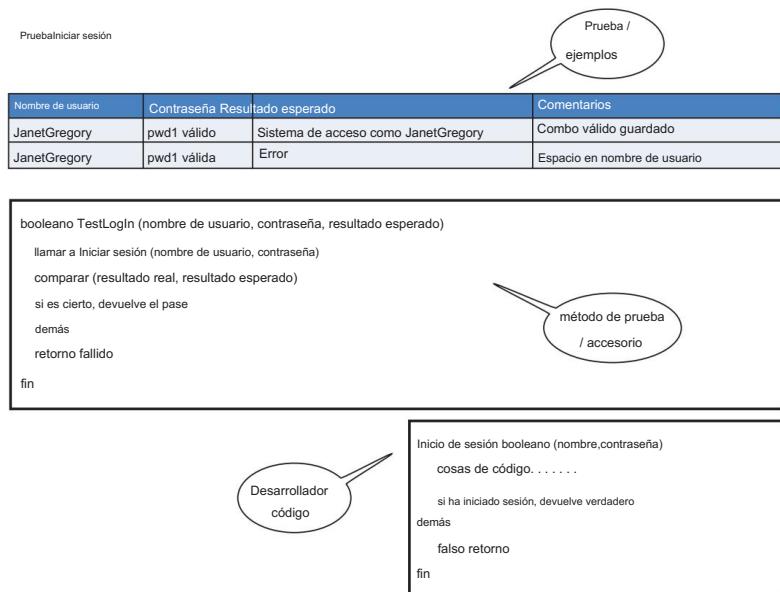


Figura 16-2 Ejemplo de prueba API

y abstracciones. Más bien, queremos llamar la atención sobre la magia de la pieza intermedia: el “pegamento” o método de prueba. Cuando los evaluadores y programadores colaboran para determinar cómo deberían ser las pruebas, suceden cosas sorprendentes que mejoran la comprensión compartida de la historia. Una vez que tenga eso, escribir el código de automatización es un esfuerzo menor.

La Figura 16-2 es un ejemplo de una prueba para un inicio de sesión simple. Hay dos pruebas: una para el camino feliz, un nombre de usuario y contraseña válidos; y uno con un nombre de usuario no válido. Los datos de entrada que se pasan en la primera prueba al método de prueba TestLogin serían <JanetGregory, Validpwd1>. Luego se pasa como variables de entrada al código de producción. El resultado esperado <Sistema de acceso como Janet Gregory> es lo que se comparará con los resultados reales provenientes del código de producción a través del marco de prueba. Por supuesto, cómo se pasan los datos de un lado a otro es algo que los evaluadores y codificadores deben discutir. En esta colaboración sobre lo que se transmite y cómo representarlo es donde ocurre la magia.

El Capítulo 11, "Obtener ejemplos", tiene más información sobre las pruebas debajo de la interfaz de usuario y analiza cómo guiar el desarrollo con ejemplos. Hay varios patrones que se pueden utilizar para trabajar con esta estructura. Un ejemplo es el patrón Puertos y adaptadores (Cockburn, 2005) analizado en Agile Testing (p. 112).

Prueba a través de la interfaz de usuario

En los últimos años, la comunidad ágil ha ideado patrones y prácticas más eficaces para diseñar pruebas automatizadas que ofrecen un gran retorno de la inversión. Esto es válido para todos los niveles de automatización, incluidas las pruebas de regresión que se ejecutan a través de la interfaz de usuario de la aplicación.

La automatización de pruebas que ejercitan la interfaz de usuario de la aplicación sigue planteando los desafíos más difíciles. Muchos equipos han invertido tiempo y dinero para automatizar las pruebas de UI, solo para descubrir que el costo de mantenimiento a largo plazo es abrumador. A lo largo de los años, han evolucionado mejores enfoques. Gojko Adzic (Adzic, 2010b) recomienda pensar en la automatización de la interfaz de usuario en tres niveles:

- Regla comercial o nivel de funcionalidad: lo que la prueba demuestra o pone en práctica; por ejemplo, obtener envío gratis dentro de los Estados Unidos continentales con un pedido por un monto determinado.
- Nivel de flujo de trabajo de la interfaz de usuario: qué actividades de alto nivel debe realizar el usuario en la interfaz de usuario para ejercer la funcionalidad que se está probando; por ejemplo, crear un pedido cuyo monto total califique para envío gratuito
- Actividad técnica: los pasos técnicos para ejercitarse en la funcionalidad en la prueba; por ejemplo, abrir una página, escribir texto en los campos de entrada, hacer clic en botones.

Este enfoque se puede implementar a través de varios marcos de prueba populares, utilizando definiciones de pasos y escenarios, palabras clave y, con algunas adiciones a nivel empresarial, objetos de página. Alister Scott también comparte su experiencia con el enfoque dividido en tres niveles, con ejemplos de especificaciones ejecutables y documentación viva utilizando Cucumber (Scott, 2011a).

Los objetos de página y los recursos de página para marcos no orientados a objetos se pueden utilizar para encapsular todas las cosas que se pueden probar en cada página de una interfaz de usuario. El objeto de página (ver Figura 16-3) incluye toda la funcionalidad para

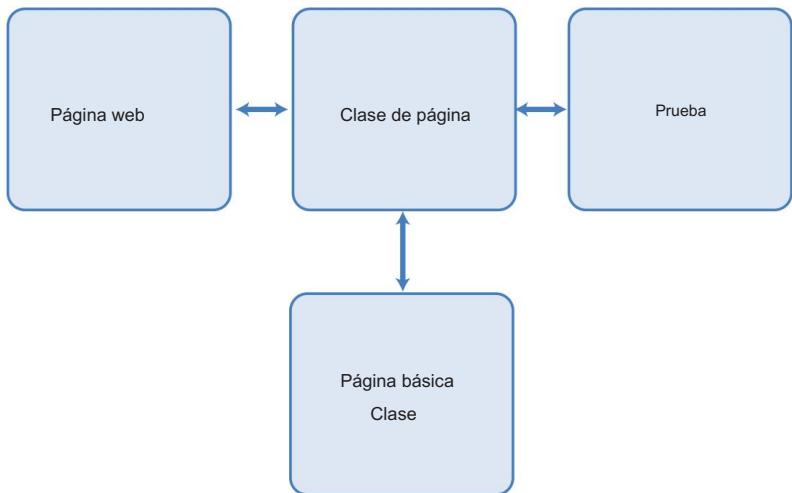


Figura 16-3 Patrón de objeto de página

interactuar con el SUT a través de bibliotecas de prueba de terceros como Selenium. Es más aplicable donde las páginas y las actividades están bien alineadas. Cuando la mayoría de las actividades abarcan varias pantallas o páginas, o cuando una sola pantalla realiza varias actividades, es posible que el objeto de página no encaje bien y eso puede provocar problemas de mantenimiento.

Jeff “Cheezy” Morgan creó un Page Object Ruby Gem de código abierto para probar aplicaciones basadas en navegador (consulte la sección “Herramientas” de la bibliografía para obtener el enlace), y existen otras implementaciones para diferentes lenguajes de programación. Se puede utilizar para probar aplicaciones web, aplicaciones de escritorio, aplicaciones móviles e incluso mainframes.

Consulte el Capítulo 20, “Pruebas ágiles para sistemas móviles e integrados”, para conocer la historia de Cheezy sobre cómo automatiza las pruebas de aplicaciones móviles.

Cheezy explica cómo se puede utilizar el patrón Objeto de página con el enfoque de tres capas (Morgan, 2014):

PageObject es un gran patrón para construir una abstracción sobre el sistema bajo prueba. Aún debe proporcionar un lugar donde exprese claramente las reglas comerciales que se verificarán, los flujos de trabajo o rutas que se tomarán a través de la aplicación para completar el comportamiento y los datos que necesita el

aplicación para completar el flujo de trabajo. Considero que estas tres partes adicionales de la prueba son muy distintas. Utilizo una herramienta BDD como Cucumber para expresar las reglas o el comportamiento comercial y otras bibliotecas para proporcionar navegación y pruebas. Encuentro que esta separación de preocupaciones hace que mi código de prueba sea mucho más limpio y más fácil de adaptar a medida que cambia la aplicación.

Su equipo debe decidir cómo quiere que se vean sus pruebas y luego experimentar con diferentes patrones y enfoques para ver cuál funciona mejor.

La historia de Lisa

El producto de nuestro equipo era una aplicación web que originalmente tenía un cliente ligero.

Toda la lógica empresarial estaba del lado del servidor y automatizamos la regresión.

lo prueba a nivel API. Nuestra herramienta de prueba de UI funcionó a través de la capa HTTP,

Así que a veces tuvimos problemas con los eventos del lado del cliente, pero funcionó bien para

Cubra las pruebas de regresión de la interfaz de usuario.

Más tarde, a nuestro equipo se le ocurrió un nuevo código de interfaz que sabíamos que ayudar a reducir los costosos errores del usuario, pero nuestra herramienta de prueba de UI no pudo "ver" el evento. Sentimos que era demasiado arriesgado implementarlo sin una regresión automatizada. pruebas, por lo que era hora de encontrar un nuevo marco y controlador de prueba de UI.

El patrón Objeto de página proporcionó una forma atractiva de crear elementos mantenibles.

Pruebas de interfaz de usuario. Iniciamos una serie de "bake-offs" para identificar el mejor enfoque que utilizó el objeto de página y permitió a los evaluadores y codificadores colaborar estrechamente. Utilizamos un enfoque de estilo de desarrollo basado en conjuntos. Primero nosotros Acordamos cómo nos gustaría que se vieran nuestras pruebas. Consideramos un dado_ entonces_ cuando el estilo, que le gustó a nuestro propietario de producto. Pero sabiendo que prefería Para no mirar los casos de prueba detallados, decidimos seguir con una afirmación formato cercano al de nuestras pruebas existentes.

A continuación, dos personas probaron cada una un conjunto diferente de herramientas para realizar una prueba de concepto y compartieron sus resultados con el resto del equipo. tomó un par de rondas y mucho tiempo, pero la inversión dio sus frutos. todo el equipo

Elegimos el mejor conjunto de herramientas para nuestras necesidades y contratamos a un entrenador experto. para ayudarnos a empezar bien. La curva de aprendizaje tomó algunas semanas, pero pronto Estábamos escribiendo nuevas pruebas rápidamente y disfrutando de un breve ciclo de retroalimentación de pruebas mantenibles.

Algunas personas podrían pensar que tomarse el tiempo para experimentar cómo funcionan las pruebas

Mira, y el mejor marco para crearlos es demasiado caro. pero puedo decir

usted por experiencia, es mucho más caro a largo plazo si está

tratando de conformarse con el formato y marco de prueba incorrectos. Las pruebas fallarán

con mayor frecuencia, y cada falla en la prueba tomará más tiempo para diagnosticarse y corregirse. Pasarás cada vez más tiempo refactorizando.

La experiencia de Lisa ilustra el valor de que miembros del equipo con diferentes especialidades y habilidades colaboren para encontrar las soluciones de automatización más adecuadas para su equipo. Uno de los principales defensores del objeto de página en su equipo fue el administrador principal del sistema, quien también es programador y defensor de la automatización de pruebas útiles. Su historia sigue.

Objetos de página en la práctica

dulces tony , un arquitecto de tecnologías de la información de Colorado, EE. UU., describe las ventajas de utilizar la automatización a Página Objeto fragmentos a través de la pieza de la prueba del usuario. Ha proporcionado algunos de patrones y ejemplos de cómo implementarlos en la práctica: ejemplos", para

aquellos lectores que más detalles deseo Código completo para el encuentra puede Cuenta de GitHub de Tony (Sweets, 2013).

El software orientado a objetos es una forma probada de producir código que se pueda mantener y sea fácil de modificar. Los patrones de diseño son clave al escribir en un forma orientada a objetos; saber qué son y cómo utilizarlos mejora tu código. Un patrón de diseño puede considerarse como una receta para cómo construir algo. Un patrón que los desarrolladores de software y la comunidad de pruebas han aceptado es el patrón de objeto de página que permite implementar sus pruebas automatizadas de forma orientada a objetos.

Entendiendo el Página Objeto Patrón

Uno de los problemas que los objetos de página intentan resolver es la reutilización. Es mejor para encapsular la pieza que normalmente habrías cortado y pegado en un solo módulo (u objeto) y reutilizar ese único objeto donde lo necesita. De esa manera, generalmente puedes actualizar un solo objeto. Mientras la interfaz del objeto siga siendo la misma, los detalles de implementación no importan y sus pruebas deberían seguir funcionando.

Su objetivo como redactor de pruebas automatizado es separar su prueba de cómo está implementado. Por ejemplo, cuando necesite escribir una prueba de inicio de sesión, la prueba de alto nivel no debería necesitar cambiar cuando algo en el cambios de aplicación. Digamos que este es su script de prueba de alto nivel:

Paso 1: abra el navegador web y vaya al sitio del SUT.

Paso 2: Verifique que estemos en la página de inicio y que haya un cuadro de inicio de sesión.

Paso 3: inicie sesión como Jane Doe.

Paso 4: Verificar que estemos en la página de inicio de Jane Doe.

Observe que no dije cómo iniciar sesión. No dije: "Escriba 'jdoe' en el cuadro de nombre de usuario y 'contraseña123' en el cuadro de contraseña, luego presione Enviar". Simplemente dije: "Inicie sesión como Jane Doe". La cosa u objeto al que llamo debe tener esos detalles. ¿Qué pasa si en el futuro el propietario del producto presenta un nuevo requisito para agregar un tercer cuadro al cuadro de diálogo de inicio de sesión para obtener un PIN, de modo que necesite un nombre de usuario, un PIN y una contraseña para iniciar sesión? Si tuviera 50 pruebas que comenzaran con el inicio de sesión, tendría que realizar 50 cambios. Pero si utiliza objetos de página, debe realizar el cambio en un solo lugar.

La idea es simple; necesita una capa entre su prueba y la implementación de la prueba. Su prueba puede llamar a métodos en un objeto cuya interfaz debería ser bastante estable. Cuando se producen cambios en el sistema y debe tenerlos en cuenta en su prueba (por ejemplo, hay un nuevo requisito para agregar un campo CAPTCHA en la página de inicio de sesión, o el PIN en el ejemplo anterior), simplemente puede cambiar la implementación de el objeto de página de inicio de sesión pero no su interfaz; por lo tanto, no necesitará cambiar su prueba real.

Implementando el Página Objeto Patrón

La idea simple detrás del patrón Objeto de página es representar cada página de su sistema como un objeto. Este objeto de página encapsulará los elementos de la página, así como cualquier operación que se pueda realizar mientras se visualiza esa página.

Lo primero que debe hacer un objeto de página es verificar que realmente represente la página que se muestra. Esto generalmente se hace cuando se inicializa el objeto. Por ejemplo, un objeto de página de inicio podría verificar que las palabras "Página de inicio de Cool System" estén en la barra de título.

Las operaciones (o métodos) deben devolver un objeto de página. Si, después de iniciar sesión en una aplicación, espera ver un panel, su método de inicio de sesión debe devolver el objeto de página del panel.

No hay pruebas en el objeto de página (aparte de verificar que la página se haya representado correctamente). Sus pruebas son independientes y están escritas como un script, pero incluyen estos objetos de página y los utilizan. El siguiente fragmento de script muestra el script de prueba para verificar que un usuario puede navegar a una página específica en la aplicación bajo prueba:

```
// Ir a la lista de automóviles Nissan log.info  
("Haciendo clic en la lista de automóviles Nissan"); NissanCarListPage  
nissanCarListPage = nissanPage.selectListOfCars();  
Assert.assertNotNull(nissanCarListPage);
```

El marco de prueba (en este caso, la clase `Assert` de JUnit) le permite verificar los resultados esperados, como nulos, igualdad y verdadero/falso. Una prueba El error detiene el script y registra información sobre el error para que pueda Puede depurarlo en los resultados de la prueba a través del sistema de compilación, la línea de comando, o un informe. Al final del script de prueba enviamos un comando de salida a el objeto WebDriver, que realiza tareas de limpieza, como cerrar el navegador.

Consulte el Apéndice A, “Objetos de página en la práctica: ejemplos”, para ver los ejemplos de Tony sobre cómo implementar el patrón Objeto de página, con fragmentos de código y detalles técnicos. Consulte la bibliografía de la Parte VI para obtener más recursos, incluido uno de Anand Ramdeo sobre el manejo de problemas comunes con el modelo de objetos de página (Ramdeo, 2013).

Mantenimiento de prueba



Como señalamos anteriormente, es necesario aplicar buenas prácticas de diseño para crear código de prueba automatizado cuyo valor supere su costo de mantenimiento. Si es nuevo en la automatización de pruebas, el artículo de Dale Emery sobre cómo escribir pruebas automatizadas mantenibles es un buen lugar para comenzar (Emery, 2009). Recuerde, esta es un área donde se requiere experiencia tanto en programación como en pruebas, así que colabore y experimente para desarrollar la automatización de pruebas que funcione para su equipo.

Muchos equipos automatizan y luego se olvidan de las pruebas, siempre y cuando estas vayan pasando. Sin embargo, con el tiempo, es posible que notes que las pruebas tardan cada vez más en ejecutarse. Las pruebas automatizadas requieren atención continua. Es posible que desees estar atento a la duplicación o a los lugares para refactorizar porque encuentras una mejor manera. Es posible que descubra lagunas en la cobertura de sus pruebas automatizadas. Al brindar visibilidad a los resultados de sus pruebas automatizadas, puede monitorear continuamente y asegurarse de que sigan haciendo lo que espera. Los marcos de prueba actuales, ya sea que esté realizando pruebas a nivel de servicio o a través de la interfaz de usuario, pueden hacer que escribir pruebas automatizadas sea rápido y fácil, así que tenga en cuenta que ese no es el final de la historia. Debe mantener esas pruebas a lo largo del tiempo, asegurarse de que arrojen resultados precisos y mantener breve el ciclo de retroalimentación. Hay muchas áreas en las que pensar más allá de redactar el examen inicial.

La importancia de la gestión de datos de prueba

Jeff "Cheezy" Morgan

comparte lo suyo sobre la importancia

de la gestión de datos para la automatización.

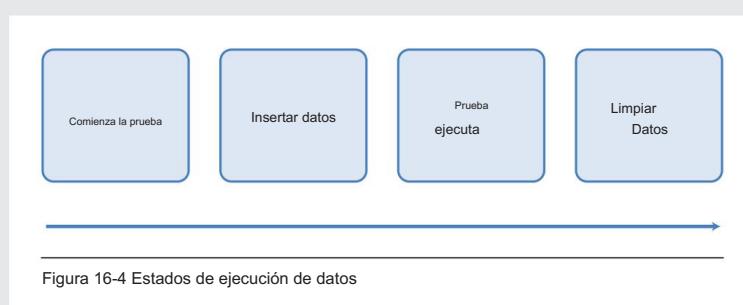
histórica

No hay peor sensación que llegar al trabajo por la mañana, sólo que ver que una gran cantidad de las pruebas que se realizaron durante la noche han fallado. A medida que investiga el motivo de las fallas, descubre rápidamente que Alguien ha cambiado los datos de los que dependen sus pruebas.

Este escenario es demasiado común. Los evaluadores pasan mucho tiempo investigando fallas en las pruebas, tratando de descubrir si fallaron debido a un problema con el sistema bajo prueba o debido a algún problema ambiental. cuando las pruebas fracasan debido a problemas ambientales, el equipo a menudo pierde la fe en el pruebas. Cuando eso suceda, probablemente comenzarán a ignorar los fracasos. Prueba Los datos contribuyen de manera importante a los problemas ambientales.

Es extremadamente importante que nuestras pruebas automatizadas se ejecuten constantemente y otra vez. Para hacer esto tenemos que tener buenos datos de prueba. estrategia administrativa. Nuestras pruebas nunca pueden fallar porque alguien haya Cambiamos los datos de los que dependen nuestras pruebas o porque algunos datos que estábamos esperando estar en la base de datos no estaba allí.

La única manera de garantizar que los datos estén disponibles en el estado exacto lo que desea es que sus pruebas creen los datos antes de cada ejecuciones de prueba. Además, para devolver el sistema a un estado conocido, sus pruebas deben limpiar los datos al final de la ejecución (ver Figura 16-4).



Si su conjunto de pruebas tiene acceso directo a la base de datos, esto es fácil. Puede simplemente inserte los datos en las tablas apropiadas, ejecute su prueba, y luego eliminar los datos de la base de datos.

A veces no tenemos acceso directo a la base de datos sino que debe utilizar algún tipo de servicio web para poder configurar los datos. El

El patrón debe seguir siendo el mismo. Deberíamos utilizar los servicios para crear los datos que necesitamos, ejecutar la prueba y luego eliminar los datos que necesitamos utilizado para nuestra prueba.

¿Qué debemos hacer si los datos que estamos utilizando provienen de otra empresa y no tenemos control sobre ella? Además, ¿qué pasa si esos datos es completamente volátil? En casos como este, a menudo necesitamos construir soluciones inteligentes. servicios sustitutos con los que nuestros conjuntos de pruebas pueden ejecutarse para tener datos consistentes. Estos servicios deberían permitir que las pruebas especifiquen el datos que se devolverán cuando se llamen en determinadas situaciones. Estos servicios introducen riesgos para nuestro equipo, ya que no estamos ejecutando contra los servicios reales contra los que se ejecutará nuestra aplicación cuando esté en producción. Para mitigar este riesgo necesitamos realizar pruebas con el servicios reales con frecuencia.

Si está utilizando el sistema bajo prueba para configurar los datos de prueba, deberá uso, debería considerar el uso del patrón de datos predeterminados (Morgan, 2010) con datos aleatorios. Esto le permitirá especificar los datos. eso importa dejando el resto de los datos para el marco crear.

Hay algunas razones por las que eliminamos los datos después de cada ejecución de prueba. La razón más simple es que si insertamos algunos datos, los modificamos mediante una prueba, y luego, durante una ejecución de prueba posterior, intentamos insertar los mismos datos, estamos Es probable que se produzca un error. Este error puede deberse a que algunas tablas de la base de datos tener restricciones que no permitan múltiples copias de los mismos datos.

Otra razón para limpiar los datos con cada ejecución de prueba es que los datos que permanece en la base de datos durante largos períodos de tiempo es sospechoso. Somos Nunca estoy seguro si alguien o algo ha cambiado los datos desde que fue insertado originalmente. Como resultado, no se puede confiar en estos datos.

La gestión de datos es una parte importante del mantenimiento de las pruebas y, a menudo, no recibe el respeto que merece. Existen bibliotecas para generar datos de prueba aleatorios pero realistas, como Ruby Faker y jdefault (consulte la sección "Herramientas" de la bibliografía para obtener enlaces). Experimente con diferentes opciones para crear y administrar datos para su uso en pruebas automatizadas. Asegúrese de tener una forma de reproducir cualquier problema descubierto mediante pruebas automatizadas utilizando datos generados aleatoriamente.

Otro aspecto de la mantenibilidad es la facilidad para depurar fallas en las pruebas. Es importante poder profundizar en los fallos de las pruebas para identificar rápidamente exactamente qué salió mal. Puede resultar útil escribir scripts de prueba.

información para registrar archivos para una depuración más profunda cuando sea apropiado. Tenga en cuenta que si ha mantenido sus pruebas con un solo propósito, es posible que no necesite tanta información de registro, porque es más sencillo encontrar la falla.

Resumen

Los mismos principios de diseño, como Don't Repete Yourself (DRY), que ayudan a crear un código de producción sólido, también se aplican al código de prueba. La forma en que desarrollemos nuestras pruebas determinará cuánto valor obtendremos de ellas.

- El código de prueba automatizado es tan valioso como el código de producción. Tiene sentido que los programadores del equipo colaboren con los evaluadores para escribir ese código de prueba automatizado.
- En este capítulo analizamos diferentes principios y patrones de diseño, incluido un enfoque de tres niveles para la automatización de la interfaz de usuario y el patrón Objeto de página. Experimente con diferentes principios y patrones para encontrar cuál se aplica mejor a sus pruebas.
- Comprender cómo colaborar para aprovechar las herramientas que automatizar a través del nivel de API/servicio y debajo de la capa de UI.
- La gestión de datos para pruebas automatizadas es fundamental, así que dedique tiempo a crear la estrategia de datos adecuada para su equipo u organización.
- Monitorear los costos de mantener pruebas de regresión automatizadas y tomar medidas para mantenerlas oportunas, con un retorno de la inversión positivo.

Esta página se dejó en blanco intencionalmente.

Capítulo 17

Seleccionar la automatización de pruebas

Soluciones

17. Selección de soluciones de automatización de pruebas

Soluciones para equipos en transición

Enfrentando nuevos desafíos de automatización con todo el equipo

Lograr un consenso de equipo para soluciones de automatización

¿Cuánta automatización es suficiente?

Soluciones colaborativas para elegir herramientas

Ampliación de la automatización a grandes organizaciones

Otras consideraciones de automatización

Tiene algunas ideas sobre cómo superar su deuda técnica de prueba.

Ha utilizado los modelos piramidales del Capítulo 15, “Pirámides de automatización”, para considerar una estrategia para automatizar pruebas en diferentes niveles.

Ha utilizado técnicas del Capítulo 16, “Patrones y enfoques de diseño de automatización de pruebas”, para determinar cómo deberían ser sus pruebas en uno o más niveles. Ahora necesitas ir a los detalles. Por ejemplo, su equipo debe crear una infraestructura para escribir y ejecutar pruebas automatizadas, lo que puede requerir hardware y software nuevos.

Debe seleccionar marcos y controladores de prueba adecuados y, en la mayoría de los casos, se trata de un esfuerzo a largo plazo. Un enfoque incremental e iterativo le permite aplicar lo que aprende a lo largo del camino para desarrollar pruebas automatizadas valiosas. En este capítulo compartiremos no solo nuestras ideas, sino también historias de equipos que han luchado por la adopción de la automatización y cómo superaron sus problemas.

Soluciones para equipos en transición

Elegir e implementar nuevas soluciones de automatización de pruebas es especialmente desafiante para los equipos que también son nuevos en el desarrollo ágil. Conseguir un entrenador o tener miembros del equipo con experiencia en esta área no

Garantiza un camino fácil hacia el éxito, pero puede ayudar al equipo a superar el “dolor” de la automatización del aprendizaje (*Agile Testing*, p. 266).

Implementación de soluciones de automatización en una organización grande

está la historia de un de Aquí Zanahoria Cirilo, prueba de automatización y consultor en los Países Bajos, sobre cómo evolucionaron las soluciones de automatización en una gran organización bancaria.

Al ser un tester práctico y completo, con mucha experiencia de campo, estaba ansioso por empezar a entrenar a la próxima generación de testers ágiles. La empresa para la que trabajo se especializa en coaching ágil y estaba ejecutando un programa completo de adopción ágil en uno de los bancos más grandes de los Países Bajos. La transición fue notablemente bien; Grandes espacios abiertos con enormes tableros Scrum marcaban los espacios de oficina, y la colaboración animada entre la mayoría de las disciplinas ya era una práctica común.

Algo que no tuvo tanto éxito como la adopción ágil general fue la integración de los evaluadores en los equipos. Sí, había evaluadores en cada equipo y sí, intentaban colaborar. Pero la mayoría de ellos tenía poca experiencia con el desarrollo ágil y solo unos pocos tenían experiencia con la automatización de pruebas. Nuestro objetivo era llevar la automatización a un nivel superior e involucrar más a los evaluadores en el proceso.

Empecé entrenando a tres equipos que ya habían comenzado a utilizar FitNesse en combinación con Selenium. Ambas herramientas eran bien conocidas por mí, que fue una de las razones por las que fui seleccionado para el trabajo.

Los equipos enfrentaron problemas prácticos similares. La automatización de pruebas era inmadura, los entornos de prueba eran inestables y no había control de versiones ni de las pruebas ni de las herramientas. Tampoco había gestión de configuración, lo que resultó en muchas combinaciones diferentes de versiones de herramientas y navegadores. No había ningún concepto de arquitectura de prueba ni ningún enfoque en la mantenibilidad de la prueba. La Figura 17-1 muestra lo que sucede si no hay colaboración cuando no hay ningún enfoque.

Dediqué mi tiempo a soluciones prácticas, ayudando con problemas técnicos e introduciendo configuraciones y control de versiones más maduros.

Tuve que explicar los fundamentos de la automatización de pruebas y enseñar todas las habilidades técnicas necesarias. Nosotros (los entrenadores asignados y yo) colaboramos para intentar crear más sinergia entre programadores, evaluadores y analistas.

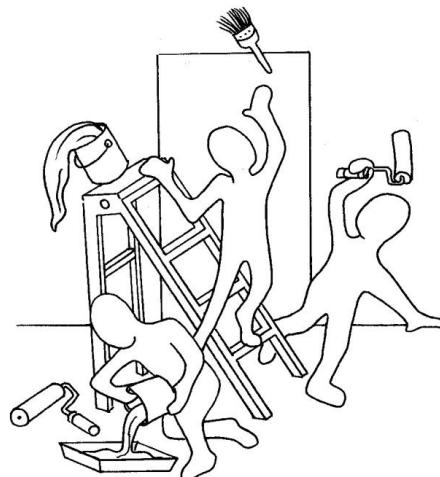


Figura 17-1 Trabajar sin colaboración

Los equipos con los que comencé tuvieron bastante éxito y la noticia se extendió por todo el banco. Pronto, personas de otros departamentos empezaron a utilizar la misma configuración. Esto hizo que la cantidad de equipos que tenía que apoyar creciera rápidamente.

Con esta expansión surgieron nuevos obstáculos. Se perdió mucho tiempo simplemente haciendo instalaciones y configuraciones, y había muchos problemas técnicos que resolver. Además de cuestiones más específicas, como tener que lidiar con molestias o peculiaridades de FitNesse y Selenium, también hubo problemas de infraestructura o de software, que estaban fuera del alcance de mis actividades. Cuanto más se difundió el nuevo enfoque, más obstáculos aparecieron. Pronto resultó evidente que el sistema establecido no estaba tan abierto al cambio.

Hablé con gerentes que estaban promoviendo la automatización de pruebas y las pruebas ágiles y animé al departamento de “Soporte de herramientas” a permitir que las herramientas que estábamos usando se difundieran más fácilmente. También llevó tiempo convencer al “Centro de excelencia de automatización de pruebas” de que obtendrían nuevas opciones positivas. A menudo, la burocracia y los “estándares” de la empresa se utilizaron como barrera. El uso por parte de la empresa de herramientas comerciales listas para usar se defendió basándose en sus costosas licencias, aunque las herramientas no servían a los equipos.

Mucha gente teme el cambio y hará cualquier cosa para impedirlo. Las razones que usaron contra mis soluciones propuestas iban desde divertidas hasta ridículas y rara vez tenían sentido. El estándar de la empresa simplemente no era ágil. Creé el siguiente conjunto de reglas para convencer a mi "oposición" de que se requería un cambio:

1. Cuando se trabaja en un entorno ágil, la automatización es vital
importancia para mantener el ritmo del proyecto. Depender únicamente de pruebas manuales no es una opción.
2. La automatización debe ser posible dentro del equipo multidisciplinario; no se debe requerir experiencia externa para crear o mantener pruebas.
3. Una herramienta de prueba debe ser flexible, facilitando la respuesta al cambio en términos de limitaciones ambientales, de datos y técnicas.
4. La creación de casos de prueba automatizados debería ser relativamente simple y intuitivo.
5. Las pruebas deben ser legibles y comprensibles para cualquiera. Esto se apoya en la documentación y el formato de las pruebas en un lenguaje inequívoco y ubicuo.
6. Las pruebas deben ser fácilmente accesibles e intercambiables entre los miembros del equipo y preferiblemente también para otras partes interesadas. Cualquier miembro del equipo debería ser capaz de agregar, ver y ejecutar pruebas.
7. La ejecución de la prueba debe ser rápida y confiable, ya que el objetivo principal es una retroalimentación rápida.

Con mucha persuasión y perseverancia, sin mencionar la colaboración con las personas adecuadas, la popularidad del conjunto de herramientas creció y la mayoría de los equipos ágiles pasaron a utilizarlo de una forma u otra.

Las herramientas se hicieron más populares, pero la atención que podía prestar al coaching era limitada. FitNesse es, en muchos sentidos, una herramienta excelente y permite a los usuarios mucha libertad. Para muchos evaluadores sin experiencia, esto puede suponer demasiada libertad. Con el tiempo limitado del que disponía, todo el esfuerzo se salió de control; creció más allá de mis capacidades. ¡Era hora de más cambios!

Para equilibrar mis actividades, tuve que difundir el conocimiento de una manera más eficiente. Para difundir información en profundidad, comencé a organizar cursos. En los cursos podían participar de 10 a 15 personas a la vez, lo que era mucho más efectivo que pasar información uno a uno.

Creamos una wiki con conocimientos básicos sobre pruebas ágiles. Contenía una importante sección de herramientas, junto con una guía de instalación, con soluciones para los problemas de instalación más comunes. Tenía una sección de preguntas frecuentes, llena de los problemas más comunes recopilados en el período anterior y contenía métodos reales que podrían usarse en situaciones complejas específicas.

Para construir activamente una comunidad, organicé sesiones de conocimiento, y estas sesiones mensuales atrajeron a una audiencia considerable. Se invitó a evaluadores de diferentes equipos exitosos a presentar sus esfuerzos, especialmente aquellos que utilizaron enfoques muy diferentes para tratar de ampliar las opiniones de las personas. Estas sesiones fueron extremadamente valiosas; La gente empezó a intercambiar ideas y me pareció que surgió una mayor cohesión entre los equipos. La Figura 17-2 muestra cómo colaborar y trabajar juntos puede hacernos avanzar.

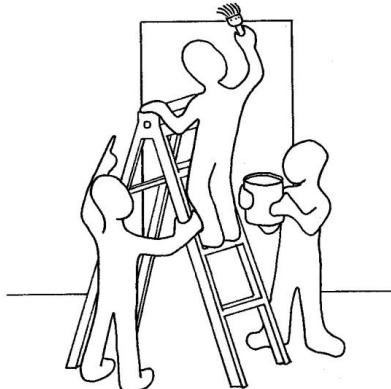


Figura 17-2 Cuando colaboramos

Los colegas comenzaron a hacer que FitNesse fuera más atractivo para una audiencia más amplia, lo que finalmente resultó en una refactorización completa del marco.

Recopilé más medidas de progreso. Un aprendiz de gestión inició un experimento con dos equipos comparables para medir la eficacia, uno utilizando el enfoque ágil y el otro un método más tradicional.

Desafortunadamente, justo cuando las cosas finalmente estaban en marcha, un cambio organizacional significó que mi función ya no estuviera financiada y mi asignación terminó.

En retrospectiva, debo concluir que el grado de éxito de mi trabajo fue algo decepcionante. El nivel general de calidad estaba por debajo de mis estándares personales y algunos equipos fracasaron por completo en la automatización. Sin embargo, cuando se considera desde una perspectiva más amplia, los evaluadores se involucraron más en la automatización de pruebas y en el proceso ágil. A menudo escuché que el cambio había aumentado drásticamente su satisfacción laboral. Muchos equipos todavía utilizan el mismo enfoque y la misma configuración que se introdujo y se convirtió en el estándar para proyectos ágiles dentro del banco.

Lo que al final es más importante para mí es el crecimiento personal que experimenté. Una experiencia de aprendizaje como esta es invaluable.

Cirilo logró ayudar a los equipos dentro de la organización bancaria a implementar un conjunto de herramientas específico que funcionó para su contexto. La cultura organizacional es difícil de superar, pero tenía buenas estrategias para ayudar a los equipos a tener éxito con la automatización. Estableció reglas para que el enfoque fuera consistente en todos los equipos. Él y sus colegas refactorizaron uno de sus marcos de pruebas de código abierto, FitNesse, para adaptarlo mejor a sus necesidades. Encontró formas efectivas de medir el progreso y creó una comunidad de automatización de pruebas. Considere estas ideas mientras su equipo implementa sus propias soluciones de automatización de pruebas.

Enfrentando nuevos desafíos de automatización con todo el equipo

Si necesita llenar un vacío en la automatización de sus pruebas, mejorar el retorno de la inversión (ROI) de la automatización o automatizar algo que desafía las soluciones actuales, haga que todo su equipo piense en nuevos experimentos para probar.

La historia de Lisa

Nuestro equipo estaba escribiendo una nueva versión de nuestra API. Cada vez que probamos una historia recién entregada, encontramos requisitos faltantes. También hubo múltiples fallas, a menudo con problemas de autorización y condiciones límite. Terminamos rechazando las historias varias veces, lo cual resultó costoso.

Discutimos este problema y nos dimos cuenta de que las historias fueron escritas para una audiencia de programadores que estaban familiarizados con versiones antiguas de la API.

Sin embargo, los programadores contratados recientemente estaban codificando las historias y no estaban familiarizados con todos los detalles de cómo debería funcionar la API.

Decidimos que la mejor manera de expresar ejemplos serían pruebas de aceptación que se automatizarían a medida que se codificara la historia. Necesitábamos un marco, así que me uní a un desarrollador para mejorar un marco de estilo especificación por ejemplo (SBE). En retrospectiva, deberíamos haber tenido dos parejas, cada una probando un enfoque diferente, aprovechando los beneficios del desarrollo basado en conjuntos. Aún así, pensé que se nos ocurrió un lenguaje específico de dominio (DSL) interesante, relacionado con el enfoque RSpec que nuestro equipo estaba usando para el desarrollo basado en pruebas (TDD).

Desafortunadamente, cuando mostramos nuestros scripts al resto del equipo, les preocupaba que nuestras pruebas, que probaban la API a través de la capa HTTP, tuvieran demasiada duplicación con las pruebas funcionales RSpec escritas como parte del proceso TDD. Personalmente no estuve de acuerdo, pero sentí que tenía que seguir los deseos del equipo.

Todavía quería ayudar a guiar la codificación en el nivel de aceptación o prueba de historia, así que escribí pruebas en forma de pseudocódigo, usando un DSL similar al que habíamos probado para nuestro primer esfuerzo SBE. Pude conseguir que un programador se asociara conmigo para automatizar mis casos de prueba en este DSL modificado. ¡Encontramos tres errores en el proceso de automatización!

Una vez que estuvieron funcionando, especifiqué pruebas de manera similar para un punto final de API diferente. Los programadores utilizaron estas pruebas mientras trabajaban en las historias, las encontraron útiles e hicieron sugerencias para mejorárlas, junto con casos de prueba adicionales. El número de rechazos de historias de API disminuyó, lo que demuestra el valor de utilizar pruebas para ilustrar los requisitos.

Sin embargo, algunos de los programadores descubrieron que la curva de aprendizaje para el DSL modificado les suponía una sobrecarga adicional. Aunque era más legible para los evaluadores, había suficiente diferencia con sus propias pruebas RSpec como para que pareciera un cambio de tarea. Sintieron que los estaba frenando demasiado. La cultura de nuestro equipo era que la pareja que escribía el código también automatizaba las pruebas en todos los niveles. Cambiar de DSL parecía demasiado caro, por lo que quisieron utilizar el mismo formato RSpec para las pruebas SBE.

Una desventaja de este enfoque era que las pruebas no podían pasar por la capa HTTP, por lo que algunas pruebas no podían automatizarse. Este enfoque también significaba que no podía participar en la automatización de la prueba, pero necesitábamos un enfoque con el que todos se sintieran cómodos.

Continué especificando casos de prueba usando un formato de pseudocódigo, documentándolos en la wiki del equipo y vinculando esas pruebas a la historia. Luego, la pareja de programadores automatizó estas pruebas utilizando el mismo marco que para sus pruebas unitarias. Este seguía siendo un tipo de enfoque SBE y proporcionó valor, aunque a mí me pareció torpe. Además, los casos de prueba que requirieron ir

a través de la capa HTTP no se pudo automatizar, lo que provocó agujeros en el conjunto de pruebas de regresión automatizadas. Tenemos que hacer más controles manuales para compensar, y esto constituye probar la deuda técnica, lo que nos ralentiza.

Continué esta práctica para las historias de puntos finales principales, pero me pareció demasiado pesada para las historias más pequeñas. Pasé a enumerar casos de prueba en la historia misma, pero descubrí que a veces los programadores los pasaban por alto y la historia entregada era rechazada. En mi experiencia, si los programadores no incluyen la automatización de las pruebas de aceptación como parte de su esfuerzo de desarrollo, a menudo no cumplirán esos requisitos.

Descubrí que después de esta serie de experimentos, los programadores pensaron en más casos de prueba por su cuenta mientras codificaban usando TDD. Guiar el desarrollo con pruebas orientadas al cliente es nuevo en la cultura empresarial y se necesitarán más debates y experimentos para encontrar el enfoque correcto.

En nuestros esfuerzos diarios por ofrecer valor a nuestros clientes, puede resultar difícil tomarse el tiempo para probar algo nuevo. Sin embargo, esa inversión vale la pena para mejorar nuestra capacidad de incorporar calidad y valor a nuestros productos de software a lo largo del tiempo.

Lograr un consenso de equipo para soluciones de automatización

Lograr que un equipo, especialmente uno grande, se ponga de acuerdo sobre un marco de automatización u otra solución puede resultar complicado. Por doloroso que pueda ser, cuando te enfrentas a un problema difícil, todo el equipo necesita discutirlo y proponer experimentos para tratar de encontrar una solución.

La historia de Lisa

Durante ocho años, mi equipo anterior estuvo satisfecho con nuestro marco de automatización de interfaz de usuario (UI). Nuestras pruebas mantuvieron las fallas de regresión fuera de producción y, aunque podrían haberse diseñado mejor, era razonable mantenerlas.

Pero necesitábamos algunas funciones nuevas en nuestra interfaz de usuario para ayudar a evitar que los usuarios cometieran errores que hicieran que nuestro equipo perdiera un tiempo valioso en el soporte de producción. Los programadores decidieron que deberían usar Ajax para implementar estas características, pero nuestra herramienta de prueba no funcionaba con el código del lado del cliente. Teníamos nuestra propia regla de nunca publicar código que no fuera compatible con pruebas de regresión automatizadas, por lo que, aunque la solución de codificación estaba lista, tuvimos que posponer la entrega hasta que solucionáramos el problema de la automatización de pruebas.

Investigamos algunos marcos y controladores potenciales de automatización, pero necesitábamos un plan de acción. Reuní al equipo para una reunión para decidir

qué hacer a continuación. Después de discutir el problema por un rato, hubo un largo y doloroso silencio. Estaba sentado pensando: "Sí, le digo a otras personas cómo resolver problemas de automatización, pero ni siquiera puedo conseguir que mi propio equipo hable de ello".

Reuní coraje y enumeré algunas de nuestras opciones potenciales. Podríamos contratar a un entrenador experimentado para que venga y nos diga qué marco y controlador de prueba de UI usar y nos enseñe cómo usarlo. Podríamos seguir buscando controladores y marcos por nuestra cuenta.

Finalmente, desde detrás de una pared cónica (ver Figura 17-3), habló nuestro administrador del sistema (a quien no había invitado a la reunión). "Creo que WebDriver hará lo que necesitamos. Haré un pincho". Todos nos sentimos aliviados. ¡Por fin un plan!

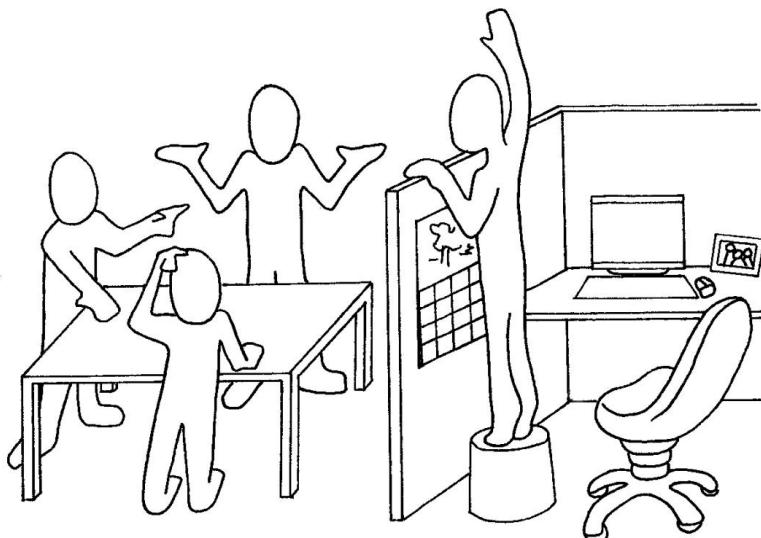


Figura 17-3 Invitar a las personas adecuadas

El pico tuvo éxito y pasamos a los "bake-offs" para determinar qué marco funcionaría mejor para nosotros con WebDriver. Todo fue una gran inversión, pero la inversión más importante fue esa dolorosa reunión inicial.

Incluso cuando crea que su equipo cuenta con una buena cobertura de pruebas de regresión, las necesidades cambiantes de los clientes y las nuevas tecnologías pueden requerir un nuevo enfoque. Ya sea que esté dando sus primeros pasos para satisfacer las necesidades de automatización o considerando un cambio a un enfoque que anteriormente había tenido éxito, asegúrese de invitar a las personas adecuadas para que le ayuden a abordar el problema.

problema. El enfoque de todo el equipo funciona cuando hay diversidad de habilidades y perspectivas. Y cuando todo lo demás falla, celebre sus reuniones al alcance del oído de las personas que olvidó invitar.



¿Cuál es el mayor problema de tu equipo en este momento? ¿Está relacionado con la automatización? Tal vez necesite una forma de identificar cuellos de botella en el rendimiento o agujeros en la seguridad. Reúna a las personas interesadas. Tormenta de ideas. Haz una prueba de concepto. Experimente con algunos horneados. Busque experiencia externa si es necesario. Dedique tiempo a aprender y, sí, ¡a menudo esto es extremadamente difícil de hacer! Consiga la ayuda de un compañero de equipo con ideas afines y recuerde el valor del valor del evaluador ágil.

Sí, es difícil dar un paso atrás y pensar en formas de solucionar un problema difícil cuando sus clientes quieren las funciones X, Y y Z.

Planifique siempre su carga de trabajo para tener tiempo de avanzar en áreas que son cruciales para el éxito a largo plazo de su equipo, como las pruebas automatizadas adecuadas.

¿ Cuánta automatización es suficiente?

En el Capítulo 14 de Pruebas ágiles, dimos ejemplos de actividades de prueba que se benefician de la automatización, así como aquellas que necesitan participación humana o donde la automatización es difícil. A lo largo de los años, hemos disfrutado de muchos marcos y controladores de prueba nuevos y mejorados que nos ayudan a utilizar la automatización de más formas y con menos molestias. Por ejemplo, podemos escribir y administrar nuestro código de prueba en los mismos entornos de desarrollo integrados (IDE) y sistemas de control de código fuente que nuestro código de producción.

A medida que los equipos producen más código de producción y el código de prueba que lo acompaña, nos enfrentamos a un nuevo problema, especialmente con respecto a las comprobaciones de regresión automatizadas. La integración continua (CI) debería proporcionar información rápida si algún cambio ha introducido un fallo de regresión. Pero a medida que agrega nuevas pruebas de regresión en cada iteración, los trabajos de creación del conjunto de pruebas toman cada vez más tiempo. Es fácil perder ese breve ciclo de retroalimentación.

Podemos acelerar nuestras compilaciones de varias maneras, tal vez usando máquinas virtuales para ejecutar muchos conjuntos de pruebas en paralelo. Aun así, tenemos que investigar cada prueba fallida. Más pruebas significan más posibilidades de fallas que no son una regresión en el código sino un problema con la prueba en sí. Más pruebas significan una base de código de prueba más grande que requiere refactorización y actualización para mantenerse actualizado con el código de producción.

La historia de Lisa

Nuestro equipo luchó con el CI en rojo durante días. Al menos un par se dedicó al 100% cada día a hacer que las construcciones fueran ecológicas. Este fue un obstáculo importante para que las historias se entregaran, probaran y aceptaran. Y los verdaderos fracasos de la regresión a menudo no se identificaban hasta pasadas horas o días. Los mayores problemas se produjeron con las pruebas a través de la interfaz de usuario, que fallaron espuriamente por varias razones. Este fue un tema constante de discusión en las retrospectivas del equipo, pero las soluciones fueron difíciles de alcanzar.

El gerente de desarrollo creó un documento compartido donde los miembros del equipo podían intercambiar ideas sobre formas de abordar la construcción de CI, que lleva mucho tiempo y a menudo falla. Fuimos invitados a compartir problemas, inquietudes, preguntas, dilemas y realidades, así como posibles soluciones o mejoras.

Esto se convirtió en una hoja de cálculo. Para cada idea, enumeramos beneficios, desventajas, riesgos y notas sobre la complejidad. Un pequeño grupo de programadores y evaluadores interesados con una variedad de habilidades se reunieron para discutir cada idea. Calificamos cada idea según sus beneficios y complejidad, utilizando una escala de puntos del 1 al 10.

Luego restamos la puntuación de complejidad de la puntuación de beneficio.

Utilizando los resultados, identificamos los “frutos más maduros” y escribimos historias para hacerlo. Algunas eran bastante simples; por ejemplo, comenzamos a reiniciar las cajas de CI todas las noches para ayudar a garantizar que las compilaciones siempre comenzaran con un entorno limpio y sin procesos obsoletos o bloqueados.

Los navegadores en los entornos de prueba que ejecutaron las pruebas de un extremo a otro fueron sujeto a actualizaciones automáticas de la versión del navegador que a menudo causaban fallas en las pruebas. Se creó un segundo conjunto de entornos de prueba, donde se cambió la configuración del navegador para evitar actualizaciones automáticas. Eso hizo que fuera más fácil saber si una actualización automática del navegador provocó una falla en la prueba.

Otra idea fue crear una prueba de humo de corta duración que verificara las partes más críticas de la interfaz de usuario y que solo tardara un par de minutos en ejecutarse. El conjunto completo de pruebas de UI aún se ejecutaría, pero la implementación automatizada en el entorno de prueba se basaría en el éxito de la prueba de humo en

Además de las pruebas unitarias y a nivel funcional.

Estas acciones relativamente simples y de alto beneficio mejoraron la estabilidad de los conjuntos de pruebas de la interfaz gráfica de usuario (GUI). Las historias recién terminadas ahora se implementan con mayor regularidad, lo que equilibra nuestra carga de trabajo y facilita los lanzamientos regulares.

Es valioso utilizar la automatización para ayudar a verificar cada caso límite y condición límite durante el desarrollo. Sin embargo, dependiendo del sistema bajo prueba (SUT) y los riesgos involucrados, puede que no tenga sentido ejecutar todas las permutaciones de cada caso de prueba en el CI. Es posible que no sea necesario agregar todos los casos de prueba ejecutados durante la codificación de una historia al conjunto de regresión.

Es posible que tenga un conjunto particular de pruebas de integración seleccionadas que tardan en ejecutarse, pero es suficiente ejecutarlas una vez al día o quizás justo antes del lanzamiento. Dependiendo de su situación, lo mismo puede aplicarse a las pruebas de rendimiento, pruebas de interfaz de usuario o cualquier prueba que pueda ser inherentemente lenta y pueda ejecutarse con menos frecuencia sin demasiado riesgo.

Soluciones colaborativas para elegir herramientas

Cuando decide qué herramientas elegir, hay muchas consideraciones a la hora de evaluar una determinada. En el Capítulo 14 de Pruebas ágiles, recomendamos abordar una herramienta a la vez, identificar sus requisitos y decidir qué tipo de herramienta elegir o construir que se ajuste a sus necesidades e infraestructura. Desde que escribimos esto, hemos aprendido aún más sobre buenas formas de elegir los marcos y controladores de prueba adecuados, así como otras herramientas, para su equipo. Como mencionamos en el Capítulo 16, "Patrones y enfoques de diseño de automatización de pruebas", su equipo primero debe decidir cómo quiere que se vean sus pruebas automatizadas y luego encontrar herramientas que lo respalden. Si elige la herramienta antes de saber cómo quiere expresar sus pruebas, automáticamente habrá eliminado muchas de sus opciones.

Una de las características más importantes de un marco de prueba es si fomenta la colaboración entre los diferentes roles de un equipo.

En el Capítulo 16, "Patrones y enfoques de diseño de automatización de pruebas", hablamos sobre el método de prueba que une las pruebas al código de producción y la importancia de la colaboración entre programadores y evaluadores.

Es esencial utilizar herramientas que permitan y mejoren esta colaboración.

La tabla 17-1 muestra algunas de las diferencias entre lo que consideramos herramientas colaborativas y aquellas que no lo son.

El único comentario negativo que hemos escuchado de algunas personas es que cuando los programadores dedican tiempo a trabajar en pruebas, ese tiempo no lo dedican a escribir código de producción. Sin embargo, los programadores que trabajan de la manera que hemos descrito nos han dicho que les ayuda a escribir código mejor, más comprobable y con menos defectos.

Ampliación de la automatización a grandes organizaciones

En las organizaciones grandes, es típico tener varios equipos trabajando en la misma línea de productos, a menudo cinco, seis o más. Esto tiene repercusiones para

Tabla 17-1 Herramientas colaborativas versus no colaborativas

Herramientas colaborativas	Herramientas no colaborativas
Permitir que los evaluadores/empresas definan pruebas	Las pruebas generalmente se realizan a través de la interfaz de usuario.
El código de prueba puede estar en un lenguaje de programación común.	Los programadores no suelen estar dispuestos a ayudar
Los programadores pueden ejecutar pruebas mientras codifican. Las pruebas se implementan después de que se escribe el código.	escrito
Los evaluadores pueden pedir ayuda a los programadores	Los evaluadores crean e implementan todas las pruebas.
Haga que las pruebas sean visibles para que los miembros del equipo puedan discutir la capacidad de prueba antes de codificar.	

cómo pensamos sobre muchos aspectos de las pruebas del producto. En esta sección nos concentraremos en formas de escalar la automatización para productos grandes con varios equipos.

Los estándares de prueba comunes, incluidas las convenciones de nomenclatura, se vuelven aún más críticos cuando varios equipos trabajan en una base de código compartida. Es posible que diferentes equipos estén tocando diferentes partes del código de producción y necesiten sentirse cómodos cambiando las pruebas y el código de prueba detrás de ellas.

También es necesario que haya coherencia en la selección de herramientas. Si forma parte de un pequeño equipo que trabaja en un solo producto, seleccionar las herramientas que mejor se adapten a sus necesidades es relativamente sencillo. Cuando es necesario considerar otros equipos, y tal vez un equipo de soporte y mantenimiento, existe una necesidad imperiosa de herramientas comunes.

Una forma de lograr esto es definir clases de herramientas que sean aceptables para diferentes necesidades. Por ejemplo, si está en un equipo que trabaja en código Java en el producto X, puede tener una llamada estándar a Selenium para sus pruebas de UI, una opción de Cucumber o FitNesse para sus pruebas de nivel API (servicio) y JUnit. para sus pruebas unitarias. Esto le da al equipo cierta autonomía pero permite coherencia entre el equipo de mantenimiento y otros equipos que trabajan en el mismo producto.

La barra lateral de Cirilo Wortel anteriormente en este capítulo ilustra cómo un conjunto común de herramientas, pautas y creación de comunidades promueven la automatización de pruebas exitosa en una gran organización. Veamos otro ejemplo.

Desafíos de la transición: automatización ágil de pruebas

Geoff Meyer ,

Arquitecto de Dell Inc., explica cómo una prueba de su gran organización de desarrollo encontró más adecuado un enfoque de automatización.

Probar

A medida que Dell Enterprise Solutions Group (ESG) hizo la transición al desarrollo ágil, uno de nuestros proyectos en particular ilustró algunos de los desafíos de prueba inherentes a dicha transición. El equipo del proyecto estaba integrado por programadores y evaluadores de primer nivel, uno de nuestros principales gerentes de desarrollo y probablemente uno de los mejores y más comprometidos propietarios de productos de la organización. Como ocurre con la mayoría de nuestros proyectos, este también contaba con una extensa matriz de pruebas de hardware. El equipo reconoció que la automatización de pruebas era un factor crítico de éxito y tomó medidas para garantizar que, a medida que se desarrollaran las funciones, se implementara un conjunto de regresión automatizada para garantizar que el software funcionara en toda la amplia matriz de compatibilidad.

Desafortunadamente, una de las cosas que finalmente ralentizó la capacidad del proyecto para agregar funciones fue la filosofía de automatización de pruebas. El administrador de pruebas y los miembros de pruebas responsables del esfuerzo trajeron consigo muchos años de experiencia, la mayoría de los cuales se basaron en pruebas de caja negra y automatización de UI en proyectos basados en cascada. Aunque los evaluadores del equipo hicieron la transición rápidamente a los ciclos de sprint colaborativos de dos semanas basados en Scrum, su enfoque de prueba permaneció firmemente basado en una metodología de prueba basada en la interfaz de usuario.

A medida que se acercaba la fecha de lanzamiento de la primera versión, el equipo se encontró en modo de ponerse al día para actualizar los scripts de automatización basados en la interfaz de usuario para adaptarse a los cambios más recientes en la interfaz de usuario en los sprints finales. Como resultado, el equipo tuvo que realizar pruebas manuales inesperadas mientras actualizaba simultáneamente las pruebas automatizadas de la interfaz de usuario. En el lado positivo, el conjunto de pruebas de automatización basado en la interfaz de usuario permitió un conjunto de regresión automatizada. Sin embargo, debido a que el conjunto de regresión en funcionamiento estaba dos o tres sprints por detrás de la finalización de las historias de los usuarios, el proyecto incurrió en una prueba manual mucho más larga de lo que se había planeado originalmente.

Revisamos la filosofía de automatización de pruebas para el proyecto en la versión de seguimiento. Debido a que la aplicación utilizaba una arquitectura modelo-vista-controlador (MVC), propusimos un cambio en el enfoque de automatización de pruebas desde el nivel de servicio basado en UI al nivel de servicio SOAP (Protocolo simple de acceso a objetos). Debido a la relativa estabilidad de las interfaces SOAP versus la interfaz UI, estábamos seguros de que los miembros de prueba del equipo Scrum podrían desarrollar la automatización de pruebas en el sprint a nivel de servicio, minimizando así la necesidad de pruebas adicionales en

final del cronograma. La resistencia a este cambio de enfoque fue captada de manera bastante sucinta por uno de los miembros de prueba del equipo Scrum: "Si automatizamos los servicios web y la automatización de la interfaz de usuario, entonces estamos haciendo el doble de esfuerzo".

No pude convencer al equipo antes del inicio del próximo lanzamiento.

que lo que en realidad se proponía era una compensación. En lugar de realizar la automatización de la interfaz de usuario dentro del sprint, el equipo centraría sus esfuerzos de prueba durante el sprint en la automatización a nivel de servicio y las pruebas manuales de la interfaz de usuario. La segunda versión tuvo resultados similares, pero el problema estaba empeorando. Debido a que el trabajo pendiente de automatización estaba creciendo, se tuvo que asignar un mayor número de evaluadores para completar las pruebas manuales. Antes de este lanzamiento, la proporción entre programadores y evaluadores era de 3:1. Durante este lanzamiento del proyecto, esa proporción comenzó a acercarse a 1:1.

Mientras el equipo se preparaba para la tercera versión, intenté otra vez alterar el enfoque del equipo para la automatización de pruebas. Asignamos a uno de nuestros evaluadores de automatización de nivel de servicio con más experiencia para profundizar en la aplicación con el objetivo de demostrar cómo se podría utilizar la automatización de nivel de servicio para ayudar al equipo a mantener la automatización de pruebas al ritmo de los nuevos desarrollos. Desafortunadamente, después de dos lanzamientos completos, la falta de atención por parte del equipo en los aspectos de capacidad de prueba y automatizabilidad debajo de la interfaz de usuario llevó a que se incorporara más lógica de negocios y manejo de errores en la capa de la interfaz de usuario en lugar de debajo de la interfaz de servicios. Después de un análisis detallado, el equipo empresarial decidió refactorizar la lógica empresarial y la aplicación de manejo de errores en el transcurso de las próximas versiones para permitir aún más un enfoque de automatización de pruebas basado

Hubo varias lecciones que aprendimos de esta experiencia.

En primer lugar, una estrategia de prueba basada predominantemente en la automatización de la interfaz de usuario probablemente genere tiempo y esfuerzo no planificados para fortalecer el producto al final del ciclo. Además, incluso cuando los miembros de los equipos Scrum tienen una comprensión compartida de la importancia de la automatización de pruebas, es igualmente importante determinar los tipos apropiados de automatización (UI o nivel de servicio) dentro de la aplicación y el momento adecuado para emplearlos. Por último, el reconocimiento de que la arquitectura de la aplicación puede ser un facilitador o un inhibidor de la estrategia de automatización de pruebas de un proyecto es una lección importante que se debe aplicar en todas las etapas del ciclo de vida de un producto.

Una automatización de pruebas insuficiente perjudica a su producto y a sus usuarios, y el riesgo de que los fallos de regresión pasen a producción es mayor. Si su equipo tiene una política para lanzar solo aquellas funciones respaldadas por pruebas de regresión automatizadas, la implementación de nuevas funciones que podrían hacer que el producto sea más utilizable o valioso puede verse obstaculizada por la incapacidad de

automatizar pruebas. Es posible que sea necesario refactorizar la arquitectura del producto para admitir la capacidad mejorada de automatización.

Otras consideraciones de automatización

Hasta ahora nos hemos centrado en la automatización de pruebas funcionales. La automatización también es Se requieren para la mayoría de las pruebas del Cuadrante 4, las pruebas tecnológicas que critican el producto, como pruebas de rendimiento, carga, escalabilidad, capacidad, confiabilidad y facilidad de instalación. También se necesitan herramientas para monitorear y verificar la gestión de la memoria. Luego están todas las herramientas necesarias para CI, gestión de configuración, control de versiones, canal de implementación y gestión de datos. Consulte la Parte VI, "Automatización de pruebas" y las secciones "Herramientas" de la bibliografía para obtener fuentes que lo guiarán en esas áreas.

Las pruebas automatizadas son una serie de comprobaciones, pero las utilizamos primero para ayudar a guiar el desarrollo con ejemplos orientados al cliente convertidos en pruebas. Lo necesitamos como documentación viva para mostrar cómo se comporta nuestro sistema. Lo necesitamos para informarnos rápidamente de los fracasos de la regresión. Lo necesitamos para mantener la deuda técnica a un nivel manejable y para garantizar que tengamos suficiente tiempo para todas las valiosas actividades de prueba que ayudan a entregar software de alta calidad.

Incluso si su equipo cuenta con sólidos conjuntos de pruebas automatizadas, esté preparado para desafíos continuos a medida que su producto y la tecnología que utiliza evolucionan. Esté alerta a las innovaciones que pueden ayudar con problemas difíciles de automatización. Por ejemplo, el equipo de Lisa descubrió recientemente un problema de producción que se pasó por alto a pesar de la amplia cobertura de pruebas unitarias. Para encontrar un mejor enfoque de prueba de regresión para esto, uno de los desarrolladores experimentó con pruebas generativas. En una prueba generativa, usted define propiedades esperadas para los resultados esperados en lugar de valores codificados, y el ejecutor de la prueba genera datos aleatorios para verificar. Incluso "reduce" automáticamente las fallas de las pruebas al caso mínimo de falla (Kemerling, 2014). Las nuevas pruebas encontraron fallas reproducibles para que el código pudiera corregirse. Estos no reemplazaron las pruebas unitarias, pero brindaron confianza adicional.

Utilice el enfoque de todo el equipo, trabaje siempre en su mayor problema y sea paciente. No es necesario que su automatización sea perfecta para ser valiosa y debe mejorarlía continuamente.

Resumen

No existe una única solución de automatización de pruebas adecuada. Ser paciente; Involucra a todo el equipo en experimentos para encontrar buenas soluciones para su equipo. Utilice los modelos piramidales descritos en el Capítulo 15, "Pirámides de automatización", como ayuda. En este capítulo, analizamos algunos de los pasos más desafiantes en la implementación de una valiosa automatización de pruebas. Algunos puntos clave:

- Los equipos nuevos en Agile se benefician de contar con un profesional experimentado.
Un profesional que puede ayudar a seleccionar soluciones, encontrar formas de medir el progreso, establecer estándares entre equipos y crear una comunidad para compartir experiencias.
- Involucrar a todo el equipo en la selección de soluciones de automatización. Puede hacer falta valor para llegar a un consenso sobre las decisiones sobre cómo proceder. Concéntrese en probar experimentos, sabiendo que no existe una única solución "correcta".
- Sea prudente al incluir lo que incluye en los conjuntos de pruebas automatizados y Mantenga las compilaciones rápidas para obtener comentarios oportunos.
- Intente encontrar marcos de automatización que promuevan la colaboración entre evaluadores, codificadores y profesionales con otras especialidades en el equipo.

- Tómese el tiempo para aprender juntos y desarrollar las soluciones técnicas; ahí es donde se crea gran parte del valor.
- Llevar la automatización al nivel más bajo que tenga sentido. Refaccionar Arquitectura de código Tor si es necesario para permitir pruebas rentables. automatización.

Esta página se dejó en blanco intencionalmente.

Parte VII

¿Cuál es su contexto?

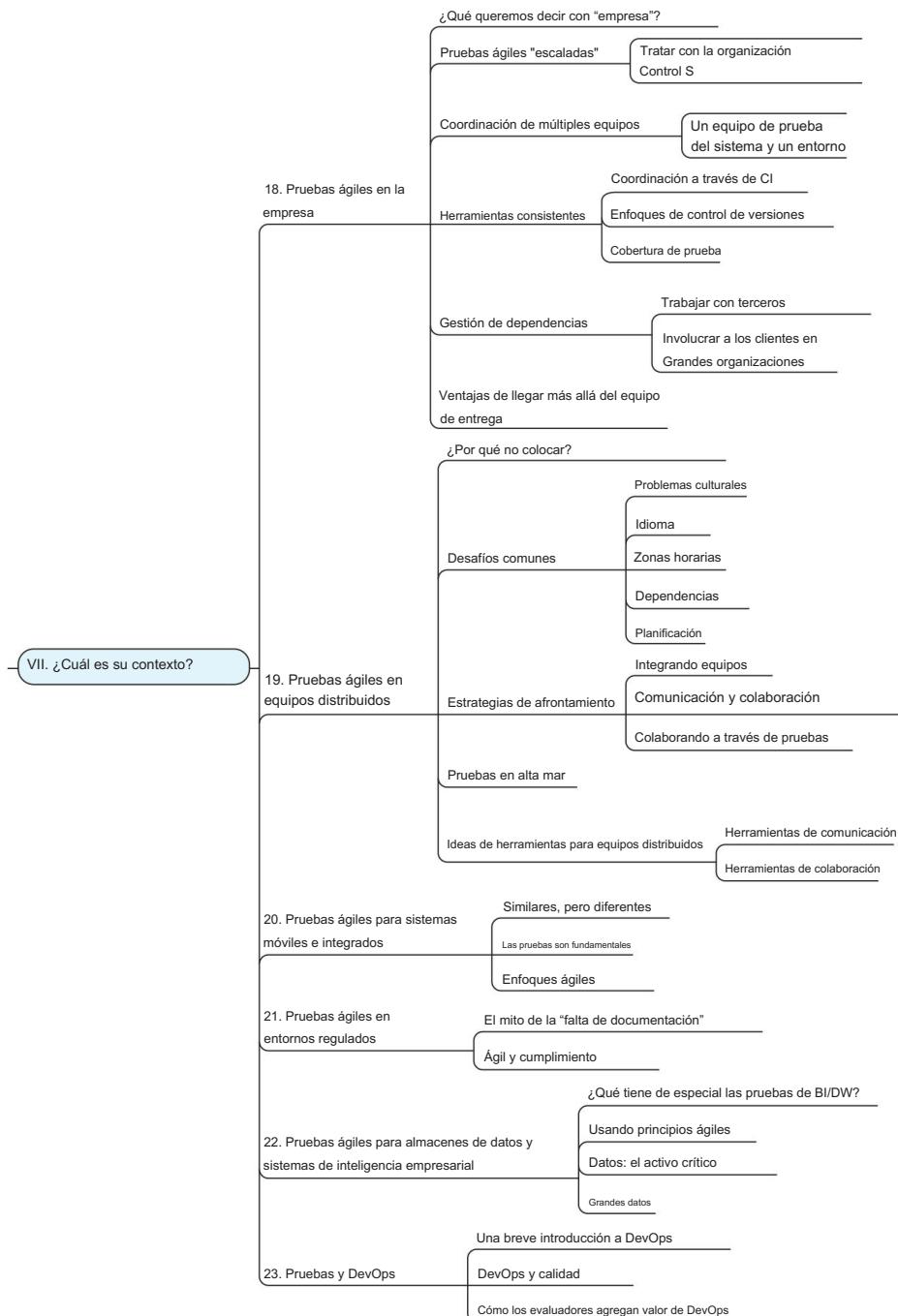
El contexto cuenta. En esta sección presentaremos un conjunto de desafíos específicos que tal vez no se apliquen a todos. Hemos recopilado muchas historias de profesionales expertos que trabajan en estas diversas áreas para ayudarle.

El desarrollo ágil se ha extendido a grandes organizaciones con soluciones para toda la empresa. Comenzamos esta parte del libro analizando los desafíos particulares de “ampliar” las pruebas ágiles. El capítulo 18 se centra en la historia de cómo una de esas organizaciones hizo la transición de sus pruebas a la metodología ágil.

Las organizaciones grandes y pequeñas se encuentran cada vez más repartidas entre diferentes lugares, tal vez por todo el planeta. Por lo tanto, nuestro próximo capítulo trata sobre las pruebas en equipos distribuidos. Compartiremos estrategias para hacer frente a las barreras lingüísticas, culturales y tecnológicas para mantener a todos involucrados en las pruebas antes, durante y después de la codificación.

Otras áreas que se han expandido dramáticamente en los últimos años son las aplicaciones móviles y los sistemas integrados. Estas aplicaciones tienen altos riesgos, tanto para la seguridad como para el éxito, en un mercado obstinado y que cambia rápidamente. También plantean desafíos de prueba únicos. El Capítulo 20 analizará lo que necesita probar y cómo encajar todas esas pruebas en ciclos de lanzamiento cortos utilizando un enfoque de equipo completo.

Otro dominio que ha desconcertado a los equipos que quieren adoptar la metodología ágil es el software regulado. Muchos equipos sienten que las limitaciones de cumplimiento y documentación son mutuamente excluyentes con el desarrollo ágil. Compartiremos algunas historias de éxito en el Capítulo 21 que muestran qué tan bien la metodología ágil puede satisfacer las demandas de las agencias reguladoras. Los evaluadores tienen oportunidades únicas de contribuir con su experiencia en el campo y sus habilidades de colaboración para ayudar a crear productos que deleiten a los clientes y satisfagan a los reguladores.



Las pruebas de almacenes de datos y sistemas de inteligencia empresarial en entornos ágiles también se benefician de ciclos de retroalimentación cortos y un enfoque incremental. En el Capítulo 22, compartiremos formas de exponer tempranamente los problemas de calidad de los datos. También discutiremos las habilidades técnicas especializadas y el conocimiento del dominio empresarial necesarios para verificar los datos utilizados para las decisiones comerciales.

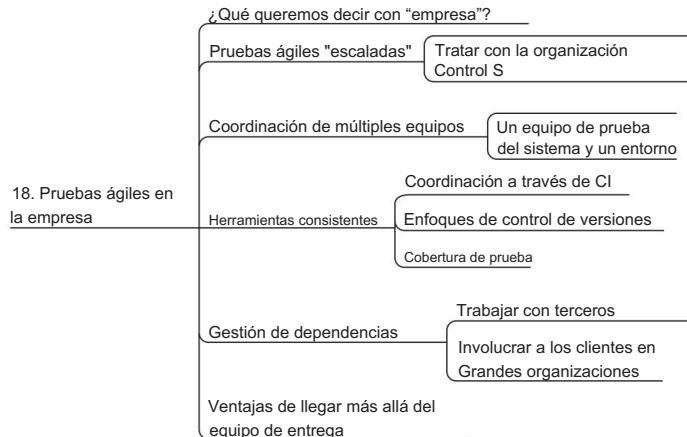
El término DevOps ganó popularidad a partir de 2009, pero los profesionales con experiencia en operaciones y desarrollo han colaborado con los evaluadores y otros miembros del equipo durante muchos años. En nuestro capítulo sobre pruebas y DevOps, mostraremos cómo las pruebas y DevOps funcionan juntos.

- Capítulo 18, "Pruebas ágiles en la empresa"
- Capítulo 19, "Pruebas ágiles en equipos distribuidos"
- Capítulo 20, "Pruebas ágiles para sistemas móviles e integrados"
- Capítulo 21, "Pruebas ágiles en entornos regulados"
- Capítulo 22, "Pruebas ágiles para almacenes de datos y empresas
Sistemas de Inteligencia"
- Capítulo 23, "Pruebas y DevOps"

Esta página se dejó en blanco intencionalmente.

Capítulo 18

Pruebas ágiles en la empresa



Escuchamos mucho hablar sobre escalar ágilmente. Desde los primeros días de la metodología ágil, algunos expertos han insistido en que los principios y valores ágiles son sólo para equipos pequeños y ubicados en el mismo lugar. Algunos profesionales creían que las grandes organizaciones empresariales ni siquiera deberían intentar el desarrollo ágil. Sin embargo, muchos han intentado una transición ágil y hemos escuchado muchas historias de éxito. El desafío no es tanto escalar la agilidad como tratar de mantener los principios de agilidad en el contexto de organizaciones que exigen disciplina y prácticas estrictas en sus equipos. Este capítulo no pretende decirle cómo escalar ágilmente en grandes organizaciones. Para eso, le sugerimos buscar en la bibliografía de la Parte VII, “¿Cuál es su contexto?”, libros sobre ese tema. Nos centramos en buenos enfoques para pruebas ágiles en empresas empresariales.

¿Qué queremos decir con “empresa”?

En primer lugar, no nos referimos al Starship Enterprise, aunque a menudo sentimos que estamos explorando nuevos mundos y civilizaciones extrañas. Usamos el término empresa para referirnos a unidades de negocios o unidades de negocios grandes, que lo abarcan todo y que lo abarcan todo.

geografías, el conjunto de sus operaciones comerciales. "Empresa" se aplica a más que grandes empresas. Las universidades, las organizaciones sin fines de lucro y los gobiernos son ejemplos de organizaciones enormes que necesitan software para toda la empresa para satisfacer una variedad de necesidades. Estas organizaciones pueden estar distribuidas, pero no necesariamente. El capítulo 19 trata sobre pruebas en equipos distribuidos, por lo que en este capítulo nos concentraremos en sistemas de aplicaciones grandes, que generalmente significan organizaciones grandes.

Independientemente del tamaño de la organización, la atención debe centrarse en el cliente. En 2009, Mark P. McDonald dio un pequeño consejo que creemos que es válido para cualquier empresa que implemente principios ágiles y eficientes (McDonald, 2009):

Pon al cliente en el centro de la empresa, porque es lo único que todos compartéis. Suena trillado pero es cierto. Busque reducir las barreras o impedimentos a las operaciones internas que reducen el servicio al cliente o dificultan que el cliente interactúe con todas las partes de la empresa. Después de todo, como empresa, su modelo económico se basa en la capacidad de involucrar y atender al cliente en todas sus operaciones.

Pruebas ágiles "escaladas"

En su discurso de apertura en Agile Testing Days 2013 (Hassa, 2013), Christian Hassa habló sobre la ampliación del desarrollo basado en pruebas (TDD) a la empresa. Su descripción sonaba sospechosamente similar a lo que se hace en muchos equipos ágiles. Recomendó que los equipos establezcan una meta deseada, determinen las partes interesadas, definan un cambio de comportamiento deseado, determinen los resultados, escriban una prueba de aceptación fallida, escriban el código para que esa prueba pase usando TDD, etc. Esto suena como el programa "Discutir—Destilar—Desarrollar—Demo" ciclo de desarrollo basado en pruebas de aceptación (ATDD) descrito por Elisabeth Hendrickson (Hendrickson, 2008).

Christian recomendó que las empresas midan el impacto del sistema implementado y refinen el resultado en consecuencia. Puede perfeccionar continuamente su estrategia en función del cambio de comportamiento que realmente se produzca, lo que eleva sus pruebas. Su conclusión fue que escalar no se trata de cómo hacer más trabajo con más personas. Más bien, vale la pena probar los objetivos y los impactos lo antes y con mayor frecuencia posible, en organizaciones de cualquier tamaño.

Christian aplica el lenguaje de planificación de Tom Gilb, Planguage (Gilb, 2013), para definir objetivos. En el proceso de definición de objetivos, seleccionas una escala: decides qué medir. A continuación, eliges una métrica: cómo medirás. Establezca niveles comparando primero la situación actual, estableciendo restricciones como el resultado mínimo aceptable y definiendo el objetivo o resultado deseado. El mapeo de impacto, el mapeo de historias y la entrega incremental e iterativa pueden ayudarlo a establecer objetivos y lograr el valor comercial deseado. El capítulo 9, “¿Estamos construyendo lo correcto?”, explicó con más detalle el mapeo de impacto y el mapeo de historias. Las mismas técnicas funcionan en empresas de todos los tamaños, pero a medida que las empresas crecen, se necesita más diligencia a la hora de establecer objetivos y evaluar los

Aprendizaje: alinear el establecimiento de objetivos en toda la organización

Eveliina Vuolli, La gerente de desarrollo operativo de Nokia en (Networks Business), comparte las lecciones que aprendió al intentar introducirla automatización en el sistema de una gran empresa.

Toda la organización debe comprender los beneficios de realizar la automatización de pruebas. Por ejemplo, la gestión de productos debería participar para garantizar que no vean la automatización de pruebas sólo como una "práctica divertida de I+D". En lugar de ello, es necesario debatir cómo la automatización de pruebas beneficia al desarrollo de productos; por ejemplo, la automatización mejora la calidad del producto en general y facilita la realización de cambios de última hora en el software. En mi experiencia, es bueno tener ejemplos de cálculos de ROI. Reforzan el mensaje, especialmente cuando se habla de automatización de pruebas con las partes interesadas del negocio.

También es bueno tener un objetivo común para la automatización de pruebas en toda la organización. Esta es una forma de crear un entendimiento común de cuál es el nivel esperado de automatización en diferentes niveles de prueba. Cuando establecimos el objetivo por primera vez, utilizamos el mismo nivel de cobertura porcentual de automatización de pruebas como objetivo para todos los equipos. Sin embargo, debido a que los equipos se encontraban en situaciones muy diferentes, el número objetivo para un equipo era un gran desafío, mientras que otros ya lo habían alcanzado. Aprendimos que es mejor establecer objetivos desde abajo hacia arriba. Aún puedes tener un nivel mínimo definido para todos los equipos, pero por encima de eso, los números objetivo exactos pueden variar. En un establecimiento de objetivos, los objetivos para diferentes áreas en la cobertura de pruebas unitarias variaron entre el 60% y el 95% y para las pruebas de aceptación entre el 50% y el 100% durante un período de tiempo determinado.

Durante las discusiones sobre el establecimiento de objetivos de automatización de pruebas (TA), la organización en su conjunto a veces no logró una comprensión común de la terminología relacionada con las pruebas, lo que generó malentendidos cuando hablábamos sobre el estado de la TA en diferentes áreas. Aprendimos a iniciar el establecimiento de objetivos mediante la definición de la nomenclatura. Me ha resultado útil incluso tener una definición oficial para contar el índice de automatización de pruebas. En principio, es una fórmula muy simple: casos de prueba automatizados/ todos los casos de prueba = relación TA. Con el software heredado es necesario acordar la fórmula y tener en cuenta las versiones antiguas y sus casos de prueba en el proceso. cálculo.

Nos pareció fundamental hacer un seguimiento del tema para que la gente se diera cuenta de que era importante. Si nadie pregunta sobre el tema después del "primer apuro", pierde inmediatez, por eso hacemos el seguimiento mensualmente. Tenemos un panel centralizado para mostrar el estado en línea.

Escuchamos comentarios como: "Pero la proporción de TA en realidad no dice mucho sobre la calidad o la cobertura de las pruebas, incluso si es del 100%". Los equipos a menudo necesitan ayuda para considerar realmente cómo aumentar la cobertura de sus casos de asistencia técnica u optimizar el tiempo de ejecución de las rondas de automatización, para que puedan pasar al siguiente nivel. En lugar de limitarse a observar los números, también empezarán a darse cuenta de cómo beneficiarse de la automatización de las pruebas. Un ejemplo es hacer que las computadoras hagan el "trabajo simulado" para que las personas puedan concentrarse en áreas de prueba más desafiantes o tener una buena red de seguridad para permitir la refactorización y cambios rápidos.

Tratar con los controles organizacionales

Las grandes organizaciones tienden a contar con controles para garantizar la coherencia. Se desarrollan a lo largo de los años a medida que la empresa crece e intenta resolver problemas específicos. Muchas organizaciones se muestran reacias a cambiar o eliminar estos controles porque eso es lo que les ha funcionado en el pasado. Un ejemplo sería la aplicación de modelos de calidad como CMMI (CMMI Institute, 2014) o requisitos de auditoría interna. Otro ejemplo sería la forma en que los silos funcionales trabajan juntos y tienen traspasos firmados. En este capítulo, queremos llevarlo de regreso a lo básico y hacerle pensar en cómo aplicar los valores y principios ágiles.

incluso si se trata de soluciones empresariales. Hablaremos más sobre las restricciones regulatorias en el Capítulo 21, "Pruebas ágiles en entornos regulados".

La historia y la cultura de la organización son las más difíciles de superar. Hemos descubierto que para tener éxito, los equipos a menudo tienen que

demostrar su valía y ser capaz de articular los beneficios que la agilidad puede ofrecer. También debemos poder explicar a la dirección cómo los equipos pueden seguir dándoles confianza en su capacidad de cumplir sin aprobaciones, sin informes constantes y sin las medidas existentes.

La gerencia ha aprendido durante muchos años cómo tomar decisiones basadas en esos informes, y los equipos deben poder ofrecerles una alternativa realista a la antigua estructura de informes. No basta con decir: "Confía en nosotros".

También necesitan circuitos de retroalimentación.

Janet ha descubierto que el uso de herramientas simples pero visibles, como la matriz de pruebas de la Figura 7-6 del Capítulo 7, "Niveles de precisión para la planificación", puede aumentar el nivel de confianza de la gerencia en que las pruebas se están considerando y rastreando.

El viaje de Dell: Parte 1

Geoff Meyer, Arquitecto de pruebas en Dell, comparte el viaje ágil. Su historia es similar a la ha experimentado en Dell. Aquí está la historia inicial de Dell que hemos visto. organizaciones avanzar hacia la agilidad. Está dividido en cinco partes y subiremos Examina cada paso del camino.

El Desafío

En la mayoría de las grandes empresas, desarrollar y probar software no siempre es un proceso ágil. Tal fue el caso del Enterprise Solutions Group (ESG) de Dell en 2008. Los silos funcionales y los procesos de desarrollo en cascada habían creado una cultura arraigada y antagónica de "desarrollo versus prueba". Esta falta de cohesión se vio exacerbada por el hecho de que los equipos estaban ubicados físicamente en diferentes pisos y, a veces, en diferentes zonas horarias. Además, las prácticas de prueba que se habían optimizado durante años de prácticas en cascada se basaban en el enfoque de caja negra, y los conjuntos de habilidades de automatización de pruebas se limitaban a herramientas de automatización de la interfaz de usuario (UI).

El contexto de Agile @ Dell en 2009

- Los proyectos de software abarcaron productos de Sistemas Embebidos y Gestión de Sistemas de Servidores.
- Los silos funcionales habían creado una cultura antagónica y arraigada de "desarrollo versus prueba".

- El desarrollo y las pruebas se ubicaron físicamente en diferentes pisos.
 - La gran matriz de pruebas de compatibilidad de hardware se basó en el soporte de tres generaciones de servidores.
 - Los evaluadores realizaron únicamente pruebas de caja negra.
- Los conjuntos de habilidades de automatización de pruebas se limitaron a las herramientas de automatización de la interfaz de usuario.

El Solución

Ingresan Brian Plunkett y Dave Spott, dos ejecutivos de desarrollo de software, quienes habían llegado a Dell con experiencia previa en metodología ágil. Fue entonces cuando comenzó nuestro viaje ágil, en gran parte gracias al respaldo total de estos dos defensores ágiles.

Dell ESG adoptó por primera vez prácticas ágiles a mediados de 2008. Comenzamos a pequeña escala utilizando un proyecto piloto, con un único equipo formado únicamente por programadores. En los primeros seis meses, los resultados del proyecto piloto fueron suficientes para demostrar a los gerentes de toda la organización (especialmente aquellos en la organización de prueba) que este nuevo enfoque para el desarrollo de software podría tener algún mérito.

Debido a que el equipo de prueba estaba alojado en una organización funcional separada, se requirió una buena cantidad de lobby por parte de Dave y Brian para incorporarlos al redil ágil. Al principio fue difícil venderlo.

¿Cómo podríamos sugerir cambios tan drásticos en la gestión del programa, la dotación de personal del proyecto y, en consecuencia, las pruebas mismas? Inicialmente, hubo resistencia, pero una vez que Test experimentó ser un miembro integral del proceso de desarrollo iterativo, la organización Test se convirtió en uno de los mayores defensores y defensores de la metodología ágil.

Como puede verse en la historia de Geoff hasta ahora, Dell tuvo el mismo problema que muchas organizaciones empresariales al adoptar principios y prácticas ágiles. La organización de pruebas se resistió a la idea de integrarse con el desarrollo. Sin embargo, tuvieron la suerte de contar con dos ejecutivos en el equipo directivo para ayudar a vender la idea. Si la gerencia ha decretado que los equipos de software “deben” volverse ágiles, busque a alguien en el equipo gerencial que sea un patrocinador que respalde los cambios que deben realizarse. Recomendar primero un proyecto piloto para resolver los problemas del proceso. Hable con el equipo directivo sobre los cambios que se deben realizar en toda la organización para tener éxito. Esté dispuesto a hacer compromisos, como informar defectos, pero mantenga retrospectivas periódicas con la gerencia y hable sobre beneficios y compensaciones. Tenga en cuenta que algunos

Los grupos tienen temores comprensibles acerca de realizar cambios fundamentales en el proceso de desarrollo y prueba.

Veamos qué hizo Dell a continuación.

Implementación: obstáculos en el camino: Dell, parte 2

En el primer año completo de adopción ágil, se hizo evidente que las técnicas de prueba tradicionales de prueba de caja negra, la automatización de pruebas basada en la interfaz de usuario y la necesidad de registrar todos los defectos no encajaban bien con la colaboración y el ahorro de tiempo. Pautas en caja de Scrum. Dentro del equipo Scrum, los evaluadores desarrollaban y ejecutaban pruebas manuales en la interfaz de usuario de cada historia de usuario. La única automatización que se estaba desarrollando eran los scripts de prueba en la interfaz de usuario. Para colmo, un indicador de desempeño tradicional de la organización de pruebas era el "número de defectos encontrados"; por lo tanto, los miembros del equipo de prueba Scrum registraron todos y cada uno de los defectos. Como puede imaginar, el resultado fueron tareas de prueba para una historia de usuario que no se completaron dentro del límite del sprint, lo que llevó a historias de usuario remanentes. Además, la automatización de pruebas no pudo mantenerse al día con los cambios posteriores en la interfaz de usuario en los sprints posteriores. Los evaluadores dedicaron más tiempo a mantener casos de prueba que a desarrollar y ejecutar otros nuevos. Finalmente, todo el equipo Scrum se vio abrumado por la sobrecarga administrativa de los defectos de procesamiento.

Y estos fueron solo en cada nivel de equipo Scrum. A nivel organizacional, la automatización de pruebas ya era reconocida como un factor crítico de éxito. Sin embargo, la directiva de "seguir adelante y automatizar" solo agravó los problemas a nivel de equipo, ya que se emplearon diferentes herramientas de automatización de la interfaz de usuario, como Rational Functional Tester (RFT), Ranorex y Telerik, debido a las diferencias en las arquitecturas de los productos. .

Como puedes ver en la historia de Geoff, no todos los problemas son problemas de personas. Debido a las iteraciones cortas y la necesidad de retroalimentación rápida, muchos problemas subyacentes, como la arquitectura o las herramientas, se vuelven muy visibles. Las organizaciones grandes tienden a avanzar más lentamente debido a su estructura y procesos de aprobación, por lo que los cambios se producen lentamente y con dificultad. Las organizaciones grandes a menudo necesitan funciones adicionales, como la de arquitecto de pruebas. Hay muchas partes móviles en una solución empresarial y ningún equipo puede saberlas todas.

Geoff Meyer continúa su historia con Dell y explica cómo resolvieron algunos de los problemas de las pruebas con su adopción ágil.

Evolución ágil: Dell, parte 3

Entré en la organización de pruebas en 2010 como arquitecto de pruebas; Mis objetivos incluían desarrollar la estrategia de prueba ágil para respaldar proyectos a gran escala y mejorar las capacidades de automatización de pruebas. Una lectura esencial para estos desafíos fue la de Lisa Crispin Ágil Pruebas: A Práctico Guía para Probadores y Ágil equipos y Janet Gregory. El auto-

La pirámide de información y los cuadrantes de pruebas ágiles se convirtieron en las bases para el siguiente paso en la evolución de nuestra estrategia de pruebas Agile @ Dell.

Una de las primeras cosas que hicimos fue analizar el estado de capacidad de prueba y automatizabilidad de nuestras arquitecturas de aplicaciones debajo de la interfaz de usuario, como la interfaz de línea de comandos (CLI) o los servicios web SOAP o las capas RESTful. A continuación, comenzamos a capacitar a nuestros ingenieros de pruebas en estas nuevas habilidades. En el caso de proyectos en los que incorporamos nuevas contrataciones, contratamos estas habilidades, así como experiencia ágil.

Luego, deliberadamente, restringimos cualquier automatización de la interfaz de usuario desde los sprints de Scrum. También eliminamos el requisito de registrar todos los defectos y, en su lugar, alentamos a los equipos a colaborar en tiempo real para localizarlos, resolverlos y verificarlos. Finalmente, ordenamos a nuestros evaluadores de Scrum que "automatizaran primero" a nivel de servicio, y trabajamos con el propietario del producto y los equipos de desarrollo para incorporar la automatización de pruebas en los criterios de aceptación de la historia del usuario.

Este último era un objetivo elevado, inimaginable para muchos en la organización Test en ese momento. Sin embargo, en el transcurso de los siguientes dos años, los evaluadores de Scrum se volvieron muy eficientes en el desarrollo y ejecución de la automatización de pruebas en la CLI y las interfaces de servicios web dentro del sprint.

La automatización de pruebas pudo seguir el ritmo del desarrollo de historias de usuario, lo que dio como resultado una regresión automatizada activa y la creación de conjuntos de verificación que se ejecutan para cada proyecto. La Figura 18-1 muestra las actividades de codificación y prueba dentro del ciclo de sprint de dos semanas.

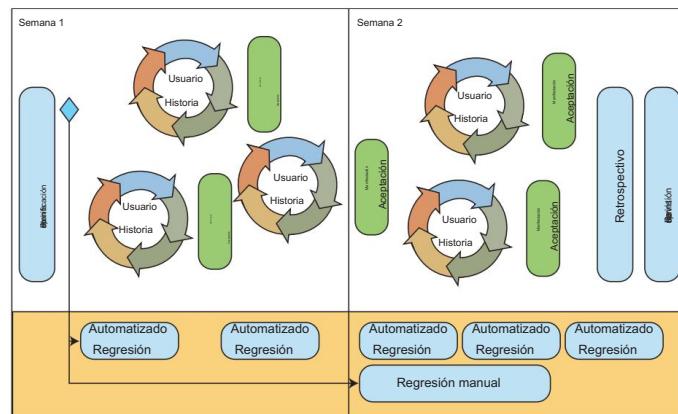


Figura 18-1 Ejemplo de ciclo de vida de sprint de dos semanas

Hemos notado que en organizaciones grandes, donde existen jerarquías de comando y control, es más difícil para los equipos autoorganizarse y adaptarse que en organizaciones más pequeñas con menos estructura. En la historia de Geoff, vemos que se necesitaba cierta dirección para lograr que los equipos aceptaran la nueva forma de trabajar. Sin embargo, con el tiempo, las nuevas prácticas se convirtieron en el status quo.

Coordinación de múltiples equipos



Realizar pruebas en una gran empresa significa pensar en muchas partes móviles. A menudo, cambiar un componente o una aplicación tiene repercusiones en todo el sistema. Se agrega más complejidad cuando algunos equipos practican la metodología ágil, mientras que otros trabajan en un proceso tradicional gradual y cerrado. Las transiciones ágiles, que normalmente requieren grandes cambios en la cultura empresarial, son especialmente dolorosas para las grandes corporaciones. Hay algunos desafíos únicos que los equipos de empresas más pequeñas no enfrentan.

Hay varias formas posibles de acercarse a varios equipos que trabajan en el mismo producto. En la Parte III, "Planificación para no olvidar el panorama general", hablamos sobre los niveles de precisión y la planificación de pruebas en cada nivel del producto. Usaremos algunos de esos conceptos para realizar pruebas en la empresa.

Usemos un ejemplo simple. La Figura 18-2 muestra un diagrama de contexto para un sistema escolar basado en el supuesto de que el sistema escolar es propietario de los autobuses. La nueva funcionalidad que se necesita es automatizar la programación de autobuses para que los padres sean notificados si hay un problema con la entrega de sus hijos. El escenario que usaremos como ejemplo es "Un autobús se avería, se envía un autobús nuevo para transportar a los niños y se notifica a los padres que los niños llegarán tarde".

Las características prioritarias que necesitamos son

- Capacidad para localizar autobuses si tienen problemas
- Capacidad de contactar a un conductor sustituto y asignar un autobús sustituto
- Capacidad de notificar a los padres mediante mensajes de texto, teléfono o correo electrónico
- Capacidad para notificar a la administración escolar sobre problemas con el mantenimiento del autobús.

tenencia y entrega tardía de niños

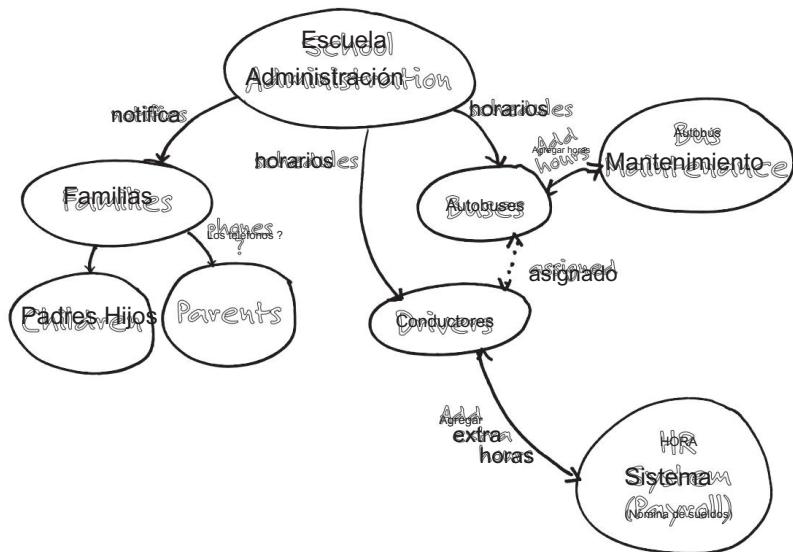


Figura 18-2 Diagrama de contexto para un sistema escolar y de autobuses

Aplicaremos el concepto de niveles de precisión a una organización con múltiples aplicaciones y varios equipos trabajando para ofrecer la mejora del producto. Cada equipo prueba sus características lo mejor que puede, pero sería ingenuo pensar que no hay dependencias entre los equipos o que pueden probar todo sobre una historia o característica dentro de su propio equipo. Según nuestra experiencia, los productos dentro de una empresa tienen sistemas muy complejos, lo que resulta en entornos de prueba muy complejos.

Un equipo de prueba del sistema y un entorno

Una de las opciones cuando hay varios equipos trabajando en el mismo producto es crear un equipo de prueba del sistema, quizás formado tanto por programadores como por evaluadores. Este equipo trabaja en estrecha colaboración con los otros equipos de entrega. Todos participan en una reunión de inicio del proyecto o de planificación de alto nivel al inicio del proyecto. Cada equipo tiene su propio entorno de integración continua (CI), pero también integra su código en una compilación maestra para todo el producto. El equipo de prueba del sistema prueba continuamente el producto "potencialmente entregable" entregado al final de cada iteración, o posiblemente con mayor frecuencia. Las pruebas pos-

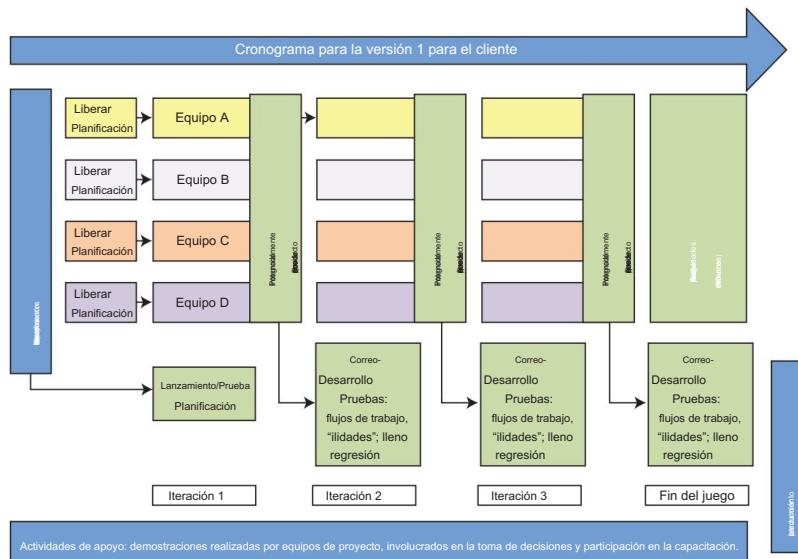


Figura 18-3 Una posible solución para varios equipos

y las actividades finales del juego no son iteraciones de refuerzo o corrección de errores. (Consulte Agile Testing, págs. 456–57, para obtener más información sobre el final del juego). Más bien, son una parte integral del proceso de desarrollo, probando todo el sistema de principio a fin en un entorno similar a la producción, que puede no estar disponible a cada equipo individual. En la Figura 18-3, mostramos cuatro equipos trabajando en funciones independientes, cada una de las cuales mejora el producto en su conjunto.

En nuestro ejemplo de programación de autobuses escolares, cada función se asignaría a un equipo diferente. Por ejemplo, el equipo A podría asumir la función "Ubicar autobuses que tienen problemas", mientras que el equipo B tiene "Contactar al conductor y asignar un autobús de reemplazo". Cada equipo descubriría qué historias necesitaban en la planificación del lanzamiento y estimaría aproximadamente cuánto tiempo llevaría completar su función. Sin embargo, como puedes imaginar, existen dependencias entre equipos, por lo que también deben considerar cómo probarlas.

Una vez que los equipos comprenden sus características, se reúnen, piensan en realizar pruebas en el nivel de versión superior, plantean dependencias y toman decisiones sobre cómo probarlas (consulte la Figura 18-4). A veces, los administradores de pruebas son una opción natural en organizaciones grandes, ya que ayudan a coordinar las dependencias entre equipos.

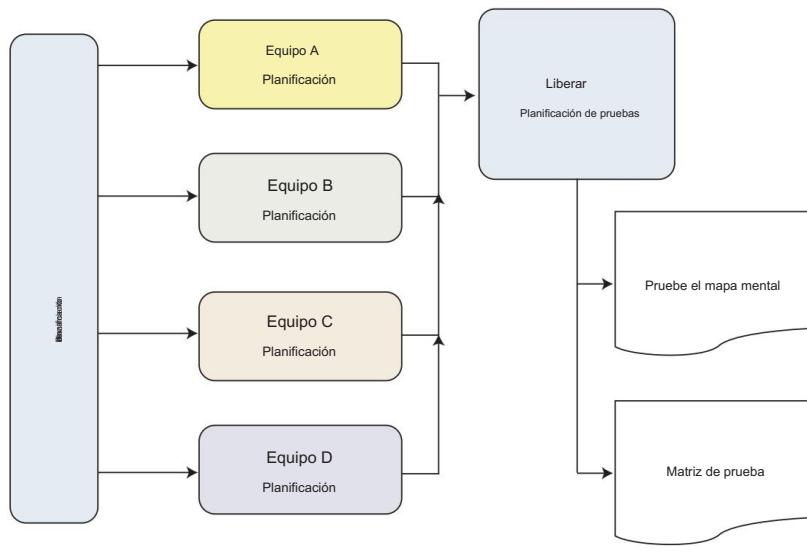
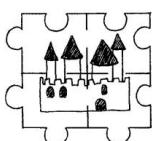


Figura 18-4 Planificación de pruebas: nivel de producto

Las actividades de prueba para historias deben realizarse dentro del equipo tanto como sea posible. A veces es difícil completar las pruebas, porque es posible que un equipo no tenga un entorno de prueba de sistema completo similar al de producción; puede resultar demasiado costoso. Las historias son pequeñas y granulares, por lo que un equipo debe poder llevarlas a Story Done dentro de la iteración.



Dado que las características pueden consistir en muchas historias, puede que no sea posible para un equipo individual probar completamente una característica en su propio entorno de prueba, por ejemplo, asegurándose de que las horas adicionales para el conductor sustituto solo puedan probarse en su totalidad. entorno del sistema que tiene acceso al sistema de recursos humanos (RRHH). Esto podría ser parte de Característica terminada y no de Historia terminada.

Sin embargo, dado que el equipo de entrega es en última instancia responsable de completar esa característica, es necesario que haya una manera de mostrar cuándo una característica necesita más pruebas y cuándo está completa, por lo que la comunicación entre los equipos es esencial. La validación de que el producto se puede enviar puede ser completada por un equipo de prueba del sistema que tenga acceso a un sistema completo equivalente a producción que les permita aprovechar las economías de escala para tipos de pruebas que son difíciles de justificar en características individuales.

equipos. Por ejemplo, las pruebas de rendimiento, carga, interoperabilidad y compatibilidad del navegador pueden requerir entornos de prueba especializados. Las pruebas de regresión del sistema completamente automatizadas también pueden ser parte de la responsabilidad de este equipo de prueba del sistema. Durante el final del juego, sugerimos que todos los equipos participen en las actividades necesarias para llevar el lanzamiento a Release Done.

Hay un par de formas diferentes de gestionar un equipo de prueba del sistema. Una forma es tener programadores como parte del equipo para ayudar a identificar y solucionar de inmediato los problemas que se encuentren. Diferentes organizaciones pueden optar por definir diferentes procesos para abordar defectos y problemas.

A veces, cualquier defecto encontrado volverá al equipo original.

Sin embargo, esto a veces puede ser un problema; Si no hay un dueño claro, los equipos pueden transferir el defecto a otro equipo, lo que puede degenerar en un juego de culpas. Lisa trabajó con una empresa de 25 equipos donde los defectos detectados por el CI se plantearon en un Scrum-of-Scrums de “sólo ayuda”. Un ScrumMaster hablaría y diría que su equipo asumiría la responsabilidad. Como ocurre con cualquier proceso ágil, encontrar y corregir defectos debe ser un ciclo de retroalimentación breve.

Volvamos a la historia de Geoff Meyer sobre cómo Dell manejó la ampliación de sus pruebas ágiles.

Creciendo: Dell, Parte 4

A mediados de 2011, comenzamos a aplicar prácticas ágiles a nuestros primeros proyectos a gran escala. Un proyecto tenía nueve equipos Scrum y el otro tenía 15.

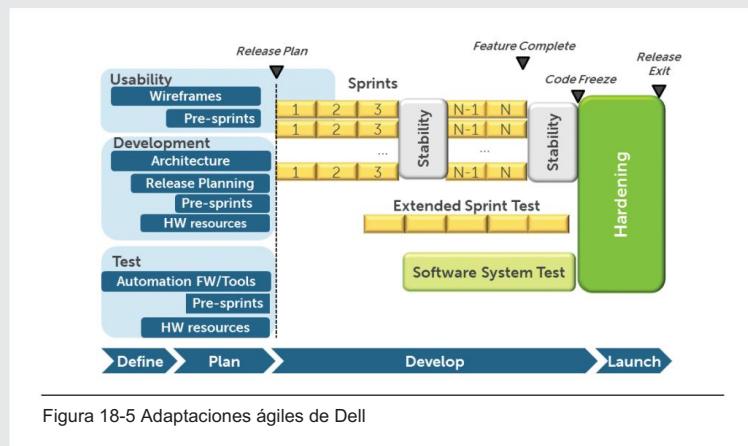
Formamos equipos basados en funciones, con evaluadores integrados que desarrollaron automatización de pruebas en sprint para historias de usuarios (tal como lo hicimos con nuestros proyectos más pequeños).

Sin embargo, determinamos que este nivel de pruebas iba a ser insuficiente para un esfuerzo de tan gran escala, ya que ningún equipo Scrum tenía visibilidad del uso de la aplicación desde la perspectiva del cliente.

Por lo tanto, introdujimos el concepto de prueba de sistema de software (SST), que adoptó un enfoque de prueba de caja negra más tradicional, centrándose en la interfaz de usuario de las funciones completadas. Este enfoque incluyó la automatización de los casos de prueba de la interfaz de usuario. Nuestro objetivo era comenzar estas pruebas lo antes posible, en paralelo a los sprints, pero no tan pronto como para obstaculizar el desarrollo de historias de usuarios en progreso (en realidad, no antes del segundo o tercer sprint).

Otro aspecto de estos proyectos a gran escala que el equipo de pruebas debía abordar era probar nuevas funciones, en forma de historias de usuarios, en varias generaciones de servidores existentes. Aquí es donde la automatización del sprint a nivel de servicio realmente dio sus frutos. Capacitamos a los equipos de Scrum que tenían historias de usuarios que abarcaban una gran matriz de compatibilidad para limitar sus pruebas de aceptación a una o más configuraciones de referencia.

Luego establecimos un equipo de “prueba de sprint extendida” que tomó como entrada historias de usuarios aceptadas (que incluían scripts de prueba automatizados). Luego, este equipo configuró los entornos de prueba físicos y ejecutó los scripts de prueba automatizados contra estas configuraciones de compatibilidad extendida para completar las historias de los usuarios (consulte la Figura 18-5).



Durante las etapas 2012-2013 de nuestro viaje ágil de automatización de pruebas, hicimos nuestra incursión inicial en el cuarto cuadrante de pruebas de Lisa y Janet: pruebas no funcionales. El énfasis estaba en aplicar nuestras habilidades de automatización a aquellas áreas donde los esfuerzos manuales generalmente resultaban poco prácticos. La longevidad, la escalabilidad y la base de rendimiento eran áreas que habían recibido una cobertura limitada en el pasado, pero pudimos incorporarlas al proceso de desarrollo ágil y ejecutarlas en paralelo a nuestros sprints. También aplicamos la automatización a tareas de prueba que consumen mucho tiempo y no están relacionadas con casos de prueba.

En la historia de Geoff se puede ver que a las grandes organizaciones de desarrollo y pruebas de Dell les tomó tiempo descubrir nuevas formas de trabajar mediante la adaptación y la experimentación. El equipo de prueba de sprint extendido aprovechó las economías de escala y trabajó en paralelo con los equipos de Scrum para brindar retroalimentación lo más rápido posible. A menudo equi-

Necesitan iteraciones de estabilización mientras se adaptan, pero con el tiempo, la necesidad de ellas debería disminuir.

Herramientas consistentes

Existen varias otras prácticas que pueden mejorar el éxito empresarial.

Por ejemplo, podemos aprovechar las herramientas actuales disponibles para las organizaciones. En el Capítulo 17, "Selección de soluciones de automatización de pruebas", incluimos una sección sobre selección de herramientas para la automatización. Sin embargo, es común contar con herramientas existentes y muchas organizaciones dudan en cambiarlas. Es necesario que haya una razón convincente para cambiar, por lo que los equipos deben poder articular esos beneficios.

Coordinación a través de CI

La integración continua con pruebas de regresión automatizadas en todos los equipos, con todos los equipos trabajando en la misma base de código, garantiza que la mayoría de los problemas de integración se detecten tan pronto como se registre el código. Lograr esto puede requerir mucho tiempo y esfuerzo, pero la recompensa es Más que vale la pena. Lisa trabajó en una empresa donde tomó algunos años lograr que un CI para todo el sistema funcionara solo un par de veces por semana. Pero una vez que lograron esto, fue cuestión de meses ponerlo en funcionamiento dos veces al día, y los equipos responsables de las fallas de regresión solucionaron los problemas el mismo día.

Uno de los mayores problemas que enfrentan las grandes organizaciones es dar prioridad a la corrección de errores. ¿Quién soluciona los defectos encontrados por las pruebas de regresión automatizadas? ¿Cómo determinamos cuál es la prioridad? Muchas organizaciones cuentan con un equipo de soporte de producción que es responsable de solucionar los defectos de producción informados por los clientes, pero a nadie le importan los defectos internos encontrados. Las fallas de las pruebas de regresión automatizadas deben ser reparadas inmediatamente por el equipo que verificó el código que causó la prueba fallida; de lo contrario, se pierde el beneficio de la retroalimentación rápida. Los equipos deben decidir cómo manejar errores encontrados durante el desarrollo o en producción. Como analizamos en el Capítulo 14, "Deuda técnica en las pruebas", los errores que persisten en un sistema de seguimiento de defectos aumentan la carga de la deuda técnica del equipo.

Utilice su CI para comunicar los resultados de las pruebas de regresión funcional automatizadas en toda la organización de desarrollo. Hay varias posibilidades para gestionar los fallos de las pruebas. Una es tener un equipo de prueba del sistema.

investigue las fallas, decida qué equipo de entrega es responsable y asigneles la prueba fallida. Necesitan una buena descripción general de las funciones que se están desarrollando para poder tener una mejor idea de qué puede haber superado las pruebas. Otra opción es rotar qué equipo analiza los fallos de regresión. La desventaja de este enfoque es que la persona o el equipo responsable de verificar los resultados puede no saber en qué están trabajando los otros equipos y puede resultarle difícil determinar el origen del problema. La ventaja es que comparte la responsabilidad de investigar fallas y puede ofrecer un poco de capacitación cruzada. No existe un único enfoque correcto, así que experimente con alternativas.

Enfoques de control de versiones

Nos gusta que todos los equipos trabajen con la misma base de código. Esto no sólo mitiga los riesgos de integración sino que también reduce los riesgos de prueba. Sabemos que el conjunto de pruebas de automatización se ejecuta (como mínimo, todas las noches) en el código, por lo que detectamos los problemas con antelación. Realizamos nuestras pruebas exploratorias en la misma base de código para saber cómo se comporta en cada implementación. Cada equipo de entrega tiene su propio entorno de prueba, pero se pueden utilizar entornos compartidos según sea necesario. Si hay un defecto de producción y tenemos que hacer una reparación e implementación de emergencia, solo tenemos dos ramas para probar: producción y la rama principal de desarrollo. La Figura 18-6 muestra un gráfico de los entornos que el equipo de Lisa siguió mientras actualizaban sus versiones de Ruby y Rails.

Otro enfoque, que hemos visto en algunas organizaciones grandes, es que los equipos individuales trabajen en ramas separadas y se fusionen con el tronco principal al final de la iteración. Aunque puede haber problemas adicionales para solucionar problemas de integración para la fusión, este enfoque puede funcionar bien, siempre que el sistema de control de versiones maneje las fusiones correctamente. Una desventaja de este proceso es tener que decidir qué rama obtiene la solución de producción en situaciones de emergencia. ¿Qué equipo se asegurará de que se pruebe antes y después de la fusión? ¿Quién realiza un seguimiento para asegurarse de que las pruebas se realicen tanto en la rama de producción como en la rama de desarrollo? Existe un mayor riesgo de que algo se escape al utilizar esta estrategia, aunque se puede mitigar si el equipo de prueba del sistema tiene una buena comprensión de los riesgos y las dependencias entre los equipos. El riesgo también se reduce si el equipo tiene una cobertura de automatización de pruebas confiable y puede realizar un seguimiento con algunas pruebas exploratorias sobre el riesgo.

áreas.

Ambiente	Dueño	Implementado por	Aplicación 1	Aplicación 2	Rieles Rubí
DESEARROLLO	desarrolladores	Automatizado	4.1.0_124	2.1.0	3.2 2.1.1
PRUEBA	probadores	Pares de probadores	4.1.0_122	2.1.0	3.2 2.1.1
AUTO	DevOps	Automatizado	4.1.0_124	2.1.0	3.2 2.1.1
COMPARTIDO	DevOps	Implementar par	4.1.0_120	2.1.0	3.2 2.1.1
CARGA	DevOps	Implementar par	4.1.0_120	2.1.0	3.2 2.1.1
PUESTA EN ESCENA	DevOps	DevOps	4.1.0_110	2.1.0	2.3 1.87-2012
PRODUCCIÓN	operaciones	operaciones	4.1.0	2.1.0	2.3 1.87-2012

Figura 18-6 Haga visibles sus entornos de prueba.

Cobertura de prueba

Uno de los riesgos de que varios equipos trabajen en el mismo producto son las posibles superposiciones y lagunas en las pruebas. En un mundo ideal, todas las funciones serían autónomas y no dependerían de otras funciones. Sin embargo, no vivimos en un mundo ideal, especialmente cuando se trata de grandes corporaciones. Probablemente hayan crecido a lo largo de los años mediante fusiones y adquisiciones. La coordinación de pruebas entre equipos que pueden haberse originado en diferentes empresas, trabajando en productos de software separados, puede convertirse en una pesadilla.

Ahora existen herramientas que pueden ayudarle a comprender qué cobertura de prueba tiene su sistema. En una conferencia reciente, Janet vio una demostración de un producto diseñado para ayudar a visualizar dónde podrían estar las brechas en las pruebas. Comprenda sus problemas y sus riesgos, y luego vea si puede desarrollar o adoptar una herramienta para ayudar a resolverlos. Ninguna herramienta solucionará todos los problemas de cobertura de pruebas.

Animamos a los equipos a experimentar, pero siempre teniendo en cuenta que el objetivo no es comprar más herramientas o automatizar cada vez más pruebas, sino reducir el riesgo mediante una retroalimentación rápida. Esto significa que desea abordar los problemas de integración lo antes posible y evitar encontrarlos durante el final del juego, inmediatamente antes del lanzamiento.

Gestión de dependencias

Los equipos intentan dividir las funciones y las historias para que no haya dependencias, pero eso es casi imposible, especialmente cuando se trata de soluciones empresariales. Hay proveedores externos e internos, otros equipos de entrega o incluso clientes de los que los equipos pueden depender antes de poder entregar sus historias y funciones.

Trabajar con terceros

Uno de los mayores problemas que tienen las grandes organizaciones es trabajar con organizaciones de desarrollo fuera del equipo inmediato. Hay terceros tanto internos como externos, y le animamos a pensar en ellos de manera similar. Los proveedores externos pueden incluir organizaciones como proveedores de tarjetas de crédito sobre las cuales usted no tiene control, o compañías de productos más pequeñas que estén dispuestas a trabajar en estrecha colaboración con su equipo. Los terceros internos pueden incluir su equipo de base de datos o el departamento de contabilidad. Puede resultar difícil trabajar con estos grupos internos porque esperamos que respondan cuando queremos que lo hagan. Sin embargo, tienen sus propios conjuntos de prioridades y es posible que no respondan como se esperaba.

Las prácticas ágiles para el desarrollo en asociación con proveedores externos se están consolidando. Técnicas como pasar a un contrato centrado en productos y resultados son cada vez más frecuentes (véase McDonald, 2013). Quizás no sea sorprendente que, si bien han surgido prácticas para integrar a terceros en el trabajo de programación de un proyecto, haya poco acuerdo sobre dichas prácticas para probar proyectos ágiles.

La historia de Janet

Últimamente parece que muchos de mis clientes entran en la categoría de grandes empresas. A veces, una de las primeras señales de alerta que veo es una columna en su guión gráfico para historias o tareas bloqueadas. Me dice que están planeando tener historias bloqueadas. Me gustaría ver esto como una excepción y no como una regla.

Un consejo que les doy a estos equipos es que tengan cuidado con las historias que dependen mucho de otro equipo o de un tercero externo. Sugiero que el equipo elimine la historia de la iteración hasta que se elimine la dependencia. De esa manera, el equipo puede centrarse en otras historias en lugar de esperar a que otro equipo entregue algún código o que un tercero entregue el producto prometido.

Una forma de eliminar dependencias es tener una columna o estado "Listo" en sus historias. Por ejemplo, si su historia tiene una dependencia en la que necesita que un administrador de base de datos le ayude con algunas tablas de base de datos nuevas, o necesita que el grupo de experiencia del usuario cree una estructura alámbrica, mantenga la historia en el trabajo pendiente hasta que tenga el artefacto necesario o compromiso de su parte para trabajar con usted en su historia. Comuníquese con esos grupos con anticipación para que estén preparados para brindar sus entregables antes de que se priorice la historia. Esto es parte de la planificación de la preparación de su historia. Trabaje para eliminar estas dependencias durante la planificación. Algunos equipos celebran reuniones específicamente para discutir las dependencias antes de planificar la iteración para abordar situaciones como las que describimos.

La historia de Lisa



Mi último equipo comenzó a hacer algo similar cuando teníamos muchas historias en las que la gente de negocios no tenía las reglas comerciales listas. Pensaron que conocíamos el dominio tan bien que podríamos crear las reglas nosotros mismos. ¡Mmm no! Retrocedimos. Si no teníamos todos los requisitos, maquetas y reglas de negocio necesarios para una historia en el segundo día del sprint, esa historia era eliminada y usábamos el tiempo para trabajar en nuestras propias iniciativas para gestionar los aspectos técnicos. deuda. Cuando las partes interesadas del negocio se dieron cuenta de que no tener todo para nosotros al inicio de la iteración significaba un retraso de dos semanas para la historia, fueron más disciplinados a la hora de prepararse para el futuro.

Cuando trabaja con terceros externos, la automatización puede realmente ser una ventaja para usted. Obtenga la descripción de la API del proveedor y cree herramientas de prueba para replicar las entradas del proveedor en su sistema, y luego podrá probar el comportamiento de su sistema con la automatización. Esto le permite probar su sistema sin tener instalado un sistema de terceros. Sí, habrá cambios en el camino. Sin embargo, estará más adelante y comprenderá más fácilmente cuáles son los problemas.

ily ya que ha definido el comportamiento. Una de las mayores quejas que escuchamos de los equipos es: "No puedo obtener esa información". Intente solicitar piezas pequeñas a la vez si realmente necesita comenzar a desarrollar sin tener la API completa. Es posible que se sorprenda de cómo abrir la puerta e iniciar la conversación crea una mejor relación de trabajo con el tercero, ya sea otro equipo dentro de su corporación o un proveedor externo.

Aceptar nuevas aplicaciones de terceros puede resultar aterrador si no confías en sus cambios. Trabaje para mejorar esa confianza, pero al mismo tiempo considere cómo puede hacer que su equipo se sienta más seguro al aceptar los últimos cambios. Tal vez cree un entorno de prueba donde pueda comparar lo que hizo el sistema antes del cambio con cómo maneja la funcionalidad crítica después del cambio. Piense en compartir sus pruebas con el proveedor para intentar llegar a una comprensión común de lo que se entrega. El proveedor puede incluso aprender a realizar pruebas antes de realizar la entrega.

Piense en todas sus partes interesadas, incluidos sus clientes, las personas que realmente utilizan su sistema. Si usted es un tercero externo que ofrece un producto, piense en cómo puede hacer que sus clientes confíen en lo que ofrece.

Involucrar a los clientes en grandes organizaciones

Las pruebas en sistemas empresariales requieren la colaboración no sólo dentro de cada equipo de desarrollo sino también con muchos grupos externos. El más importante de estos terceros es, por supuesto, nuestro cliente. En muchas corporaciones, los equipos de desarrollo están muy alejados de sus usuarios finales. Involucrar a los clientes en el proceso de prueba y desarrollo puede requerir creatividad adicional.

Pruebas de aceptación del usuario en la empresa

Susan Bligh , analista de negocios en Alberta, Canadá, le dice que los incluye Sobre lo que sucedió cuando las historias de sus clientes y pasó desapercibida.

Uno de los mayores problemas que encontramos cuando era analista de negocios y trabajaba en una implementación de software importante para toda la empresa fue cómo integrar el diseño en todas las funciones (silos) del programa. Implementamos seis funciones diferentes que se integraron entre sí, incluidas Finanzas y Contabilidad, Cadena de Suministro, Gestión de Activos, Planificación y Presupuesto, Proyectos de Capital y Recursos Humanos.

Al comienzo del programa, éramos muy conscientes de que necesitábamos tener un diseño integrado que abarcara todas las funciones. Naturalmente, durante toda la implementación, los equipos funcionales trabajaron para diseñar, probar y documentar sus propias funciones con solo un mínimo

diseño y pruebas multifuncionales. Realizamos pruebas integradas de extremo a extremo durante dos de los cuatro ciclos principales de prueba de productos, y los miembros del equipo del proyecto ejecutaron las pruebas. Estos miembros del equipo tenían mucho conocimiento sobre el software, pero no siempre formaban parte de las consideraciones más amplias del diseño empresarial.

Cerca del final del programa, se tomó la decisión de renunciar a las pruebas de aceptación del usuario (UAT) y, en su lugar, realizar una simulación empresarial en algunas áreas piloto clave. La simulación empresarial fue dirigida nuevamente por los equipos del proyecto, pero esta vez participó el usuario final. Cada equipo funcional creó por separado la documentación y la capacitación para el usuario final.

Una mirada retrospectiva al proyecto reveló algunas fallas de implementación que llevaron a problemas de diseño e inefficiencias que podrían haberse evitado.

Algunas de las lecciones aprendidas del proyecto incluyen:

- Dedicar más tiempo a diseñar y probar los puentes entre los silos que a diseñar los silos. Los equipos de diseño deben estar compuestos por miembros del equipo comercial, técnico y de pruebas.

Nuestra historia: Sólo un par de semanas antes de la puesta en marcha descubrimos que un equipo funcional estaba creando órdenes de compra de cierto tipo, solo para darnos cuenta de que el equipo de Cuentas por Pagar no admitía ese tipo de orden de compra. Si hubiéramos implementado ese defecto de diseño, cualquier orden de compra de ese tipo emitida a un proveedor no podría haberse pagado. Este problema podría haberse evitado si hubiéramos dedicado más tiempo a colaborar en las áreas de integración entre las funciones.

- Capacitar a los usuarios finales utilizando la documentación y los programas de capacitación para usuarios finales, y permitirles realizar la UAT con el conocimiento proporcionado. El equipo del proyecto debe participar únicamente para ayudar a documentar los resultados de las pruebas (incluidos los defectos del sistema, la capacitación y la documentación). Esto permite a los usuarios probar no sólo el diseño del sistema, sino también la documentación del usuario final y los programas de capacitación.

Nuestra historia: Proporcionamos cierta documentación para la simulación empresarial, pero la capacitación la realizaron los miembros del equipo del proyecto que conocían y entendían el diseño en detalle. No toda la documentación para el usuario final estaba disponible. La simulación empresarial fue realizado por miembros del equipo del proyecto con los usuarios finales. Aunque no se dieron cuenta en ese momento, el equipo del proyecto estaba solucionando los defectos de diseño para permitir que las pruebas pasaran. Si los usuarios finales hubieran realizado todas las pruebas ellos mismos sin la ayuda de los miembros del equipo del proyecto que habían estado trabajando con el sistema durante meses, los fallos de diseño se habrían encontrado mucho antes. Además, dado que los miembros del equipo del proyecto estuvieron disponibles para responder preguntas durante la simulación empresarial, no se encontraron muchos defectos en la documentación y la capacitación.

Como puede verse en la historia de Susan, incluso el más alto nivel de diligencia por parte de los equipos de ejecución podría no ser suficiente si no se involucra a todos los representantes de las partes interesadas en las actividades de prueba.

Ventajas de llegar más allá el equipo de entrega

Comenzamos el capítulo hablando de impactos y entendiendo los objetivos del cliente. Cuando adaptamos nuestros procesos para satisfacer sus necesidades, no solo cumplimos, superamos o incluso deleitamos a nuestros clientes; Por lo general, terminamos con una dinámica de equipo positiva. Únase a nosotros para leer la parte final de la historia de Geoff Meyer sobre el viaje de Dell.

Resultados: Dell, parte 5

Enterprise Solutions Group es, en última instancia, responsable de ofrecer valor a través de nuestros productos y, a menudo, las entregas de nuestros productos están vinculadas a los hitos de los socios.

La adopción de la metodología ágil ha proporcionado a Dell ESG un mayor nivel de previsibilidad de cronograma. Dentro de Ingeniería, se ha mejorado enormemente el entorno de trabajo diario para programadores y testers.

Los equipos de Scrum son muy comprometidos, colaborativos y amigables y reciben satisfacción regular al completar funciones de trabajo con un ritmo regular. Los miembros del equipo de desarrollo agradecen y respaldan las habilidades aportadas por los miembros de prueba del equipo Scrum, y Test ahora es un socio pleno en todo el proceso de desarrollo de software.

La inclusión de ingenieros de pruebas en el proceso de requisitos iniciales, combinada con su curiosidad natural sobre cómo pueden funcionar o fallar las características, ha mejorado la usabilidad y funcionalidad de nuestros productos.

Dentro de la organización de pruebas, la metodología ágil nos ha llevado a un perfil de conjunto de habilidades avanzadas para nuestros evaluadores. La necesidad de la automatización de pruebas en apoyo de la metodología ágil nos ha llevado a adquirir y/o desarrollar habilidades de ingeniería de software. Las habilidades de automatización, a su vez, han llevado a reducciones del tiempo de ciclo en las ejecuciones de pruebas y a una mayor cobertura de la matriz de pruebas que no habría sido concebible antes de las iniciativas ágiles y de automatización.

Considerándolo todo, ha sido un viaje muy productivo tanto para las empresas como para los individuos.

La organización de Geoff agregó diferentes roles y grupos de especialistas, como arquitectos de pruebas y un equipo de pruebas de sprint extendido, para permitir que la organización grande y complicada aproveche los principios y prácticas ágiles. Se centraron en resolver un problema a la vez y experimentaron con una variedad de enfoques para integrar la codificación y las pruebas. Parece como si los nuevos miembros del equipo multifuncional disfrutaran colaborando y cada miembro del equipo contribuyera con su experiencia especializada. Disfrutar de su trabajo mientras deleita a sus clientes es un gran resultado de practicar la mejora continua.

No importa el tamaño de la organización o cuántos productos admite, se aplican principios ágiles. Los equipos de las grandes empresas están sujetos a un nivel más extremo de dificultades que encontramos en la mayoría de los equipos que practican pruebas ágiles. Por ejemplo, es mucho más difícil mantener un ojo en el panorama general e identificar efectos dominó en un sistema grande compuesto por múltiples productos de software que con una sola aplicación. Puede resultar más difícil comunicarse eficazmente con las partes interesadas, ya que hay más partes involucradas.

Hay diferentes formas de abordar problemas de pruebas a gran escala, pero los principios y valores ágiles lo guían para probar experimentos útiles, obtener comentarios rápidos y tomar buenas decisiones. La historia de cómo Spotify escaló el desarrollo ágil es una buena fuente de ideas sobre cómo utilizar principios ágiles sin volverse demasiado riguroso (Kniberg e Ivarsson, 2012).

Resumen

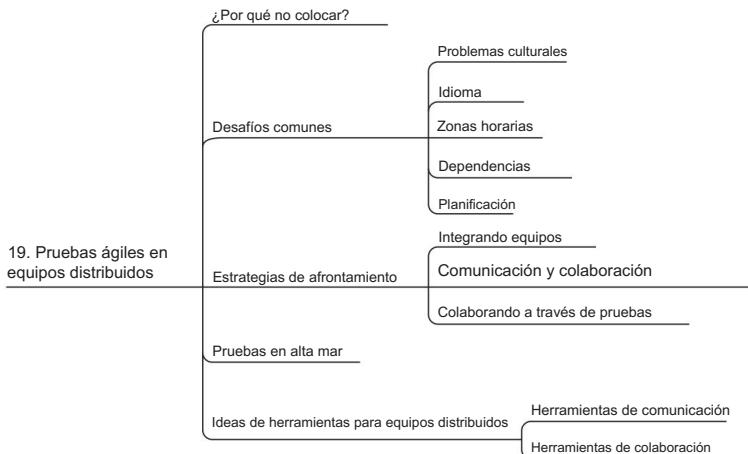
Los entornos empresariales magnifican los desafíos de las pruebas para los equipos ágiles al igual que lo hacen para los equipos que utilizan cualquier proceso de desarrollo. Necesitas más disciplina y más experimentos que intentar, pero siempre empezando con el enfoque más simple posible.

- Aprenda a trabajar dentro de los controles organizacionales, pero trate de encontrar nuevas soluciones ofreciendo sugerencias. Entender la historia y cultura organizacional para poder ofrecer soluciones alternativas.
- Las técnicas de especificación como Planguage pueden ser útiles para definir objetivos y utilizar un lenguaje coherente al describir los atributos de calidad.
- Cuando las grandes empresas implementan la metodología ágil, ayuda tener una Campeón a nivel ejecutivo para ayudar a apoyar a los equipos en la transición.

- La retroalimentación rápida es fundamental en las soluciones empresariales cuando muchos equipos y aplicaciones necesitan trabajar juntos. Comprenda el panorama general y las dependencias entre equipos para determinar la mejor manera de dar y recibir comentarios.
- Gestionar las dependencias es un desafío en las grandes empresas. Hacer visibles las dependencias para fomentar el pensamiento en ellas durante la planificación.
- Considere agregar roles adicionales, como arquitectos de pruebas o administradores de pruebas, para ayudar a coordinar actividades entre muchos equipos.
- Considere la posibilidad de crear un equipo de prueba del sistema para aprovechar las economías de escala en los entornos de prueba.
- Utilice herramientas consistentes; La CI y el control de versiones ayudan a coordinar a varios equipos que trabajan en el mismo producto o en productos relacionados.
- Ir más allá del único equipo de entrega y de las pruebas que pueden realizar. Incluya a otros miembros de la organización, clientes y usuarios finales para completar la solución.

Capítulo 19

Pruebas ágiles en distribución equipos



Casi todos los libros que hemos leído sobre cómo escalar equipos grandes o distribuidos recomiendan que los equipos de funciones existan en una ubicación, si es posible. Esto se considera un requisito previo para la colaboración tan necesaria en Agile. Sin embargo, reconocemos que es difícil, si no imposible, para algunas organizaciones cambiar su estructura distribuida existente. Incluso puede haber ventajas en su naturaleza distribuida a las que preferirían no renunciar.

Hacemos una distinción entre equipos dispersos, equipos distribuidos y pruebas en el extranjero, pero usaremos el término equipos distribuidos para referirnos a los tres, a menos que estemos hablando específicamente de dispersos o en el extranjero.

- Los equipos dispersos pueden tener varios miembros trabajando de forma remota desde casa, aunque también puede haber algunos miembros del equipo ubicados juntos. Trabajan juntos como un solo equipo (ver Figura 19-1).

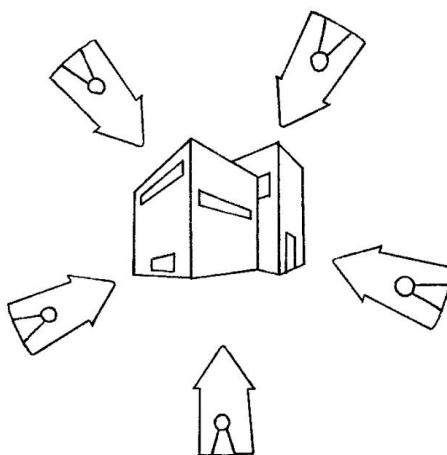


Figura 19-1 Representación de equipos dispersos

- Los equipos distribuidos son aquellos que tienen subequipos o equipos completos ubicados en múltiples ubicaciones, posiblemente en diferentes zonas horarias o diferentes países, todos trabajando en el mismo producto (consulte la Figura 19-2).



Figura 19-2 Representación de equipos distribuidos

■ Las pruebas en el extranjero se producen cuando una organización decide subcontratar las pruebas de sus productos a un proveedor en otra ubicación, generalmente en otro país donde los costos laborales son más bajos. El nearshoring es una variación de esto, donde las pruebas se subcontratan a países en una ubicación más cercana. Por ejemplo, una empresa estadounidense podría realizar un acercamiento con una empresa en Sudamérica o México. El Near-shoring generalmente significa que no hay diferencia horaria, o es pequeña.

Hay otras variaciones de estas categorías. Una empresa puede seguir realizando pruebas en su ubicación principal pero deslocalizando algunos trabajos de desarrollo. Las recomendaciones de este capítulo siguen siendo válidas.

Nos consideramos un equipo disperso. Ambos trabajamos de forma remota pero trabajamos como un solo equipo sin una base de operaciones. Al escribir este libro, utilizamos muchas de las herramientas que le sugerimos en este capítulo. Por ejemplo, utilizamos Google Docs y Dropbox para compartir información a la que ambos necesitábamos acceso en tiempo real, como nuestro plan de lanzamiento. Usamos MindMeister para nuestras sesiones de mapas mentales, a veces colaborando en tiempo real en una idea y otras veces cada uno de nosotros haciendo actualizaciones para compartir con el otro más tarde. Cuando necesitábamos conectarnos y tener conversaciones, utilizamos Skype o Google Hangouts. Probamos nuestras ideas e intentamos obtener comentarios rápidos compartiendo capítulos con revisores en Google Docs, y utilizamos un grupo de Google para que los revisores compartieran ideas o colaboraran para aclarar algo que podría resultar confuso. Puede que no haya sido la situación ideal, pero la hicimos funcionar (Gregory, 2014).

Hay muchos libros y recursos que le ayudarán a que su equipo distribuido tenga éxito. Las dificultades no se limitan a las pruebas. Sin embargo, es mucho más difícil colaborar en actividades de prueba y compartir la responsabilidad colectiva de la calidad en equipos distribuidos. En este capítulo veremos cómo las pruebas pueden tener éxito en equipos distribuidos.

¿Por qué no colocar?

Hay varias razones por las que las organizaciones tienen equipos distribuidos. A menudo se debe a fusiones o adquisiciones a medida que la empresa ha ido creciendo. Una de las principales razones por las que los equipos tienen miembros remotos es porque no pueden encontrar candidatos cualificados en su zona geográfica. Por ejemplo, cada vez es más difícil encontrar buenos evaluadores.

en algunas zonas del mundo. Las buenas personas ayudan a que los proyectos de software tengan éxito. Algunos equipos han cambiado sus normas para aceptar personas que trabajan desde otros lugares. En estos casos, las desventajas de la no colocación se ven compensadas por el valor que los empleados remotos aportan al equipo.

Los equipos distribuidos tienen ventajas, pero la siguiente sección cubrirá los desafíos relacionados con las pruebas que enfrentan los equipos distribuidos y ofreceremos algunas sugerencias para ayudar a superar esos desafíos.

Desafíos comunes

No importa si tiene equipos distribuidos globalmente o miembros individuales del equipo remoto; aprender a trabajar juntos desde múltiples ubicaciones físicas es difícil. Para los nuevos equipos ágiles donde los evaluadores están aprendiendo a colaborar con programadores, clientes y otros miembros del equipo, la distancia física dificulta la transición.

Problemas culturales

Los equipos distribuidos por todo el mundo necesitan abordar cuestiones culturales.

Por ejemplo, la gente de un país (llamémoslo Reservandia) puede ser tranquila y reservada, mientras que en otro país (lo llamaremos Assertivana) la gente dice lo que piensa.

Tómate un momento para pensar en estas dos perspectivas. Esperamos que estos dos grupos trabajen como un solo equipo. Los programadores están en Assertivana y los testers en Reservandia. Los programadores empiezan a hablar sobre cómo creen que debería implementarse el software. Los evaluadores se sientan y escuchan; pueden sentir que ni siquiera tienen la oportunidad de decir una palabra. Puede que este no sea siempre el caso, pero hemos escuchado la misma historia una y otra vez, de ambos lados. Los programadores de Assertivana se quejan de que los evaluadores nunca hablan, mientras que los evaluadores dicen que nunca tienen la oportunidad de ofrecer sugerencias.



Se trata de diferencias problemáticas, pero con algo de entrenamiento por parte de ambas partes se pueden superar. Primero, ambas partes deben sentirse seguras para compartir sus inquietudes o sugerencias, especialmente Reservandia. Los programadores de Assertivana pueden aprender a escuchar primero y luego a dar su opinión. Rese-

Los evaluadores pueden estar preparados y tal vez presentar sus ideas por correo electrónico para empezar, y luego estar preparados para hablar sobre el tema.

Las empresas que optan por deslocalizar determinadas actividades de desarrollo, como las pruebas, deben ser conscientes del impacto que esa decisión puede tener en la cultura del equipo, tanto en la ubicación principal como para el equipo deslocalizado. Como nos señaló Jan Petter Hagberg (Hagberg, 2013), los miembros del equipo “en casa” probablemente no eligieron trabajar con un equipo offshore. El proceso de deslocalización puede haber sido traumático para los miembros del equipo que aún se encontraban en la ubicación original. Asegúrese de que reciban ayuda durante esta transición, que puede llevar mucho tiempo.

Las diferencias culturales pueden hacer que un comentario que una persona hizo como broma resulte insultante para otra. La jerga también puede obstaculizar una comunicación fluida. Tómese el tiempo para conocer a los miembros del equipo en otros lugares y su cultura. Experimente con formas de mejorar la comunicación.

Una advertencia: con demasiada frecuencia vemos que los equipos utilizan el término cuestiones culturales como una excusa para no buscar una solución.

Idioma

Los equipos que abarcan países enfrentan problemas de idioma. El inglés es el idioma más utilizado, pero no es el primer idioma de todas las personas y, cuando se habla, suele hacerlo con acento. Incluso con el inglés como primera lengua, el acento del hablante y la velocidad al hablar afectan la forma en que los demás escuchan.

La jerga y los coloquialismos dentro del mismo idioma también pueden crear malentendidos. Crear un lenguaje común al hablar de historias y características es aún más importante para los equipos distribuidos.

La historia de Janet

Paso mucho tiempo en otros países entrenando y asesorando. En una sesión de capacitación en Dinamarca, uno de los participantes se acercó a mí y me felicitó por hablar despacio y usar palabras comunes, pero dijo que Probablemente se perdió entre el 5% y el 10% de lo que dije. Me sorprendí porque él Hablaba un inglés excelente. Lo discutimos más a fondo y me hizo darme cuenta de la Impacto de la comunicación dentro de los equipos.

Consideraremos tres equipos: uno en Estados Unidos, uno en China y otro en Francia. Cuando la gente en Francia habla con acento francés, y tal vez

luchan por encontrar las palabras adecuadas, la gente en Estados Unidos probablemente perder el típico 5% a 10% en traducción. ¿Qué crees que sucede cuando ¿El equipo de China está tratando de entender? ¿Hasta qué punto llegamos realmente a un entendimiento común cuando todos estamos perdiendo un poco, o quizás mucho?

La conciencia es la mitad de la batalla, pero podemos ayudarnos a nosotros mismos y a nuestros equipos a mejorarlala. Pruebe cosas como repetir una idea o parafrasear para asegurarse de que la entendió, o pedirle a un portavoz que repita la oración o el concepto. Las personas necesitan sentir que es seguro decir que no entendieron y pedirle a alguien que repita un comentario. Dé tiempo a la gente para traducir mentalmente lo que se ha dicho antes de continuar. Quizás resumir las decisiones por escrito en una wiki del equipo para que los miembros del equipo puedan leerlas y tener tiempo para interiorizarlas. Piensa antes de hacer una pregunta. "Hagamos una encuesta para conocer la opinión de todos" en lugar de "¿Alguien tiene alguna objeción?" ayudará a determinar si todos entendieron la pregunta. Por supuesto, es posible que aún así no funcione como le gustaría, pero tiene más posibilidades. Lars Sjödahl nos dio ese pequeño y valioso consejo durante su charla relámpago "El daño hecho por los actos de silencio" en los Agile Testing Days 2013.

En la sección "Estrategias para afrontar la situación", más adelante en este capítulo, le daremos algunas ideas de prueba que pueden ayudarle a ganar esta batalla.

Zonas horarias

Uno de los principales problemas que enfrentan los equipos distribuidos son las zonas horarias. Cuando un subequipo, o todo el grupo de prueba, está a 12 horas de distancia, probablemente significa que hay poca o ninguna comunicación verbal, a menos que alguien trabaje hasta altas horas de la noche. Algunas organizaciones organizan que un equipo trabaje de noche para igualar horas con otro. Otros hacen que un miembro del equipo se una a una reunión diaria de forma remota a última hora de la tarde o temprano en la mañana para tratar de mantener cierto nivel de comunicación verbal.

Cuando la comunicación cara a cara no es una opción, los equipos deben encontrar una manera de describir sus necesidades para que un equipo no se quede inactivo, tal vez porque no sabía qué hacer a continuación. Otro ejemplo de problemas que enfrentan los equipos distribuidos es la falta de entornos para realizar pruebas.

El soporte debe estar disponible para todos los equipos, en todo momento, si los equipos utilizan una base de código común y un entorno de integración continua (CI).

La historia de Janet

En una organización teníamos equipos en dos países diferentes y practicábamos integración continua para que los testers que viven en Europa puedan venir a trabajar por la mañana y recoger cosas para probar. Uno de los problemas que descubrimos fue que no siempre tenían gente disponible para responder preguntas, por lo que el ciclo de retroalimentación real se alargó. Una de las soluciones que tenemos Lo que implementamos fueron correos electrónicos nocturnos: en qué habíamos trabajado, qué era sobresaliente, qué problemas existían. Los hicimos lo más completos posible. Había una diferencia horaria de siete horas, así que cuando eran las 4:00 p.m. hora, eran las 9:00 am nuestra hora. Intentamos que la gente llegara temprano. (7:30 am), así que teníamos al menos un par de horas de superposición si necesitábamos hablar en persona. Hoy en día, probablemente recibiría una llamada de Skype todas las mañanas.

Dependencias

Sea consciente de las dependencias entre grupos. Si está esperando un envío de personas en una zona horaria diferente y no lo tienen listo, es posible que deba esperar al menos otras 24 horas. Los miembros del equipo pueden iniciar otras tareas mientras esperan, lo que da como resultado el cambio de tareas entre múltiples compromisos. Discuta posibles dependencias en las reuniones de planificación y encuentre formas de hacerlas visibles o eliminarlas. Se pueden utilizar herramientas de seguimiento en línea para mostrar si los evaluadores están esperando que se entreguen las historias o si las reciben todas a la vez, o si hay demasiadas historias esperando a que alguien las acepte.

Planificación

Las sesiones de planificación son especialmente difíciles cuando no todos los equipos están en el mismo lugar. Es útil reunir físicamente a todo el equipo en un solo lugar para el inicio del lanzamiento o las reuniones de inicio del proyecto. Los equipos que han hecho esto han visto beneficios significativos en la forma en que trabajan cuando los miembros del equipo están de regreso en sus respectivas ubicaciones. En el Capítulo 9, “¿Estamos construyendo lo correcto?” y en el Capítulo 11, “Obtener ejemplos”, hablamos sobre cómo los evaluadores pueden ayudar probando el valor del negocio mientras aún se encuentra en la etapa de idea. Esos beneficios se aplican a equipos distribuidos, pero son mucho más difíciles de lograr si todo el equipo no está en un solo lu-

Experiencias trabajando en equipos distribuidos

Huib Schoots , quien trabaja Mejorar la Calidad de los Servicios en el Holanda, comparte lo que ha aprendido trabajando en equipos. en repartido

"Ir juntos es un comienzo. Mantenerse unidos es progreso.
Trabajar juntos es el éxito". -Henry Ford

He trabajado en muchos proyectos internacionales en mi carrera. Y sin excepción de los proyectos, siempre transcurrían mejor cuando yo estaba in situ. La colaboración mejoró, la comunicación fue mucho mejor y el trabajo Terminé más rápido. Un proyecto que hice en Yakarta siguió el mismo patrón; en Al principio estuve allí solo una semana cada mes, y luego estuve allí tiempo completo. ¡La colaboración mejoró dramáticamente! ¿Por qué? Porque nosotros nos conocimos mejor y nos tomamos el tiempo para superar nuestros diferencias culturales.

Cuando no están en el mismo lugar físico, las personas pueden esconderse más fácilmente y tienden a aislarla. La comunicación se vuelve más difícil y menos rico. Las cosas en las que todos están trabajando se vuelven menos transparentes, y para superar este problema utilizamos nuestras herramientas en combinación con más documentación completa. En mi experiencia, los equipos dependen demasiado sobre documentación y herramientas. En realidad, estos pueden hacer que sea más difícil mantener todo actualizado y la información no se compartirá tan fácilmente.

Las personas necesitan encontrarse entre sí para tener realmente éxito. Las organizaciones deben hacer que la mezcla sea lo más simple y obvia posible. Conseguir personas reunidas regularmente; darles tiempo para generar confianza y llegar a conocer unos a otros. La comunicación efectiva está en los detalles y Para reconocerlos tienes que conocer a tu equipo, así que gasta tanto tiempo juntos como sea posible.

En mi experiencia, lo más importante es reconocer que La comunicación es el mayor riesgo en el proyecto. cuando todo el mundo esta Conscientes de este riesgo, los equipos tienen mayores posibilidades de superar las dificultades que tendrán. Los equipos harán un esfuerzo deliberado para asegurarse que la comunicación sea efectiva; necesitan ser creativos para hacer lo mejor de lo que tienen.

Usamos mucho Skype y Google Hangouts, pero hay muchas herramientas. disponibles para facilitar la comunicación en línea. Usamos video para ver unos con otros, y esto nos ayudó a conocer mejor a la gente. Tú Al menos ten la sensación de saber con quién estás hablando. Nosotros también preparado para nuestras reuniones; los líderes del equipo de cada ubicación obtuvieron juntos (en línea) antes de la reunión real para discutir los aspectos más destacados.

Al hacer esto, siempre obtuvimos una buena agenda para cada reunión e invitamos solo a las personas que realmente necesitaban estar allí. Otra cosa que aprendimos de las videoconferencias es que esas reuniones toman más tiempo que las reuniones cara a cara normales. Además, es muy difícil discutir en profundidad con grupos grandes; por lo tanto, lo evitamos. Una ventaja de utilizar la videoconferencia es que puede grabar la reunión y reproducirla para las personas que no pudieron asistir.

Experimentamos con tableros ágiles en línea, pero en nuestra situación realmente no funcionaron. Fueron útiles en la comunicación con la otra ubicación, pero no trabajaron en mi equipo en la ubicación. No teníamos una descripción general de las cosas en las que estábamos trabajando, por lo que utilizamos una pizarra como nuestro panel ágil y actualizamos la herramienta en línea diariamente antes de la reunión con los otros equipos.

Para crear espíritu de equipo, probamos cámaras web en cada lugar para poder ver lo que estaba sucediendo al otro lado. Esto tiene una ventaja extra, ya que puedes ver quién está disponible para hablar. Funcionó bastante bien.

Pruebas Consideraciones

Las pruebas también tuvieron que hacer frente a dificultades de comunicación, y hacer preguntas, compartir información y emparejar se volvió más difícil.

Es más difícil involucrar a programadores en una ubicación diferente para mostrarles sus hallazgos o hacerles preguntas. La tecnología ayuda, pero aun así tuvimos que anotar los hallazgos y utilizar el rastreador de errores con más frecuencia de lo normal. Las pruebas de pares se distribuyeron y el uso de herramientas para compartir pantalla a través de la web resolvió parcialmente este problema.

Después de que descubrimos que las pruebas de pares funcionaban bastante bien usando herramientas, la comunicación mejoró y se volvió más frecuente.

Haciendo Él Trabajar

Para construir buenas relaciones, animo a los equipos distribuidos a que se reúnan tanto como sea posible. Ayuda a conocerse, a familiarizarse con la cultura y la ética laboral, pero también a optimizar la comunicación. Es especialmente importante utilizar la comunicación cara a cara para retrospectivas y discusiones en profundidad.

Los equipos distribuidos pueden funcionar, pero tenga en cuenta las consecuencias. Pruebe varias formas de comunicación, discútalas y evalúelas con frecuencia.

Adáptalos cuando no estén funcionando. Trabajar juntos se trata de relaciones, comunicación y personas. Cuando las personas están lejos unas de otras pero aún tienen que trabajar juntas, hay muchas maneras, incluidas varias herramientas, de hacerlo posible. Sin embargo, no importa qué tan bien funcionen las herramientas y cuán apasionados y talentosos sean sus colegas, los equipos distribuidos siempre no son óptimos.

Huib comparte algunas buenas estrategias para permitir que los miembros del equipo colaboren en las pruebas y ha proporcionado algunos enlaces que incluimos en la bibliografía de la Parte VII. Veamos algunas formas más de hacer posibles las pruebas ágiles en equipos distribuidos.

Estrategias de afrontamiento

Idealmente, los miembros distribuidos de su equipo pueden reunirse periódicamente en un lugar para discutir los desafíos que presenta la distancia geográfica. Si esto no es posible, aún puedes encontrar buenas maneras de comunicarte y construir relaciones. Hay muchas herramientas disponibles para permitir el emparejamiento remoto y la colaboración grupal. Vea la historia de "Investigación de usuarios" de los diseñadores de experiencias de usuario de Disney que colaboran de forma remota en el Capítulo 13, "Otros tipos de pruebas".

Integrando equipos

Nosotros, junto con otros profesionales, recomendamos que los miembros del equipo en diferentes lugares viajen con la mayor frecuencia posible para maximizar el "tiempo cara a cara" y las experiencias compartidas. Esta es la mejor manera de generar la confianza necesaria para trabajar juntos de manera productiva. Si es posible, recomendamos intentar agrupar equipos autosuficientes en torno a funciones, para reducir el problema de dependencia y ayudar al equipo a sentirse más cohesivo. Asegúrese de que todos los miembros del equipo tengan acceso a las partes interesadas del negocio para que puedan obtener respuestas a sus preguntas rápidamente. Tener entrenadores experimentados, líderes de equipo o expertos en la materia en cada ubicación ayuda a la cohesión de los equipos.

Los miembros del equipo en las distintas ubicaciones geográficas deben acordar la misma definición de "hecho". Hemos visto equipos en un lugar pensar

Ese Story Done significaba codificado y probado completamente, mientras que el otro equipo, del que dependían, pensó que se refería sólo al código completo, listo para las pruebas de integración. Esto provocó problemas y muchos sentimientos antagónicos entre los equipos.

Se necesita más tiempo y disciplina para ayudar a todos a comprender sus responsabilidades y obligaciones. Es posible que dedique más tiempo a identificar cómo interactúan las personas, cómo se toman las decisiones y cómo se divide el trabajo entre los equipos. Intente mantener las tareas e historias lo suficientemente pequeñas como para que puedan completarse de manera oportuna para que los otros equipos puedan trabajar con el

La falta de confianza es uno de los mayores obstáculos que enfrentan los equipos distribuidos. No pueden reunirse para hacer ejercicios de trabajo en equipo, tomar bebidas o comer. No tienen una manera fácil de establecer las relaciones necesarias para generar confianza. En lugar del contacto cara a cara, tómese el tiempo para conocer los otros equipos remotos, sus culturas y su área de trabajo. Encuentre alguna manera de tener interacción social en línea. Cree una página wiki donde las personas puedan publicar fotografías y compartir información personal. Configure un espacio para blogs personales para que los miembros del equipo puedan contar historias personales o profesionales. Tenga un canal de chat en equipo dedicado a intercambios sociales y a compartir chistes. Jueguen juntos videojuegos en línea.

La historia de Lisa

Mi equipo actual está separado por sólo dos zonas horarias. Aun así, necesitamos un
Mucho disciplina para mantener una buena comunicación. Pareja de programadores para todos
código de producción, y también a menudo emparejamos roles, como un probador
con un diseñador, o un diseñador con un programador. Cada miembro del equipo vigila
diligentemente el canal de chat del equipo en caso de que alguien pueda responder.
una pregunta o simplemente quiere compartir una imagen divertida. A veces el viernes
Por las tardes, los miembros del equipo juegan un videojuego colaborativo o miran
videoclips juntos.

Pero lo que probablemente sea más importante es que cada dos meses, el
Todo el equipo se reúne en un solo lugar durante una semana. Durante esta semana tenemos
actividades sociales juntas y una retrospectiva en equipo en persona. Entre
En esas ocasiones, las personas viajan a otros lugares para trabajar. el viaje es
caro, pero el mayor nivel de confianza y capacidad para comunicarse
nos permite hacer un mejor trabajo y más eficientemente. Con pocos probadores en el equipo.
En comparación con la cantidad de programadores, crear una atmósfera de comodidad y confianza
es especialmente útil.

Comunicación y colaboración

Cuando los equipos abarcan todo el mundo, necesitan encontrar formas de comunicarse que
abarquen el idioma, las zonas horarias, la cultura y los estilos de comunicación.
Todo el equipo debe estar al tanto de todas las tareas en las que se está trabajando, incluidas
las actividades de prueba. Las herramientas de colaboración deben funcionar por igual para
todos los miembros del equipo. En equipos colocados, los miembros del equipo pueden tener
conversaciones espontáneas. Los equipos distribuidos no pueden darse este lujo.
Necesitan crear conversaciones estructuradas y encontrar una manera de compartir
información con todos los miembros del equipo en todas las ubicaciones.



Figura 19-3 Aproveche las tecnologías: cree una persona virtual.

Cuando la mayor parte del equipo está en un lugar pero algunos miembros del equipo están trabajando desde otro lugar, tal vez desde casa, es fundamental que esos miembros del equipo tengan un tiempo cara a cara con el resto del equipo.

Lisa ha estado en equipos dispersos, a veces como uno de los miembros remotos del equipo, y ha experimentado los efectos positivos de la comunicación cara a cara, incluso si es virtual. Lisa se sintió parte del equipo cuando la incluyeron en las conversaciones. Uno de sus equipos creó una "Lisa virtual" utilizando un dispositivo de telepresencia, muy parecido al de la Figura 19-3. La llevaron a reuniones de planificación de iteraciones, reuniones diarias y a quien fuera su pareja del día. Virtual Lisa consistía en una computadora portátil sobre un carro rodante, con un micrófono de buena calidad que podía captar incluso conversaciones informales en la sala y una cámara web que podía controlarse de forma remota.

Esto hizo que Lisa se sintiera realmente parte del equipo en todos los sentidos porque podía participar en las conversaciones y no preocuparse por perderse algo.

Acordar una política de teletrabajo ayuda a promover el tipo de cultura y disciplina necesarias para que los equipos con miembros remotos trabajen de manera efectiva. Chris McMahon describió un ejemplo de política en su publicación de blog "Telecommuting Policy" (McMahon, 2009). La transparencia y la visibilidad pueden ser incluso más importantes en equipos distribuidos que en equipos ubicados.

Piensa qué información quieras compartir y cuál es la mejor manera de compartirla. ¿Cómo se puede transmitir información esencial de un vistazo? ¿Puedes crear una matriz de prueba simple para mostrar el progreso de las pruebas para una versión y colocarla en la página de inicio de la wiki del proyecto? El equipo anterior de Lisa tomó una fotografía de su tablero de tareas todos los días y la puso en la wiki para que los miembros remotos del equipo pudieran ver el estado actual del tablero físico.

En su discurso de apertura de los Agile Testing Days 2013 "Visualizing Quality" (Evans, 2013), David Evans recomendó mostrar la fotografía de cada miembro del equipo. (Figura 19-4) junto con sus historias y tareas en el sistema kanban o

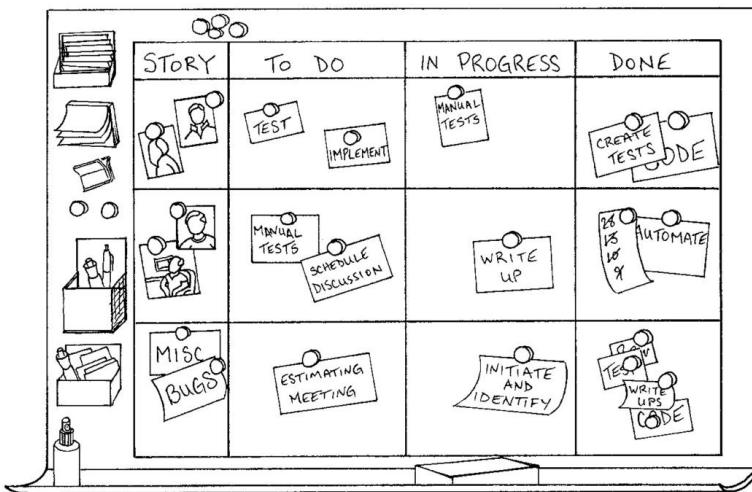


Figura 19-4 Conozca a los miembros de su equipo distribuido.

guión gráfico. Esto es particularmente útil para equipos distribuidos. Ayuda a todos a visualizar quién está anclando cada historia o tarea, y también muestra cuando alguien no tiene trabajo que hacer actualmente. Hemos conocido equipos que incluyen una foto o un avatar de la persona que presenta cada historia en su herramienta de seguimiento de proyectos en línea. Informaron que ayudó con la comunicación y ayudó a evitar que las historias cayeran en agujeros negros. El equipo anterior de Lisa hizo esto con su tablero de tareas físicas y se divirtió con las imágenes. Dado que las fotos del tablero de tareas se publicaban en la wiki todos los días, los miembros remotos del equipo podían unirse a la diversión.

Colaborando a través de pruebas



Los ejemplos y las pruebas son formas poderosas de comunicarse, pero los equipos a menudo no logran aprovecharlos. Las pruebas se pueden escribir en un lenguaje de dominio específico (DSL) que traspase las barreras del idioma. Lo ideal es que los miembros del equipo puedan colaborar al mismo tiempo, pero eso no siempre es posible. Si los evaluadores están en un país y los programadores en otro, tal vez los evaluadores puedan escribir pruebas, que los programadores puedan revisar antes de discutir la historia. Al comenzar temprano, cuando hay una gran diferencia horaria, los miembros del equipo pueden estar preparados y utilizar las pruebas para iniciar discusiones. Los propietarios de productos también deben ser parte de estas discusiones, examinando los ejemplos y las pruebas y continuando las conversaciones hasta que todo el equipo tenga una comprensión compartida de la historia.

Si la diferencia de zona horaria lo permite, intente el emparejamiento remoto para que el ciclo de retroalimentación sea breve. Janet escuchó un informe de experiencia sobre emparejamiento remoto en Agile 2013 realizado por Johannes Brodwell (Brodwell, 2013) que describía pruebas que se realizaron en Sri Lanka, mientras que el resto del equipo de entrega y los clientes estaban en Suecia y Noruega. Las zonas horarias no fueron un problema, por lo que los miembros del equipo pudieron conectarse de forma remota y colaborar mediante emparejamiento. Utilizaron herramientas como Skype para estar lo más cara a cara posible. Las herramientas para compartir pantalla y archivos, como GoToMeeting y Dropbox, facilitaron la colaboración. Para mantener la programación emparejada avanzando más rápidamente, utilizaron un enfoque Ping-Pong para el desarrollo basado en pruebas (TDD): un programador escribe una prueba y el otro programador escribe el código; luego cambian de roles. Este enfoque podría funcionar para actividades de prueba, como especificar pruebas o depurar fallas de pruebas. Uno de los mayores beneficios que encontró esta empresa fue que las parejas se conocieron mejor. Aprendieron los modismos y preferencias de cada uno y su respeto mutuo creció. Su proceso proporcionó capacitación en tiempo real para los nuevos miembros del equipo. El conocimiento del dominio se transmitió rápidamente, al igual que la comprensión empresarial.

Pruebas en alta mar

Algunas organizaciones creen que las pruebas en el extranjero son más baratas porque el costo percibido por persona-unidad de prueba es menor. Sin embargo, los gastos generales de las capas adicionales de comunicación y la falta de comunicación que ocurre entre personas cuyo primer idioma no es el mismo podrían compensar los ahorros en costos de mano de obra. A menudo, las pruebas en el extranjero significan “arrojar el código por encima del muro” para realizar pruebas y en realidad no se trata de obtener retroalimentación rápida.

La historia de Janet

Hace poco estuve hablando con una organización distribuida globalmente (Empresa A) eso estaba tratando de pasar a ser ágil. Las múltiples aplicaciones de la empresa fueron muy estrechamente acoplados, pero propiedad de múltiples proveedores, así como de los propios equipos internos de desarrollo ágil de la empresa que respaldaban el front-end. Uno de los proveedores estaba utilizando un sistema basado en flujo para desarrollar su aplicación, pero debido a que se entregó a muchos clientes diferentes, el tiempo no siempre coincidía con las necesidades de la empresa A. Los otros vendedores estaban usando métodos tradicionales de desarrollo gradual y cerrado. Una prueba interna El equipo manejó las pruebas de integración, pero las pruebas de aceptación del usuario (UAT)

fue subcontratado a otro proveedor cuyos probadores trabajaron tanto en el sitio como en el extranjero para mantener los costos más bajos para el cliente.

Se enfrentaron a muchos problemas, incluido cómo obtener funciones cuando Estaban involucradas múltiples aplicaciones y proveedores, y tenían que confiar en requisitos y documentos de especificaciones. La empresa A quería tomar ventaja de "seguir el sol", es decir, tener trabajo en constante progreso por parte de un equipo en al menos una zona horaria. Fue necesario un esfuerzo adicional para lograr asegurarse de que el traspaso de un grupo al siguiente fuera perfecto. Por ejemplo, crearon un puesto en la oficina de América del Norte para trabajar con el equipo de pruebas en alta mar para actuar como enlace entre los equipos. ellos también tenían una persona de guardia para que haya apoyo para el equipo offshore si la prueba el entorno dejó de estar disponible.

Cuento esta historia para mostrar las dificultades que enfrentan algunos equipos cuando una organización no comprende los cambios culturales y estructurales necesarios para Permitir que los equipos aprovechen la agilidad. Esta organización tiene una difícil tiempo por delante para alinear a todos estos grupos y proveedores para entregar en pequeñas porciones pero está decidido a hacerlo funcionar.

Una forma de hacer que las pruebas en el extranjero funcionen con un enfoque ágil podría ser involucrar a un equipo de integración de sistemas desde el principio y brindarles los flujos de trabajo completos a los equipos para que haya coherencia en la entrega. Quizás se podrían desarrollar flujos específicos en orden de prioridad, utilizando el enfoque de "hilos de acero" descrito en Pruebas ágiles (p. 144), trabajando en cortes delgados de un extremo a otro de forma incremental. Esto permite probar los flujos antes para reducir el riesgo. Esto puede prevenir, o al menos mitigar, el problema en el que un equipo entrega su parte del sistema pero interrumpe todas las demás aplicaciones asociadas con la capacidad.

El desarrollo de múltiples aplicaciones al mismo tiempo aumenta la probabilidad de que surjan problemas de integración. En el Capítulo 18, "Pruebas ágiles en la empresa", mencionamos la idea de un equipo de integración de sistemas. Las organizaciones distribuidas complejas pueden justificar tener un equipo que pruebe continuamente el sistema general, incluidas todas las aplicaciones empresariales. En el ejemplo de Janet, tal vez un equipo virtual formado por miembros de los equipos de integración y UAT garantizaría que todos entiendan lo que se está entregando de forma iterativa.

Muchas empresas que ofrecen pruebas como servicio han comenzado a ofrecer pruebas ágiles. Si estas organizaciones encuentran formas de proporcionar retroalimentación rápida,

puede tener cierto éxito. Si su empresa ofrece este tipo de servicio, trabaje con los equipos de producto para determinar el mejor lugar para que los evaluadores agreguen valor.

Quizás pueda incorporar evaluadores en los equipos de productos del cliente.

A menudo, las empresas de servicios de pruebas ofrecen UAT o pruebas de integración de sistemas. Pueden agregar valor si crean buenos canales de comunicación entre el equipo de producto y el equipo de pruebas subcontratado. Por ejemplo, se podrían pasar incrementos de funciones para realizar pruebas mientras la codificación aún está en progreso, en lugar de esperar hasta que toda la versión esté lista para probar.

La Figura 19-5 muestra una forma posible de realizar la entrega a un equipo de pruebas de integración del sistema al final de cada iteración y luego realizar la UAT cuando se completa una función. Esto requiere que todos los equipos de producto trabajen juntos para producir características comprobables para una entrega frecuente.

Las organizaciones de pruebas como servicio que trabajan con equipos ágiles necesitan una relación diferente con los equipos de desarrollo ágiles que con los equipos que utilizan desarrollo por fases y controlado. Necesitan nuevas formas de especificar contratos y deben trabajar más duro para interactuar con los equipos de desarrollo de clientes. Consulte los enlaces en la bibliografía de la Parte VII, “¿Cuál es su contexto?” para conocer opciones que puede explorar para contratos ágiles.

Si su equipo ágil utiliza equipos de prueba costa afuera o cerca de la costa, es una buena idea tener al menos algunos evaluadores ubicados con el equipo de desarrollo para representar al equipo de pruebas costa afuera. Pueden asistir a la planificación.

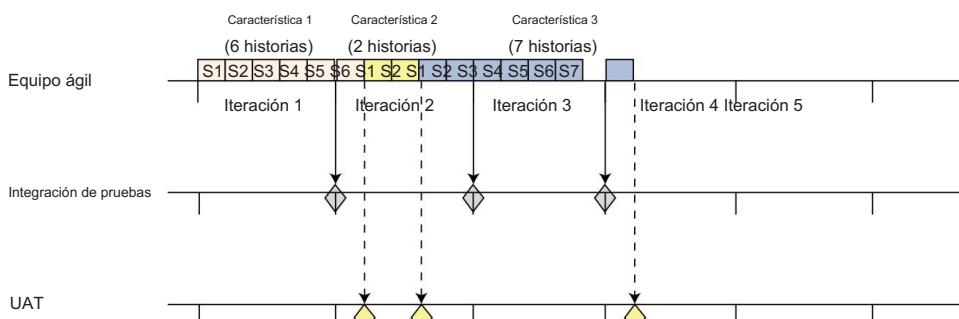


Figura 19-5 Pruebas como servicio para pruebas de integración y UAT

reuniones y reuniones diarias, colaborar con programadores y mantener al equipo extranjero informado. Reconozca que, si bien un escenario offshore puede haber percibido beneficios para la empresa, requiere mucho más trabajo y disciplina para mantener una comunicación y colaboración suficientes.

Trabajar como un equipo de pruebas en alta mar

Parimala Hariprasad , que bloguea en www.curioustester.com, analiza los desafíos y las soluciones que encontró trabajando en equipos de en pruebas en el extranjero.

He trabajado en muchos equipos que manejan pruebas de forma remota. Algunos de estos Son startups repartidas por todo el mundo con recursos limitados y infraestructura. Estas pequeñas empresas emergentes a menudo colaboran con otros desarrolladores, diseñadores y proveedores de infraestructura para implementar sus ideas de negocios. Esto significa que mi equipo a menudo actúa como prueba en alta mar. equipo para varios grupos a la vez.

Introducción y Proyecto Contexto

Cuando comienza cualquier proyecto de prueba, recibimos presentaciones en correos electrónicos. Nosotros envíe un comentario animado de cinco líneas sobre cada evaluador de nuestro equipo en el correo electrónico con una imagen divertida. Esto crea un ambiente informal para comenzar. El Luego, al correo electrónico le siguen llamadas de audio para reuniones diarias. Las demostraciones son normalmente se realiza mediante Skype, Google Hangouts, WebEx u otras herramientas de videoconferencia; Hemos explorado una buena combinación de código abierto y herramientas pagas. En mi experiencia, presentarse cara a cara ha funcionado mejor. para mi equipo porque cuando alguien habla, ponerle cara La voz da una ventaja personal. Cara a cara no significa necesariamente llevando a su equipo a la ubicación principal. Se podría facilitar compartiendo fotos de cada uno, videollamadas, videos divertidos de un día típico de trabajo, Etcétera.

Nos adentramos en el contexto del proyecto, exploramos el producto bajo prueba, obtenemos información adicional, hacer preguntas y comenzar a trabajar en el examen estrategia. Declaro temprano durante las conversaciones iniciales que el trabajo avanza Será lento durante los primeros días mientras aprendemos y exploramos. También compartimos nuestros entregables con las partes interesadas las 24 horas del día, los 7 días de la semana utilizando herramientas como Dropbox y Google Docs para que puedan ver quién está trabajando en un tarea particular en cualquier momento y qué cambios se han realizado desde el día anterior. Esto proporciona una gran visibilidad a nuestros grupos de interés. en ubicaciones principales.



Protocolo de trabajo y**Propiedad**

Reiteramos el protocolo de trabajo acordado durante la etapas de negociación del proyecto (oh, sí, nuestro equipo hace eso). Nosotros hablar abiertamente sobre los horarios de trabajo en zonas horarias iguales o diferentes, evaluadores quién puede proporcionar respaldo (probadores paralelos), acuerdos de nivel de servicio (SLA) para problemas críticos, etc. Establecemos reglas comunes que entran como solicitudes de colaboración, horarios de reuniones diarias, formato de informes de prueba y procesos, disponibilidad de infraestructura y aprobaciones necesarias para empezar a trabajar desde el primer día. Con algunos clientes, compartimos muestras de nuestro trabajo anterior (enmascarando información confidencial) para ayudar. ellos comprendan mejor el valor que aportamos como un equipo de prueba remoto.

En los equipos distribuidos, el mayor desafío es la falta de propiedad entre equipos en ubicaciones remotas y principales. Cuando las cosas van mal, algunos miembros del equipo recurren a juegos de culpas. Además de las habilidades, la actitud juega un papel clave en cómo las personas se apropián de su trabajo. Muchos equipos con los que trabajé tenían una mentalidad en la que todos se sentían responsables del éxito o el fracaso del equipo. Se reunieron constantemente para colaborar con todos los equipos con un objetivo final común en mente. Tuvieron una visión para el producto que estaban probando. No vieron las pruebas como una proceso aislado, sino como un facilitador clave para ayudar a crear excelentes productos.

Diversion y Familia

Uno de los miembros de nuestro equipo no había visto a uno de los cofundadores de una startup, aunque interactuaba con él a diario. una nueva idea nació. Tomamos fotografías y videos divertidos de nuestro equipo en nuestros cubículos y los compartimos con los miembros del equipo en la ubicación principal. Ellos respondió con una gran sonrisa y correspondió a esta acción; habla de una jefecto mariposa! También enviamos flores y regalos a nuestros clientes y sus miembros del equipo en ocasiones especiales. Si supiéramos de sus viajes planes para nuestra ubicación, los hospedamos a nuestra costa, los sacamos para cenar y hablamos del trabajo mientras comíamos. Esto nos ha ayudado a construir Fuertes vínculos con nuestros equipos en las principales ubicaciones.

La última tecnología de videoconferencia también nos brinda el lujo de interactuar con personas en un ambiente informal donde es cara a cara—¡bueno, casi! No muchas organizaciones pueden permitirse viajar hasta el cliente. ubicaciones. Las herramientas de videoconferencia de bajo costo pueden reducir los costos de viaje. En algunos casos, algunos de mis equipos fueron trasladados en avión a lugares principales después Los clientes vieron el valor que agregamos a su trabajo. Esto fue resultado de la credibilidad que construimos con ellos mientras trabajábamos de forma remota.

Visibilidad y**Comentario**

La confianza y la credibilidad desempeñan un papel importante a la hora de definir cómo se realiza el trabajo. entre la ubicación principal y los equipos de pruebas en alta mar. En el equipo

en el que trabajé, brindé gran visibilidad a nuestro día a día, que incluía informes de estado diarios, reuniones diarias, llamadas semanales, correos electrónicos separados para problemas críticos, respaldo para los evaluadores que estaban de licencia o descansos prolongados, etc. No necesitábamos una cámara instalada en nuestro escritorio para hacer eso. Éramos muy transparentes. Buscar comentarios durante las retrospectivas de proyectos y responder a ellos son otras claves. acciones que generen confianza y credibilidad.

En la siguiente barra lateral, “¿Por favor el verdadero gerente se levantará?”, Parimala comparte algunos desafíos más que están más basados en conflictos: trabajar con gerentes que tienen diferentes objetivos, o gerentes que solo saben decir que sí. Explica cómo trabajó con su equipo para resolver algunos de los conflictos.

¿Podría ponerse de pie el verdadero gerente?

En un equipo distribuido para el que trabajé, mi equipo de pruebas reportaba a un gerente de línea que manejaba operaciones como la adquisición de sistemas, la aprobación de solicitudes de licencia, la realización de evaluaciones de desempeño y la determinación de compensación en la ubicación remota. Técnicamente, mi equipo informó a un director técnico que se sentó en la ubicación principal con su equipo allá. Muchas decisiones solían ser impuestas a nuestro equipo sin apenas cualquier discusión o información. Muchas veces tuvimos que hacer a ciegas lo que nos pidieron que hicieramos. Se suponía que no debíamos decir que no. Nosotros no estábamos allí se supone que debe hacer preguntas. Al menos eso es lo que nuestro superior directo nos dijo. La forma en que lo abordamos fue generando credibilidad sobre mi contribución del equipo al proyecto. Con el tiempo, la dinámica del equipo entre nosotros cambió para mejor. A veces, es posible que necesites trabajar más duro para crear una cultura de trabajo saludable entre los equipos.

En algunos proyectos, es común que el superior jerárquico y el jefe El administrador de ubicación tiene una comprensión diferente de lo que es importante. En tales casos, los equipos sufren porque no saben a quién escuchar y a quién complacer. Si mantiene contento al superior directo, el otro gerente va a gritar desde la azotea sobre los pobres entregables. Si mantiene contento al otro gerente, su desempeño Las valoraciones se verán afectadas: así es como se comportan los evaluadores. amenazado por momentos. Esto lleva a muchos evaluadores a intentar complacer a ambos. ellos tanto como sea posible manteniéndolos a ambos en la oscuridad. Semejante La política de oficina termina dañando la moral del equipo y eventualmente desintegrándolo.

Un caso extremo es cuando los equipos de pruebas en alta mar terminan diciendo que sí. a cada demanda que proviene de la ubicación principal ya sea razonable o no. Personalmente he trabajado en un par de proyectos. donde los equipos de pruebas en el extranjero se comprometieron demasiado sin tener en cuenta los desafíos de entrega. Esto a su vez genera estrés debido a decisiones poco realistas. Expectativas.

Manejo de conflictos de Interés (Cuando Díoses Ir Loco)

La verdadera fuerza y carácter de cualquier equipo se revelan cuando surgen conflictos. surgir. A veces, las conversaciones sobre conflictos duran una eternidad. El Los siguientes métodos han ayudado enormemente a los equipos con los que trabajo:

- Cambiar de medio: si un tema se rebota más y más más de tres veces en un correo electrónico, es hora de levantar el teléfono o llamar para una reunión.
- Regla de las 24 horas: está bien escribir un correo electrónico acalorado; solo espera hasta que tu Cálmate antes de enviarlo, tal vez 24 horas.
- Regla del cara a cara: Iniciar conversaciones difíciles es una de las cosas más difíciles de hacer en un entorno lleno de conflictos. A Un comienzo equivocado puede tener consecuencias devastadoras. yo entreno a mi equipo iniciar conversaciones difíciles con empatía, cara a cara (en vídeo o en persona). Comenzamos planteando el problema y el contexto en el que ocurre, explica qué intentos hemos hecho para resolver el problema y mantener una resolución positiva. En casos donde ya se sabe que dos individuos tienen conflictos, Traer a un colega común/influencia positiva para que se haga cargo. y ayudar a resolver el conflicto.

En mi experiencia, las habilidades, las barreras del idioma y las zonas horarias han sido pequeños desafíos. Pasión, actitud, coraje y excelencia son más difícil de aprender pero necesario para trabajar en armonía en un entorno distribuido.

configuración.

Parimala plantea problemas que los equipos tal vez no anticipen, como mensajes contradictorios de gerentes en diferentes ubicaciones y una falta de sentimiento de propiedad por parte de los equipos en diferentes ubicaciones. No puedes anticipar todos los problemas, pero tener algunas pautas, como las reglas de Parimala sobre cambiar de medio cuando el estilo de comunicación actual no funciona, o tener conversaciones difíciles cara a cara, te ayudará a lidiar con los problemas a medida que ocurren. .

Ideas de herramientas para equipos distribuidos

A lo largo de este capítulo, hemos mencionado herramientas que pueden ayudarle a resolver un problema en particular. Aquí hay un resumen de lo que creemos que podría ayudarlo. Si cree que una herramienta en particular funcionará en el proceso de su equipo, experimente con ella durante una o dos iteraciones. La herramienta debe adaptarse al proceso, no al revés, así que no actualice su proceso para que se ajuste a la herramienta. Deben ser técnicas simples, como iniciar una conversación improvisada de "Tres Amigos" para discutir los requisitos de una historia o emparejarlos para pruebas exploratorias. Si la curva de aprendizaje es demasiado pronunciada o las prácticas demasiado complicadas, no se adoptarán a largo plazo.

Herramientas de comunicación

Utilice el correo electrónico para transmitir información unidireccional, como seguimiento de conversaciones individuales. Dado que los correos electrónicos pueden malinterpretarse o malinterpretarse fácilmente, intente revisar los correos electrónicos sobre temas delicados con otra persona antes de presionar el botón Enviar.

Utilice tableros de tareas/historias/kanban físicos o virtuales para realizar un seguimiento del progreso y hacer visibles las actividades de prueba. Pruebe primero las soluciones más simples. Si es suficiente publicar una foto del tablero kanban en la wiki dos veces al día, hazlo. Si una herramienta de seguimiento en línea es más apropiada, pruebe más de una para ver cuál se adapta mejor. Algunas herramientas de seguimiento no están diseñadas para adaptarse a tareas de prueba, pero normalmente puedes encontrar una solución creativa.

Herramientas de colaboración

Las herramientas de teléfono, protocolo de voz sobre Internet (VoIP) y videoconferencia permiten que dos o más personas colaboren y comparten ideas. Experimente para ver qué herramientas funcionan mejor para usted, ya que cada una parece tener sus propios problemas. Usar algo tan simple como Skype, Google Hangouts o Zoom para tener una conversación tripartita puede evitar muchos malentendidos, como cómo debe comportarse una función o qué versiones de navegador son compatibles.

Sea consciente de los problemas de idioma y recuerde hablar clara y lentamente si los participantes de la llamada no comparten un primer idioma común. Utilice auriculares cuando sea apropiado para tener las manos libres para escribir, tomar notas o

Busque información, pero si está en una reunión de grupo, preste atención a la conversación en lugar de realizar múltiples tareas. Utilice vídeo si el ancho de banda lo permite; permite a las personas conectarse de forma mucho más personal. Esto significa que todos los miembros del equipo necesitan cámaras web y micrófonos de buena calidad. Piensa en la historia del dispositivo de telepresencia virtual de Lisa en la sección de estrategias.

La historia de Lisa

Cuando me uní a mi equipo actual, habían estado usando Skype solo de voz para enfrentamientos diarios. Empezamos a utilizar la función de vídeo de Skype. Esto ayudó a todos a sentirse más conectados y la comunicación visual agrega valor.

Sin embargo, cada día, se cortaba la llamada de al menos un participante, o el audio haría eco o comenzaría a "robotizar", lo que significa que las voces sonarían como un robot mecánico hablando.

Decidimos probar Google Hangouts para reuniones y reuniones. Al principio esto fue una gran mejora. Pero después de unos meses, comenzó a mostrar síntomas similares. problemas a los que habíamos experimentado con Skype.

Nuestro siguiente experimento para reuniones y reuniones fue Zoom. Hasta ahora, esto ha sido la solución más confiable, aunque las sesiones pueden congelarse y el audio puede hacer eco.

Nuestros programadores se emparejan todo el día y, dado que estamos distribuidos, esto significa que siempre hay algunos pares remotos. También utilizan Skype, Google Hangouts o Zoom para realizar el emparejamiento. Estas herramientas suelen funcionar mejor cuando hay sólo dos participantes, pero parecen pasar por ciclos de problemas.

Cuando se actualiza a nuevas versiones. A veces también experimentamos ancho de banda. asuntos. Recibimos ayuda de nuestro departamento de TI, discutimos los problemas en nuestras retrospectivas y continuamos experimentando con herramientas de videoconferencia.

Para algunas situaciones, el chat de texto es suficiente. Dos o más personas pueden compartir ideas fácilmente, ¡especialmente si escriben rápido! Hemos escrito gran parte de este libro a través de Google Talk, haciendo preguntas rápidas mientras escribimos. El chat de texto te permite más tiempo para pensar qué decir. También le permite hacer ping a alguien que podría estar en medio de algo en este momento pero que puede comunicarse con usted en unos minutos. Funciona bien para preguntas que no son urgentes. Si la discusión se complica, puedes saltar a la videoconferencia.

Como se mencionó anteriormente, un chat grupal de equipo es una forma efectiva para que un equipo distribuido se comunique de manera no intrusiva. El software de chat grupal se puede configurar con sonidos de alerta, de modo que si alguien escribe un nombre, esa persona sabrá que debe revisar el chat. Esto permite que el equipo esté concentrado en el trabajo pero aun así busque y brinde ayuda según sea necesario.

Los wikis se utilizan ampliamente en muchas empresas, mientras que otras utilizan otras herramientas de colaboración como SharePoint. Estas son buenas maneras de registrar decisiones y compartir información entre múltiples ubicaciones, pero asegúrese de que estén bien organizadas y sean fáciles de buscar.

Los mapas mentales son una poderosa herramienta de colaboración para muchas actividades relacionadas con las pruebas, como la planificación de pruebas, la lluvia de ideas sobre escenarios de prueba y el seguimiento de resultados. Hay muchos productos de mapas mentales disponibles que admiten la colaboración en tiempo real. Experimente con diferentes ofertas para ver cuál funciona mejor con la infraestructura de su equipo.

Para emparejar pruebas o codificar de forma remota, necesita compartir el escritorio para que cada persona pueda ver y controlar la pantalla como desee. Puede utilizar la pantalla compartida integrada en su sistema operativo o probar una variedad de soluciones, como una red privada virtual (VPN), Skype y Office Communicator. Obviamente, asegúrese de que todo lo que utilice sea seguro.

Como señalamos en Agile Testing (págs. 82 y 83), los sistemas de seguimiento de defectos (DTS) no son una buena herramienta de comunicación, y continúa el debate en la comunidad ágil sobre si los errores deben registrarse o simplemente corregirse y documentarse con una prueba automatizada. Experimente con diferentes enfoques para encontrar cuál funciona mejor para su equipo. Revise periódicamente su proceso de manejo de defectos para ver si lo que está haciendo todavía encaja.

Los ciclos de retroalimentación breves son el corazón de un equipo ágil. Necesitamos una infraestructura confiable para respaldar esa retroalimentación rápida. La CI, los entornos aislados para desarrolladores, los servidores de prueba, las bases de datos, las unidades compartidas, las máquinas virtuales, los laboratorios de pruebas y otras infraestructuras son fundamentales para cualquier equipo pero, como todo lo demás, suponen un desafío aún mayor para los equipos distribuidos. Una CI verde garantiza que el equipo que se encuentra al otro lado del mundo pueda ser productivo tan pronto como se pongan a trabajar. El hardware y el software que necesitan los miembros del equipo en diversas ubicaciones deben ser confiables. Los miembros del equipo remoto necesitan la capacidad de resolver sus propios problemas de infraestructura o tener ayuda disponible en

No olvide incluir a sus administradores de sistemas y profesionales de DevOps cuando equipos de diferentes ubicaciones se reúnan o realicen actividades de formación de equipos en sus propias ubicaciones. Asegúrese de que las personas que desempeñan estos roles en varios lugares se mantengan conectadas entre sí. Pueden colaborar para construir la infraestructura que mejor respalde a los equipos distribuidos (Hagberg, 2013).

Si su equipo está distribuido, es posible que tenga una mayor necesidad de probar su infraestructura. Los equipos ahora están probando su infraestructura y entornos con pruebas de regresión automatizadas, del mismo modo que prueban su código de producción. Consulte el Capítulo 23, "Pruebas y DevOps", para obtener más información sobre las pruebas de infraestructura.

Utilice las retrospectivas de su equipo para identificar impedimentos a la colaboración y la comunicación en su equipo distribuido. Pruebe diferentes técnicas de lluvia de ideas, como mapas de impacto y mapas mentales, para establecer objetivos de mejora y pensar en diferentes soluciones que podría probar.

Recomendamos probar dos enfoques posibles uno al lado del otro para facilitar la comparación.

Resumen

La tecnología actual ofrece buenas formas para que los miembros del equipo distribuido colaboren y se comuniquen. Sin embargo, su equipo distribuido necesitará disciplina y creatividad adicionales para mantener a todos involucrados en las pruebas antes, durante y después de la codificación. En este capítulo, analizamos muchas formas en que los equipos distribuidos pueden tener éxito a la hora de incorporar calidad a su software y prevenir defectos.

- Cuando hablamos de equipos distribuidos, incluimos equipos dispersos con miembros remotos, equipos con subequipo en varias ubicaciones y equipos en el extranjero. Todas estas situaciones pueden obstaculizar la responsabilidad de todo el equipo por las actividades de calidad y pruebas.
- Sea consciente de las barreras lingüísticas y culturales. Los equipos de ayuda en diferentes ubicaciones se sienten seguros y libres de hacer preguntas. Deles tiempo para asegurarse de que comprendan lo que se ha dicho.

- Reúna a todos en persona con frecuencia. Cuando eso no sea posible, utilice reuniones por video y pruebe algunas actividades divertidas para ayudar a los miembros del equipo a desarrollar sus niveles de comodidad y confianza.
- Si es posible, tenga gerentes, clientes o sus representantes, entrenadores y líderes en diferentes disciplinas disponibles en cada ubicación.
- Las pruebas escritas en un DSL compartido ayudan a garantizar que todos los miembros del equipo de desarrollo y del cliente tengan una comprensión común de cómo deben comportarse las funciones.
- Experimentar con las herramientas de comunicación y colaboración discutidos en este capítulo y busque más si es necesario para encontrar los que funcionan mejor para su equipo.
- Utilice CI y otra infraestructura para mantener el código integrado y actualizado, y que los ciclos de retroalimentación sean breves.

Esta página se dejó en blanco intencionalmente.

Capítulo 20

Pruebas ágiles para dispositivos móviles y sistemas integrados



Los seres humanos llevan décadas integrando software en dispositivos y máquinas. Al principio, esto era terreno de la ciencia espacial. La Computadora de Orientación Apollo, desarrollada para el programa espacial Apollo, es uno de los primeros ejemplos (Wikipedia, 2014c). El uso de software integrado se extendió a medida que los microprocesadores y microcontroladores se hicieron menos costosos y estuvieron más disponibles.

¿Qué queremos decir cuando utilizamos términos como aplicación móvil y software integrado? Nos gusta la definición que utiliza Jon Haqar (Haqar, 2014):

El software móvil se puede encontrar en teléfonos inteligentes, automóviles y otros dispositivos que se mueven. Por lo general, los sistemas móviles tienen conexiones de red Wi-Fi y/o celulares, funcionan con baterías, tienen una gran cantidad de plataformas de hardware, tienen diversas limitaciones de recursos, como el tamaño de la pantalla, entradas limitadas del usuario y cantidades más pequeñas de memoria, espacio de almacenamiento y Velocidad de CPU que una computadora portátil o de escritorio.

Un dispositivo integrado es un dispositivo electrónico que contiene software, pero es posible que los usuarios no aprecien completamente que están utilizando software debido a un hardware único y una interfaz de usuario limitada. Ejemplos de dispositivos de software integrados incluyen sistemas de frenos en automóviles, sistemas de control en aviones, dispositivos médicos como marcapasos e interruptores de luz "inteligentes". La programación de software móvil e integrado presenta desafíos únicos de programación y prueba en comparación con los sistemas de TI.

Los dispositivos integrados patentados, como monitores cardíacos y equipos de salud y fitness, enfrentan desafíos de prueba similares. Muchos de estos productos de software integrados y dispositivos propietarios se encuentran en industrias reguladas como la atención médica y la aeroespacial. Nos centraremos en las pruebas ágiles en entornos regulados en el Capítulo 21, "Pruebas ágiles en entornos regulados".

La agilidad es especialmente importante en un mercado que cambia rápidamente como el de las aplicaciones móviles, y el desarrollo ágil es una opción natural para los equipos en ese espacio. Con la explosión en la cantidad y los tipos de dispositivos móviles, desde teléfonos inteligentes hasta computadoras "portátiles", los equipos más ágiles enfrentan el desafío de desarrollar aplicaciones integradas y móviles.

En el Capítulo 12 de Pruebas ágiles, describimos un ejemplo de prueba de un sistema que utilizaba software integrado. Hemos reunido más ejemplos aquí para ilustrar cómo los valores, principios y prácticas de las pruebas ágiles ayudan a los equipos que ofrecen productos integrados y móviles, con más enfoque en los dispositivos móviles.

Similares, pero diferentes

Los valores, principios y prácticas de las pruebas ágiles se aplican al software móvil e integrado tal como se aplican a cualquier otro tipo de aplicación de software. Ciclos cortos de retroalimentación, colaboración con el cliente, lo más simple que podría funcionar, desarrollo basado en pruebas (TDD), guiar el desarrollo con pruebas y un compromiso de todo el equipo con la calidad, todo se aplica al software de prueba que se ejecuta en dispositivos móviles. o como parte de una solución de software más grande.

Sin embargo, definitivamente existen diferencias. Como señala Jonathan Kohl (Kohl, 2013), es poco probable que cogamos nuestros monitores o portátiles y los agitemos. No nos vamos a pasar las manos por sus pantallas, aunque eso está cambiando. Por ejemplo, Janet acaba de comprar una computadora portátil nueva con Windows 8.1 y usó la pantalla táctil para desplazarse mientras editaba. Ciertamente usaremos nuestras computadoras portátiles en la cafetería local, pero es menos probable que las usemos mientras viajamos en el metro o caminamos por las montañas. No cambiamos continuamente de la orientación vertical a la horizontal en las computadoras portátiles, pero hacemos estas cosas todo el tiempo con nuestros dispositivos móviles, que están llenos de sensores, cámaras y otras comodidades. Entonces están todos

los demás componentes que permiten el funcionamiento de los dispositivos móviles, como torres de telefonía celular, satélites GPS y baterías.

La cantidad y variedad de los diferentes dispositivos móviles añaden otro nivel de complejidad a las pruebas de aplicaciones móviles. Incluso el proveedor de servicios puede marcar la diferencia. Intentar probar cada permutación de plataforma y dispositivo se acerca a lo imposible.

Hay herramientas para ayudar. Podemos aprovechar el software de análisis para ver qué hacen los usuarios cuando el software falla. Si muchos usuarios abandonan una aplicación móvil en una vista determinada, lo sabremos y podremos intentar descubrir qué problema de usabilidad podría haberlo causado. Se trata de obtener feedback rápido, sólo que esta vez es de los propios usuarios.

El Ministerio de Pruebas creó un mapa mental de ideas de pruebas móviles que se puede ver en línea (Sherry, 2013). Úselo para generar sus propias ideas y crear su propio mapa mental específico para su aplicación. La Figura 20-1 muestra el nodo de datos del mapa mental como ejemplo.

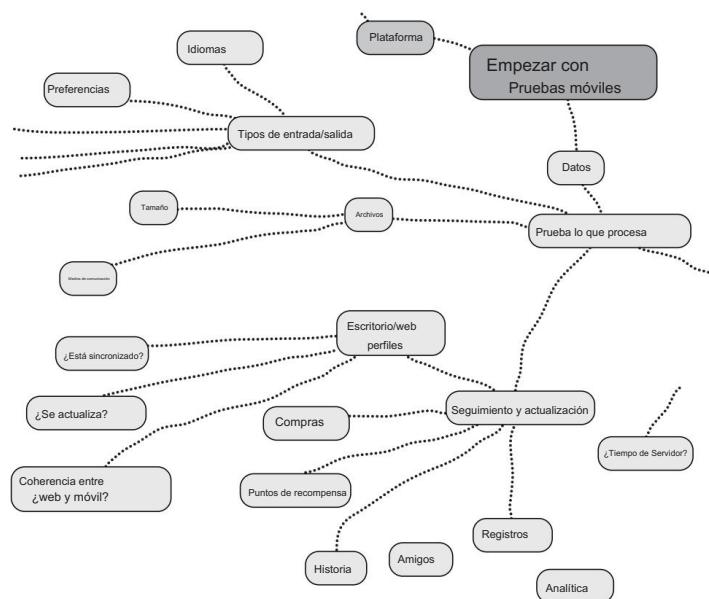


Figura 20-1 Mapa mental del Ministerio de Pruebas para aplicaciones móviles

Es más probable que los productos de software integrados se limiten a un solo dispositivo y entorno. La máquina elíptica de su gimnasio se utiliza mucho, pero permanece en un solo lugar y su software se ejecuta de formas predefinidas. Sin embargo, cada vez más productos de consumo, incluso automóviles, ofrecen API que permiten a los usuarios personalizar sus propias aplicaciones. A medida que utilizamos más software en nuestra vida diaria, crear el producto adecuado para todos los clientes se vuelve más desafiante.

Las pruebas son fundamentales

Las pruebas tienen el potencial de agregar aún más valor en el desarrollo de productos integrados y móviles que en otros productos de software. Como ha señalado Julian Harty (Harty, 2015), si lanza una nueva aplicación móvil y recibe críticas negativas de inmediato, es posible que su aplicación nunca se recupere. Si los usuarios no están satisfechos con su producto, es posible que obtenga una calificación de una estrella en cuestión de minutos, lo que podría significar el final de su aplicación antes de comenzar.

Es difícil solucionar los problemas lo suficientemente rápido como para revertir la primera impresión. Su cliente tiene muchas más opciones que están fácilmente disponibles en la App Store. Sólo para que su producto despegue, debe tener confianza en todos sus atributos de calidad. Las técnicas de aprendizaje validadas, como las pruebas A/B, de las que hablamos en el Capítulo 13, "Otros tipos de pruebas", pueden ser un tipo apropiado de prueba a utilizar.

Algunas dimensiones de la calidad pueden ser más importantes en algunos tipos de productos de software que en otros. Por ejemplo, es importante verificar la capacidad de descargar una aplicación a través de su proveedor de telefonía móvil o mostrar la interfaz de usuario (UI) en una pantalla pequeña al probar un teléfono inteligente, pero no es tan crítico en una tableta. La naturaleza personal de las aplicaciones móviles añade dimensiones que no se ven en los productos de software basados en la web o de escritorio. Julian Harty señala que dado que las aplicaciones móviles pueden querer acceder a su ubicación, su lista de contactos, su calendario y otra información muy personal, los usuarios deben sentir que pueden confiar en el producto.

Los evaluadores, programadores y diseñadores de experiencia de usuario (UX) pueden trabajar juntos para garantizar que los usuarios finales se sientan seguros y encuentren la aplicación valiosa y fácil de usar. Peter Morville (Morville, 2004) describe diferentes facetas a considerar para la experiencia del usuario, como se muestra en la Figura 20-2.

Probar aplicaciones móviles es, por supuesto, diferente a verificar aplicaciones de alto riesgo, como computadoras que guían cápsulas espaciales y cohetes.

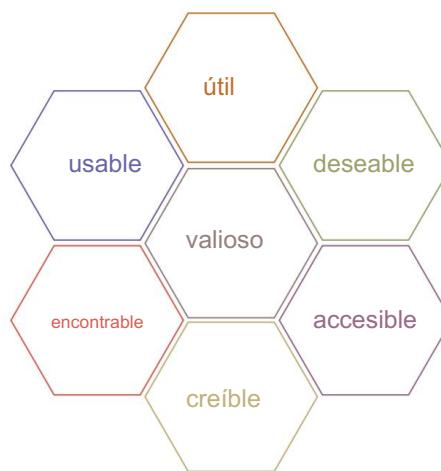


Figura 20-2 El panal de experiencia del usuario de Semantic Studios

que utilizan software integrado. También se desarrolla software integrado para productos críticos para la seguridad, como dispositivos médicos, donde los resultados de las pruebas pueden ser una cuestión de vida o muerte. En este contexto, es más obvio que las pruebas deben ser una parte integral del desarrollo de software. Los equipos deben verificar atributos de calidad como confiabilidad, estabilidad y tolerancia a ciertos cambios de hardware.

Enfoques ágiles

Estos productos obtienen los mismos beneficios de prácticas ágiles como TDD, programación en pares, integración continua (CI), un enfoque de calidad de todo el equipo y desarrollo guiado con ejemplos de cara al cliente como en el desarrollo basado en pruebas de aceptación (ATDD)/ especificación por ejemplo (SBE)/ desarrollo impulsado por el comportamiento (BDD).

La historia de Lisa

Aproveché la oportunidad para ayudar a probar la aplicación iOS de nuestro equipo. yo era un Yo mismo adopté muy tarde los teléfonos inteligentes y las tabletas, y no pude Incluso intuye los gestos correctos para la funcionalidad estándar de la pantalla táctil. Afortunadamente, los desarrolladores de iOS me pusieron rápidamente al día con los conceptos básicos de las pruebas. para dispositivos físicos, Xcode y Crashlytics. Practican TDD y emparejan programación para desarrollar nuestro código iOS, que es compatible con CI.

Probar una aplicación móvil como novato es una forma de encontrar errores y usabilidad asuntos. Descubrí que mi experiencia general en pruebas se aplica a las pruebas de iOS como Bueno. Como ocurre con cualquier producto de software, intentamos ofrecer las funciones más valiosas para nuestros usuarios y nuestro negocio. También tenemos que satisfacer App Store, que es un actor importante. Además, la aplicación móvil es compatible con nuestra producto web de software como servicio (SaaS), por lo que resulta útil estar familiarizado con cómo debería funcionar.

Sin embargo, a medida que comencé a aprender más de las sesiones de conferencias, artículos y libros de líderes de pruebas móviles como Julian Harty, Karen N. Johnson, y Jonathan Kohl, me di cuenta de que hay otro enorme nivel de complejidad en pruebas móviles. Tengo que aprender nuevas habilidades y nuevas formas de pensar para ser más eficaz.

A medida que aprenda más, podré contribuir más a nuestro equipo, que tiene Ya comenzamos a mejorar el diseño y la arquitectura de nuestra aplicación móvil. código. También están aprendiendo nuevas formas de automatizar funciones más funcionales y pruebas de regresión.

Les pedimos a los profesionales de pruebas ágiles con experiencia en aplicaciones móviles y desarrollo de software integrado que compartieran sus ideas sobre cómo realizar pruebas ágiles en estos dominios. Esperamos que sus historias le inspiren experimentos que pueda probar con sus equipos móviles o integrados.

Lecciones aprendidas con Agile en dominios móviles e integrados

jon agar , el autor de Ataques de prueba de software para romper Dispositivos móviles e integrados (Hagar , 2013), habla de equipos móviles e integrados que utilizan prácticas principios de prueba ágiles.

Érase una vez, los sistemas integrados se creaban típicamente utilizando Métodos de desarrollo graduales y cerrados, por lo que se observaron prácticas ágiles. como un gran cambio. Esta es una idea un poco errónea. Hubo temprano y adoptadores exitosos de ágil en el dominio integrado. El cohete Atlas El equipo de software integrado (Waters, 2004) adoptó algunos principios y prácticas ágiles con el tiempo. Por ejemplo, utilizaron CI con tamaños más pequeños, compilaciones de software más frecuentes; centrado más en el nivel de desarrollador pruebas con automatización (no del todo TDD pero cerca); permitió el sistema desarrollarse en el tiempo en lugar de seguir planes y especificaciones rígidos; corrió

pruebas automatizadas con enfoque exploratorio; y tenía participación directa del usuario/cliente.

He escuchado a gente decir que Agile es solo para software, no para proyectos centrados en hardware (como suelen serlo los dispositivos integrados y algunos dispositivos móviles).

Las historias de éxito proporcionan el contrapunto a esta generalización. Por ejemplo, Wikispeed (Wikispeed, 2014) es en gran medida un proyecto de hardware que sigue ideales ágiles.

Wikispeed utiliza Extreme Manufacturing, que implica diseño ágil, Scrum, optimizaciones de flujo kanban, prácticas de programación extrema y otros enfoques ágiles para convertir rápidamente ideas como cómodos automóviles de cercanías en productos que los clientes puedan usar.

Los esfuerzos paralelos de desarrollo de hardware y software significan que es probable que se produzcan lagunas en la funcionalidad y problemas de integración. Los conceptos de prueba ágiles con CI y la retroalimentación rápida ayudan a esto al proporcionar información a los desarrolladores. Por ejemplo, en los sistemas integrados, normalmente el hardware se desarrolla al mismo tiempo que se crea el software. Es un problema constante del "huevo y la gallina". El hardware define cómo se deben construir los aspectos del software, y el software debe estar implementado para evaluar muchas de las características que proporciona el hardware. En la CI de sistemas embebidos, es común construir un poco de hardware, integrarlo con algún software, cambiar el hardware en un diseño final y luego al final tener una definición más completa necesaria para terminar el software. La CI hace que esta evolución paralela sea eficiente.

También es común en el dominio integrado tener software de última hora.

cambios de software para adaptarse a los problemas de hardware. Escucho declaraciones como: "Bueno, el hardware está arreglado y es más fácil cambiar el software", lo cual puede ser cierto, pero conlleva el riesgo de que se introduzcan errores de último momento en el software.

Existe una situación similar en el mundo de las aplicaciones móviles, donde es posible que la aplicación deba funcionar con muchas versiones diferentes del sistema operativo y/o plataformas de hardware. Aquí, el problema de la CI surge en estas diferentes configuraciones y genera preocupaciones sobre las pruebas combinatorias y sobre "funcionará" para que una porción suficiente del mercado sea un éxito.

Basándome en historias como estas, estas son algunas de las lecciones que he aprendido al implementar la metodología ágil en dominios móviles e integrados:

- Muchos equipos, especialmente aquellos que desarrollan aplicaciones móviles, creen que el tiempo de comercialización anula la necesidad de realizar pruebas. Sin embargo, si observa las reseñas y calificaciones en línea, verá que tener muchos errores reduce las calificaciones y, por lo tanto, la probabilidad de que una aplicación tenga éxito. • Los cambios tardíos en el hardware afectan aún más al producto fuertemente que los cambios en el software, por lo que las pruebas deben ser muy ágiles

para acomodarlos. Cuando se desarrollan hardware y software,

Si se lleva a cabo en paralelo, la IC es especialmente importante. Los evaluadores deben esperar Cambios de última hora en el software para solucionar problemas de hardware. problemas. Se necesitan pruebas tanto automatizadas como exploratorias.

Busque patrones de errores en el código. Incluir pruebas de ataque para evitar que los errores escapen a producción.

- Comience a realizar pruebas lo antes posible. Utilice dobles de prueba como simulacros. objetos y talones de prueba para componentes que aún no se entregan. Los simuladores también pueden ayudar a dar una ventaja a las pruebas.
- La automatización de pruebas puede ser más difícil en dispositivos móviles e integrados. dominios. Aunque están cambiando, muchas herramientas de prueba más antiguas sí lo hacen. no funciona para dispositivos móviles e integrados debido a la interacción de hardware única. Debido a esto, los evaluadores en ágil Los equipos pueden necesitar muchas más habilidades técnicas además de las pruebas básicas. conocimiento. Por ejemplo, es posible que sea necesario comprender los conceptos de diseño de hardware, el código y las herramientas de desarrollo para realizar pruebas exitosas.

¿Reconoces a tu equipo en alguna de las lecciones aprendidas por Jon? Considera estas lecciones aprendidas con tanto esfuerzo para ayudar a su propio equipo a incorporar calidad en su producto de software integrado o móvil. Las prácticas y técnicas que Jon describe no son nuevas; es posible que simplemente se necesiten en diferentes momentos y de diferentes maneras que para probar aplicaciones web u otros produ Puede adaptar patrones y estrategias familiares para realizar pruebas integradas y móviles efectivas, como aprenderemos en la siguiente historia.

Estrategias para la automatización de pruebas en pruebas móviles

Jeff "Cheezy" Morgan , autor de Pepino y Queso (morgan , 2013), explica cómo prueba como él desarrolla móvil aplicaciones y se adapta a ciclos de liberación cortos.

Abordo las pruebas móviles casi de la misma manera que abordo las pruebas de un Aplicación web. Creo mis Objetos de Pantalla (Objetos de Página) para cada uno pantalla en la aplicación, definir los conjuntos de datos que usaré, desarrollar mis diferentes rutas de navegación, y luego construir las pruebas para aprovechar estos artefactos. Aunque la estructura del código es casi idéntica, Hay algunas cosas acerca de probar aplicaciones móviles que son completamente únicas.



Al probar aplicaciones web, debe compararlas con las distintas navegadores de destino y versiones de esos navegadores. Tu tienes lo mismo Lo mismo ocurre con las aplicaciones móviles, pero es más complicado. Tienes que preocúpese por diferentes dispositivos, así como por diferentes versiones de esos dispositivos. Además, debe probar su aplicación en diferentes versiones del sistema operativo en todos esos dispositivos diferentes.

La matriz de plataformas específicas puede crecer bastante rápidamente.

Para complicar aún más las cosas, existen diferentes tipos de aplicaciones móviles. Hay aplicaciones nativas, aplicaciones web, y aplicaciones llamadas híbridas, que son una combinación de ambas. Las herramientas o bibliotecas de codificación que utiliza para automatizar las pruebas pueden tener cambiar según el tipo de aplicación.

Finalmente, hay dos categorías principales de pruebas que deben realizarse en aplicaciones móviles: probar el comportamiento y probar el apariencia. La apariencia se divide aún más dependiendo de el tipo de aplicación. Si se trata de una aplicación nativa, en realidad estás probando la disposición de los controles que constituyen la pantalla. si es una web aplicación, está probando el diseño responsive de las páginas web como mostrado en la pantalla. Siempre construyo automatización para probar el comportamiento, pero rara vez para probar la apariencia. Encuentro que automatizar lo visual aspectos de la aplicación llevan mucho tiempo y las pruebas tienden a ser frágil, lo que provoca muchos fallos falsos en las pruebas. Con toda esta complejidad, Puede parecer una tarea abrumadora descubrir cómo gestionar adecuadamente Pruebe su aplicación. Esto es lo que hago.

Lo primero que debe hacer es comprender claramente los dispositivos de destino. No lo es Es difícil obtener informes de uso de dispositivos móviles en Internet. (Consulte la bibliografía de la Parte VII, “¿Cuál es su contexto?” para obtener enlaces a ejemplos de estos.) Utilizo estas listas para crear mi propia lista de dispositivos que planeamos objetivo. No olvides que debes convertirlo en una combinación de dispositivos y sistemas operativos. En algunos casos aparecerá el mismo dispositivo. más de una vez pero con diferentes versiones del sistema operativo.

Con esa lista lista, adquiero uno o dos de los cuatro dispositivos principales. Conecto uno de cada dispositivo directamente a mi servidor de compilación para que toda mi automatización se ejecute continuamente en los dispositivos cada vez que un desarrollador verifica el código en el sistema de gestión de código fuente. si solo tengo uno de cada dispositivo, también creo un emulador para esos dispositivos en el construir el servidor y configurar la compilación para verificar primero si hay un dispositivo y ejecutar las pruebas en el emulador si no existe. Realizo verificación visual y pruebas exploratorias contra cualquier dispositivo que no esté conectado al servidor de compilación.

Lo último que hago es crear varios emuladores para realizar pruebas en Dispositivos que no tengo. Coloco estos emuladores en el servidor de compilación.

y crear una compilación que se ejecute todas las noches y ejecute la automatización contra cada uno de estos emuladores. Además, creo tres emuladores en el computadora del probador que representan las pantallas grande, mediana y pequeña resoluciones de los dispositivos de destino. Estos emuladores en el probador La máquina se utiliza para la verificación visual de la aplicación.

Cheezy combina pruebas automatizadas y humanos para verificar el comportamiento y la apariencia de las aplicaciones móviles. Si aún no tiene el tipo de infraestructura que describe, como servidores de compilación y emuladores, su equipo puede abordar esto en conjunto, tal vez con la ayuda de DevOps (consulte el Capítulo 23, "Pruebas y DevOps", para obtener más información sobre la infraestructura de prueba). .



Según nuestra experiencia, aprovechar la automatización para hacer que las pruebas exploratorias sean más efectivas es una estrategia ganadora. La siguiente historia ilustra algunos aspectos de esto.

Pruebas exploratorias móviles asistidas por automatización

jon agar explica cómo las pruebas exploratorias de dispositivos móviles y El software integrado se benefician de la automatización.

Muchos profesionales piensan que las pruebas exploratorias son algo que debe ser realizado manualmente por un evaluador de pensamiento que planifique, diseñe, se ejecuta y luego aprende sobre el producto de forma interactiva en tiempo real. La automatización puede respaldar pruebas exploratorias de dispositivos móviles e integrados. sistemas sin restringir la libertad del evaluador para explorar diferentes avenidas. Por ejemplo, la automatización es útil para crear entradas de prueba, especialmente para pruebas combinatorias, llenado de bases de datos y técnicas de prueba de soporte, como análisis de valores límite y equivalencia. clases.

Las herramientas de captura y reproducción de datos pueden ayudar a proporcionar cierta repetibilidad durante una sesión de prueba exploratoria. Después de las pruebas, la automatización puede ayudar para analizar los resultados de las pruebas. Los scripts de prueba automatizados pueden ayudar con revisar archivos de registro grandes o escanear en busca de patrones particulares, incorrectos resultados o tendencias negativas. Informes de errores y otra información de prueba. se puede generar automáticamente, lo que proporciona buenas formas de visualizar los datos. Esta información se puede utilizar para análisis posteriores y manuales. exploración.

Por encima de todo, necesitamos evaluadores pensantes, pero la automatización de pruebas puede ayudar en ciertos contextos. Los siguientes ejemplos ilustran situaciones en las que la automatización fue invaluable para el esfuerzo de prueba exploratoria.

Mi primera historia se refiere a un sistema médico integrado. En este caso, el software bajo prueba podría representar un riesgo para la vida humana, por lo que fue necesaria una planificación de pruebas basada en riesgos con un análisis de herramientas de apoyo a lo largo de los ciclos de desarrollo del proyecto. Las pruebas se centraron primero en elementos de alto riesgo. Los evaluadores también estaban preocupados por el sesgo del evaluador, por lo que incluyeron el uso de herramientas de prueba combinatorias para proporcionar casos de entrada matemáticamente sólidos. Las entradas y salidas de prueba para el dispositivo de software bajo prueba se procesaron a una velocidad de 10 milisegundos. En consecuencia, se necesitaba un gran entorno de automatización de simulación de pruebas en tiempo real para proporcionar entradas cada 10 ms y registrar más de 1 MB de salidas de datos de prueba generadas durante cada secuencia de prueba. Las grandes cantidades de datos generados y registrados también necesitaban herramientas de procesamiento informático para localizar posibles errores, buscar tendencias y visualizar los resultados de los datos de las pruebas. Finalmente, los resultados se informaron en herramientas en línea para que los desarrolladores y los usuarios/clientes interesados pudieran revisar y analizar los resultados de las pruebas rápidamente y según su conveniencia. Cada una de estas áreas de automatización requirió su propio esfuerzo de desarrollo para crear las herramientas, pero estos esfuerzos adicionales se vieron compensados por mejores pruebas y un producto integrado de mayor calidad.

Mi segundo ejemplo se refiere a un juego de aplicación móvil que se está desarrollando por primera vez. En este caso, los desarrolladores querían poder repetir rápidamente las acciones que el evaluador había realizado para encontrar un problema. Los evaluadores utilizaron una herramienta de captura y reproducción que se ejecutaba tanto en un simulador como en dispositivos móviles reales para capturar la sesión exploratoria. Si se encontraba un error durante la iteración, la sesión capturada se enviaba electrónicamente a los programadores, para que pudieran reproducirla y respaldar sus esfuerzos de depuración. Si no se encontraban errores, el script de captura se conservaba sólo hasta el final de la siguiente iteración, ya que los cambios eran rápidos y los scripts quedaban obsoletos rápidamente. Sin embargo, a veces un script podría reutilizarse si los cambios en él fueran insignificantes. Además, los evaluadores utilizaron una herramienta de lista de verificación de juegos en línea (consulte la bibliografía de la Parte VII para ver los enlaces sugeridos) para ayudarlos a recordar qué buscar y probar durante una sesión de prueba exploratoria. Esta retroalimentación con una rápida ejecución de prueba semiautomática produjo la información que permitió producir un juego divertido y funcional. Después de que se lanzó el producto, los nuevos lanzamientos utilizaron formas más avanzadas de generación de datos de prueba e informes de casos de prueba para mejorar continuamente el juego. El tiempo inicial de comercialización fue rápido con una fase de prueba final muy corta y, posteriormente, el producto se probó continuamente para mejorarlo para los jugadores.

Consulte el Capítulo 12, “Pruebas exploratorias”, para conocer más formas de utilizar la automatización para facilitar la exploración. Reúna a los miembros del equipo de hardware y software para buscar oportunidades de automatización. Por ejemplo, utilice la automatización para registrar las condiciones del hardware y del sistema operativo en cuanto a tiempo, temperatura o voltaje mientras el probador invoca una serie de pasos del usuario de la funcionalidad del software.

Aprendizaje de pruebas ágiles para software móvil

JeanAnn Harrison, un probador de software de la Boston

en los Estados Unidos, explica cómo trabajar estrechamente con pro la ayudó a acelerar rápidamente.

Mi primera introducción a las pruebas de software móvil, que fue en el El dispositivo de multas de tráfico del Departamento de Policía de Los Ángeles también me dio exposición a la aplicación de una metodología ágil a un proyecto de software.

Antes de este proyecto, no recibía capacitación ni tutoría ágil formal. Eso La primera experiencia probando software móvil fue un aprendizaje inmenso. experiencia.

Oh, claro, había requisitos para comenzar, pero la mayoría de nuestros requisitos se escribieron mientras “tanteábamos” el diseño del software. I

Me di cuenta de lo estrechamente que necesitaba trabajar con el desarrollador, para poder para conocer la intención del diseño. Al mismo tiempo, estábamos probando ver si el diseño tenía sentido en términos del conjunto general de requisitos del cliente. Discutimos escenarios como el de los policías. necesidad de poder almacenar todos los datos de un día de trabajo en el dispositivo y luego cargar esos datos una vez finalizado su turno. Momento y el rendimiento se volvió fundamental para la forma en que se desarrollaron los requisitos del software móvil.

Las decisiones sobre el uso se convirtieron en discusiones de diseño basadas en pruebas exploratorias. Rápidamente me di cuenta de que con cada sprint necesitaba establecer algunos objetivos antes de la prueba. Planificar qué tipos de pruebas debían realizarse hecho dentro del alcance del proyecto y el cronograma hizo que mis pruebas tarea mucho más fácil.

La experiencia resultó ser un proyecto relativamente corto, pero éste es donde estuve expuesto por primera vez a iteraciones limitadas en el tiempo a pesar de que no le dió un nombre a nuestro proceso. Básicamente me enseñé a mí mismo lo que Trabajo en función de cómo interactué con las compilaciones y el programador. Hicimos lo que funcionó para nosotros, el proyecto y nuestros usuarios.



Asumir un nuevo dominio y proceso de desarrollo es más fácil cuando colabora con otros miembros del equipo de entrega. Los evaluadores de software móvil deben ir más allá del software para comprender cómo las condiciones del sistema operativo y del hardware afectan el comportamiento del software. La colaboración continua entre programadores, evaluadores, diseñadores de UX, administradores de sistemas y otros especialistas es vital para que los productos tengan éxito en el mercado ultracompetitivo actual.

Resumen

Los valores, principios y prácticas de las pruebas ágiles se aplican al desarrollo de software integrado y móvil, pero estos dominios presentan desafíos únicos. Se debe prestar especial atención a las pruebas de software integrado y dispositivos móviles propietarios que funcionan en situaciones críticas para la vida.

- Los dispositivos móviles y otros dispositivos con software integrado se utilizan de maneras que las computadoras tradicionales no lo hacen, y estas diferencias presentan desafíos únicos para las pruebas.
- El tamaño del dispositivo y las limitaciones de recursos requieren diferentes enfoques para el diseño de pruebas.
- Las pruebas pueden agregar aún más valor en dispositivos móviles e integrados sistemas debido a riesgos potencialmente altos desde la perspectiva tanto de marketing como de seguridad.
- Los enfoques ágiles funcionan bien para probar aplicaciones móviles y software integrado debido al rápido ciclo de retroalimentación, pero existen obstáculos únicos, como límites a las versiones beta y procesos lentos de aprobación de tiendas para dispositivos iOS.
- Es posible que se necesite cierta automatización para realizar pruebas exploratorias efectivas de software en dispositivos móviles y de otro tipo. Los equipos deben colaborar para ver qué herramientas pueden ayudar.
- Experimentar juntos para encontrar enfoques de prueba que sean apropiados para el producto, el dispositivo y los plazos de entrega.

Esta página se dejó en blanco intencionalmente.

Capítulo 21

Pruebas ágiles en regulados Ambientes

21. Pruebas ágiles en entornos regulados

El mito de la “falta de documentación”

Ágil y cumplimiento

Desde los albores de la Programación Extrema, algunos expertos nos han dicho que el desarrollo ágil no es apropiado para software altamente regulado y crítico para la seguridad, como dispositivos médicos y software para vuelos espaciales. Aparentemente muchos equipos no recibieron ese memorando. Como vio en el Capítulo 20, “Pruebas ágiles para sistemas móviles e integrados”, hemos escuchado muchas historias de éxito sobre equipos en dominios regulados como el aeroespacial y los dispositivos médicos que han adoptado valores y principios ágiles. Muchas empresas deben cumplir con regulaciones financieras como Sarbanes-Oxley (SOX). Estas empresas han adaptado el desarrollo ágil para satisfacer sus necesidades y han descubierto cómo pueden reducir el riesgo con prácticas ágiles.

El mito de la “falta de documentación”

Algunos artículos que explican por qué la metodología ágil no es apropiada para dominios regulados citan la falta de documentación. Seguimos escuchando este fundamento. malentendido mental incluso cuando el término desarrollo ágil está en su segunda década. Como mencionamos anteriormente, el desarrollo ágil consiste en entregar valor con frecuencia y a un ritmo sostenible. No existe ninguna razón inherente por la que el software desarrollado utilizando valores, principios y prácticas ágiles tenga una documentación inadecuada.

Cuando utilizamos un enfoque guiado por ejemplos para el desarrollo de software, convertimos los ejemplos en pruebas y podemos automatizar muchas de esas pruebas. Como han señalado David Evans y Gojko Adzic (Adzic, 2011), las pruebas automatizadas se convierten en documentación viva. La belleza de documentar el código

a través de pruebas automatizadas es que, dado que debe seguir pasando las pruebas de regresión, siempre mantiene su documentación actualizada. Dimos un ejemplo de esto en Agile Testing (p. 402). La documentación precisa es fundamental en ámbitos donde la vida, la seguridad o el dinero de las personas están en juego.

Ágil y cumplimiento

Las agencias reguladoras imponen reglas, y todos los que se rigen por esas reglas deben demostrar que las cumplen. Debemos realizar pruebas para asegurarnos de que nuestro software que prueba el cumplimiento produzca resultados precisos.

La historia de Lisa

En mis más de ocho años trabajando en un equipo que desarrolló una aplicación de servicios financieros, adquirí mucha experiencia verificando que nuestro software cumplía con las regulaciones federales, o que ayudaba a nuestros clientes a cumplir con esas regulaciones.

Nuestros clientes utilizaron nuestro software para administrar los planes de jubilación 401(k) de los empleados. Cada año, tenían que demostrar que operaban dentro de las reglas del IRS. Por ejemplo, los ejecutivos no podían aprovechar para llenar sus propias cuentas con ahorros libres de impuestos. Las reglas eran extremadamente complicadas y a menudo contrarias a la intuición, pero disfruto aprendiendo dominios comerciales desafiantes.

Nuestro propietario de producto nos dio muchos ejemplos de ejemplos de entradas y resultados esperados. Convertirlos en pruebas que guiaran la codificación aseguró nuestro éxito.

Entregamos sin miedo un software altamente complejo que automatizaba de manera confiable las pruebas de cumplimiento para nuestros clientes, por lo que no corrían riesgo de incurrir en multas.

Nuestros competidores no podían creer que pudiéramos hacer esto. ¡Pero teníamos las pruebas para demostrarlo!

Más tarde, nos enfrentamos a las “reglas de Bernie Madoff”, regulaciones para ayudar a evitar que los corredores estafen a los inversores. Se trataba de cosas como revelar las tarifas de mantenimiento de registros y evitar que los asesores del plan cambiaran la información personal de los inversores. El software para cumplir con estas reglas no generó ingresos, pero el incumplimiento podría significar multas importantes para nuestra empresa y nuestros clientes. Para cada requisito regulatorio, trabajamos con las partes interesadas del negocio para encontrar la solución de software más simple, minimizando costos pero garantizando el cumplimiento y la seguridad.

Las auditorías y las presentaciones regulatorias imponen responsabilidad y otros requisitos necesarios para obtener un sello de aprobación. Muchos de esos

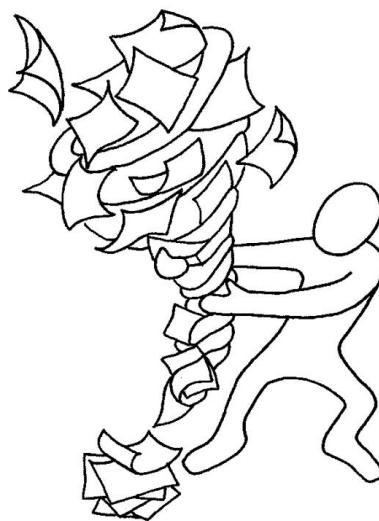


Figura 21-1 Luchando con la documentación

las regulaciones dicen “mostrar evidencia de que” o algo similar; Desafortunadamente, muchas organizaciones a menudo interpretan que esto significa “dame más documentación” y dedican su tiempo no a probar, sino a documentar (consulte la Figura 21-1).

La historia de Janet

Hace un tiempo, estaba en una empresa que quería comenzar a realizar un desarrollo más ágil y aún necesitaba cumplir con SOX. Observé los estándares de la empresa y vi que eran todo menos ágiles. El documento de estrategia de prueba tenía capa tras capa de documentación requerida, y las personas encargadas de escribirlo ni siquiera estaban seguras de quién lo estaba leyendo o por qué era necesario.

Los equipos ágiles luchaban por descubrir qué era absolutamente necesario para cumplir con su cumplimiento, pero no verse abrumados por documentación que tenía poco significado y no satisfacía sus necesidades.

Hay muchas empresas que pueden cumplir con las normas regulatorias sin documentación excesiva (ver Figura 21-2). Generalmente, siempre y cuando la empresa documente lo que hace, haga lo que documente y

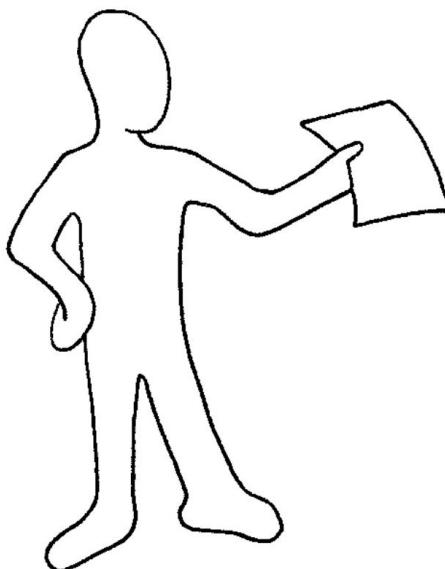


Figura 21-2 Simplifique su documentación.

puede producir evidencia de que está haciendo lo que dice que está haciendo, los reguladores estarán de acuerdo con eso. Vale la pena leer las regulaciones y luego adoptar un enfoque pragmático para cumplirlas. Por ejemplo, tomar fotografías del guión gráfico con regularidad y tenerlas disponibles para su revisión puede ser una forma perfectamente legítima de documentar lo sucedido con fines de cumplimiento.

Haga que sus auditores (son partes interesadas) formen parte de la solución y defina sus necesidades como historias o tareas para que los entregables se integren en la solución. Considere las tareas que deben completarse para cumplir con las regulaciones cuando su equipo cree una definición de "hecho" para la próxima versión. Los equipos necesitan disciplina para seguir su proceso de manera consistente y producir los artefactos requeridos. Las actividades de prueba contribuyen mucho a garantizar el cumplimiento. Como se mencionó anteriormente en este capítulo, las pruebas automatizadas son una excelente manera de crear resultados de prueba consistentes y documentación viva. El cumplimiento es parte de Release Done y, por lo tanto, parte del producto, no un examen que se debe aprobar al final.

A Janet le hacen una pregunta con bastante frecuencia: "¿Hay algún dominio que no debería intentar utilizar métodos ágiles, como el regulatorio?"

¿Es posible la agilidad en un entorno regulado?

griffin jones , un Agile, que ~~dominava~~ asesoramiento sobre el cumplimiento en los probadores, hizo esta pregunta ~~cada vez~~ que se pone en movimiento, Agile se da cuenta alguien está realizando pruebas en entornos regulatorios expertos.

Así es como responde Griffin.

Para mí la pregunta es personal. Una de mis funciones es representar a la organización ante autoridades externas que pueden no estar familiarizadas con Agile. Por lo general, comienzo reconociendo que estamos regulados, lo que eso significa para la organización y cómo hemos implementado el cumplimiento en el contexto de un proceso ágil de desarrollo y prueba de software. Es fundamental dejar claro el pensamiento del equipo sobre el tema.

Primero, el equipo tiene que reconocer que está creando simultáneamente dos cosas: el producto y la documentación reglamentaria.

En segundo lugar, los comportamientos, tareas y artefactos del proyecto deben abordar tanto las necesidades del producto como las necesidades regulatorias al mismo tiempo. Son parte del mismo flujo de acciones. No puedes completar uno y luego intentar atornillar el otro encima. Tienen que ser distintas caras de una misma moneda.

En tercer lugar, sea sensible al contexto y cree una flexibilidad razonable en los procesos para permitir un aprendizaje interactivo rápido. El control de procesos y procedimientos debe estar ";" o ~~centrado en la revisión~~. Intente minimizar la cantidad de trabajo adicional para crear los artefactos para contar y fundamentar la historia regulatoria del proyecto.

En cuarto lugar, si usted está haciendo honestamente un buen trabajo de ingeniería competente, ¿cómo puede capturar de forma natural evidencia para respaldar la historia regulatoria, sin interrumpir el flujo del trabajo y crear ineficiencia e ineffectuación?

Para mí se trata de asumir la responsabilidad y preparar el proyecto para sobrevivir a la revisión y crítica de una autoridad externa. A lo largo de cada jornada laboral hago tres preguntas:

- ¿Estarían felices las partes interesadas si vieran y escucharan lo que estamos haciendo ahora?
- ¿Es importante que el proyecto pueda compartir esta información? con las partes interesadas en el futuro?
- ¿Cuál sería la forma más eficaz y eficiente de conmemorar esto para poder acceder a él y compartirlo en el futuro, sin interrumpir el flujo?

Al centrarse en estos puntos y preguntas clave, el proyecto puede realizar un desarrollo de software ágil que cumpla con las normativas. Los "cómo" específicos del proyecto se convierten simplemente en detalles específicos del contexto.



Qué gran mensaje: haga que sus requisitos regulatorios formen parte de su trabajo, pensando constantemente en cómo hacerlos lo más simples y efectivos posible.

Las pruebas en entornos regulados son una oportunidad para desarrollar nuestras habilidades en forma de T colaborando con programadores y otros miembros del equipo, como verá en la siguiente historia.

Colaboración para probar aplicaciones de dispositivos médicos

En el Capítulo 20, "Pruebas ágiles para dispositivos móviles y software", JeanAnn Harrison integrado", relató su primera experiencia en probando software móvil y trabajando en un equipo que desarrolla ambos entornos ágiles. Posteriormente se unió y nuevamente encontró beneficios al colaborar con todo el equipo, especialmente cuando se trataba de satisfacer necesidades regulatorias.

Fui contratado por una empresa de dispositivos médicos para ayudar a reforzar el equipo de pruebas actual, brindar apoyo a ese equipo y brindar tutoría en el ciclo de vida del desarrollo de software. El dominio de los dispositivos médicos fue una experiencia completamente nueva, pero mi experiencia probando software de dispositivos móviles fue útil.

Cuando comencé a verificar las pruebas escritas previamente por un grupo subcontratado, quedó claro que las pruebas eran solo pruebas funcionales, con poca consideración por el rendimiento, el estrés, las condiciones límite y las pruebas de usabilidad.

Las condiciones del hardware y del sistema operativo no se consideraron en los requisitos, lo que significaba que no teníamos la cobertura de prueba mínima para un dispositivo médico. Nuestros usuarios no solo eran nuestros clientes, sino que también eran pacientes que llevaban un monitor para proporcionar datos cardíacos a su médico.

La diferencia entre el software de nuestro dispositivo médico y una tableta o aplicación de teléfono era cuántas aplicaciones más pequeñas se necesitaban para que un sistema de extremo a extremo llegara a su fin. Fue un proyecto de prueba muy complejo que me ayudó a prepararme para proyectos de prueba de aplicaciones móviles individuales y arrojó luz sobre la comprensión de cómo las condiciones del hardware y los sistemas operativos se integran con la aplicación móvil. Considerar las pruebas más allá de la interfaz de usuario funcional (UI) es vital para planificar cualquier proyecto de pruebas móviles.

Necesitaba aprender más sobre cómo se creaban, procesaban y luego transmitían los archivos, así como el diseño del código de alto nivel. Necesitaba aprender la arquitectura de todo el sistema para comprender mejor cómo interactuaba cada aplicación con otra. Las interdependencias eran muy complejas, lo que hacía que los casos de prueba fueran más complejos. Para poder proporcionar una cobertura de prueba sólida, necesitaba pasar tiempo con el

equipo de desarrollo y hacer muchas preguntas, participar en revisiones de código y revisar archivos de registro para comprender mejor dónde existían esas interdependencias.

Al pasar tiempo con el equipo de desarrollo, descubrí que nuestro sentido del humor compartido nos ayudó a unirnos. Mi equipo de desarrollo se sintió cómodo trabajando conmigo y estuvo dispuesto a mostrarme lo que querían que aprendiera de ellos. Todos reconocimos lo valiosas que podrían ser mis observaciones para su diseño. Uno de los mayores problemas del desarrollo de software móvil es la falta de requisitos. No sabes lo que no sabes. Los puntos de referencia en particular tienden a ser extraordinariamente vagos. Las pruebas exploratorias pueden ayudar al proporcionar información a los desarrolladores para diseñar con mayor precisión. Los evaluadores pueden aprovechar la oportunidad para aprender más sobre la arquitectura cuando participan directamente en el proceso de diseño.

Por ejemplo, mientras realizaba pruebas de estrés, reconocí que podía, en un corto período de tiempo, ver cómo reaccionaría el software del dispositivo a la carga agregada, y descubrí un error importante de larga data. Trabajando juntos, el programador y yo descubrimos que el error ocurría con más frecuencia de lo que hubiéramos creído posible. El error debía solucionarse de inmediato, por lo que el programador y yo nos asociamos para resolver el problema. Al final, aprendí no sólo más sobre el diseño del software, sino también cómo ser más preciso en mis pruebas. En las pruebas móviles, trabajar directamente con el equipo de desarrollo ayuda al evaluador a adquirir conocimientos sobre la arquitectura, lo que conduce a una gama más amplia de cobertura de pruebas más allá de las pruebas funcionales y de interfaz de usuario.

En el caso de dispositivos médicos como el monitor cardíaco móvil, el equipo de entrega debe proporcionar documentación que pueda relacionarse directamente con los requisitos. Mucha gente siente que esta trazabilidad significa hacer coincidir un requisito específico directamente con un caso de prueba específico.

Sin embargo, no todos los requisitos fueron escritos o enviados para ser probados directamente. En cambio, el equipo diseñó casos de uso para demostrarle al auditor que el dispositivo y el software funcionaban como se esperaba. Estos casos de uso proporcionaron una pista para diseñar documentos donde podría desarrollar más pruebas de casos de uso. El proceso fue menos intrusivo para el programador y los evaluadores presentaron la documentación de prueba necesaria para satisfacer a las partes interesadas regulatorias.

Probar un dispositivo médico móvil tiene sus desafíos, especialmente cuando el dispositivo es propietario. Sin embargo, todos los evaluadores de dispositivos móviles deben trabajar estrechamente con su equipo de desarrollo para aprender todo lo que puedan sobre la arquitectura. El tiempo, las diversas condiciones del hardware, las dependencias del firmware y la secuencia de ejecución del código son componentes críticos al diseñar casos de prueba. Trabajar con todo el equipo de desarrollo hizo que nuestro proyecto de software ágil en el monitor cardíaco móvil fuera bastante exitoso.

El equipo de JeanAnn encontró una solución ligera para la trazabilidad de los requisitos que era aceptable para sus auditores regulatorios. Existe la percepción de que regulado siempre significa peso pesado, pero según nuestra experiencia, los auditores suelen estar abiertos a alternativas simples para cumplir con sus requisitos de información. Experimente y colabore para encontrar una solución óptima.

Resumen

Muchos aspectos del desarrollo ágil, como los ciclos cortos de retroalimentación y las pruebas de regresión automatizadas para proporcionar documentación viva, hacen que las prácticas ágiles sean adecuadas, incluso ventajosas, en entornos regulados.

Reunir a los expertos en la materia con los evaluadores y otros miembros del equipo de desarrollo nos permite guiar la codificación y mitigar el riesgo con una amplia gama de ejemplos. Los entornos regulados brindan oportunidades únicas para los evaluadores que disfrutan dominar las reglas comerciales y ayudar al equipo a lograr lo correcto. Algunos de los aspectos de los enfoques ágiles para las pruebas en entornos regulatorios que cubrimos en este capítulo son:

- Contrariamente a la creencia popular, ágil no significa falta de documentación. Los equipos que utilizan ejemplos para guiar el desarrollo los convierten en pruebas automatizadas que proporcionan documentación viva que puede ayudar a cumplir los requisitos reglamentarios.
- Cumplir con los requisitos de información de los auditores no significa necesariamente producir documentos pesados y de la vieja escuela. Trabaje con los auditores para encontrar formas ligeras de cumplir que se ajusten a las necesidades de su equipo.
- Las regulaciones gubernamentales agregan características de calidad que deben verificarse, incluso si no agregan valor comercial directamente. Evitar multas y garantizar la seguridad del cliente son fundamentales para el negocio.
- La colaboración entre probadores y programadores puede ser esencial para probar productos de software regulados y de alto riesgo. Comprender el diseño del sistema, las consideraciones de hardware y los problemas de sincronización ayuda a prevenir defectos o encontrarlos temprano.

Capítulo 22

Pruebas ágiles de datos Almacenes y Negocios Sistemas de Inteligencia

22. Pruebas ágiles para almacenes de datos
y Sistemas de Inteligencia de Negocios

¿Qué tiene de especial las pruebas de BI/DW?

Usando principios ágiles

Datos: el activo crítico

Grandes datos

Un almacén de datos (DW) es un entorno integrado que contiene datos operativos y de apoyo a las decisiones. También almacena datos que se crean dentro del entorno de almacenamiento de datos, como agregados, resúmenes y cálculos. Es una forma eficaz de gestionar la función de apoyo a las decisiones en una organización al tratar los datos como un activo estratégico que puede utilizarse para obtener una ventaja competitiva. Un buen sistema proporciona fácil acceso a datos relevantes, lo que permite el modelado y el análisis retrospectivo y predictivo para informar mejor las decisiones comerciales.

de ahí la idea de inteligencia empresarial (BI). Modelar la empresa para un almacén de datos supone un esfuerzo enorme y, en los enfoques tradicionales de desarrollo de almacenes de datos que requieren que se defina primero el modelo de datos completo, la empresa tarda años en ver algún beneficio.

Con métodos ágiles, utilizando ciclos de retroalimentación rápidos, los empresarios pueden priorizar qué información quieren ver primero. Por lo general, tienen poca o ninguna idea de lo que quieren hasta que lo ven, por lo que, una vez más, el ciclo rápido de retroalimentación les resulta beneficioso. La idea de trabajar de forma incremental significa que los problemas de calidad de los datos quedan expuestos tempranamente, evitando problemas masivos más adelante.

¿Qué tiene de especial las pruebas de BI/DW?

Bajo el paraguas de BI, usted utiliza el sistema como un consumidor de datos, más bien como un rol de usuario final. Para DW, usted se adentra en la entrega técnica y necesita comprender la solución arquitectónica.

La Figura 22-1 es un diagrama de ejemplo de una solución BI/DW.

Un gran desafío es encontrar evaluadores con las habilidades necesarias: conocimiento técnico para verificar el movimiento de datos desde el origen al destino, conocimiento técnico de bases de datos, habilidades de prueba en integridad de datos para garantizar que los datos sean consistentes, conocimiento de pruebas de seguridad y habilidades analíticas. para comprender lo que representan los datos. ¿Es siquiera razonable pensar que estas habilidades se pueden encontrar en una sola persona? Empieza a sonar de nuevo como ese “enfoque de todo el equipo”, ¿no es así? Todo el equipo necesita abarcar aún más disciplinas. Es importante que el desarrollo de aplicaciones considere el almacén de datos, porque los problemas de calidad de los datos pueden dificultar las pruebas y la finalización de las historias.

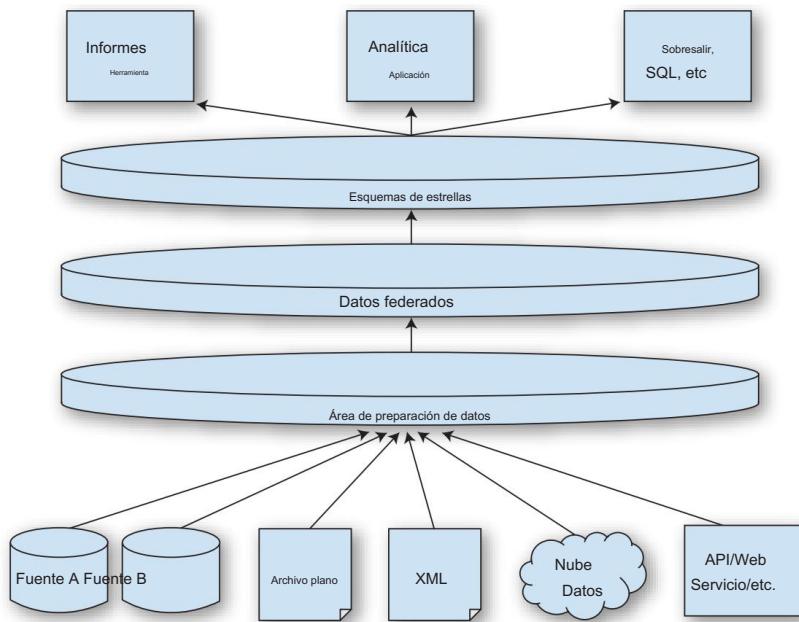


Figura 22-1 Ejemplo de configuración de un almacén de datos (proporcionado por Mike Enrique)

Los equipos implementan prácticas ágiles de manera diferente con funciones de inteligencia empresarial y almacén de datos de la forma en que lo hacen en una aplicación web normal. Por ejemplo, es posible que no haya nada visible para la demostración a través de una interfaz de usuario durante mucho tiempo. El valor está en los datos, que pueden demostrar mejor desde el principio utilizando herramientas de línea de comandos. El almacenamiento de datos se basa en un modelo por lotes, donde la funcionalidad consiste en mover grandes volúmenes de datos en lotes, en lugar de procesar transacciones continuamente, por lo que la integración continua (CI) no es tan significativa. Esto significa que es posible que las pruebas de regresión no se ejecuten como parte de una compilación diaria, sino como parte de los procesos de carga operativa diaria. Sin embargo, si se crean pruebas automatizadas con datos de prueba relacionados para cada historia, muchas de las pruebas se pueden ejecutar en un entorno de CI.

Tradicionalmente, los equipos de DW/BI han utilizado grandes volúmenes de datos para probar el código. A menudo, cargar todos esos datos y ver los resultados de las pruebas puede llevar muchas horas o días. Los equipos ágiles están encontrando formas de reducir el volumen de datos necesarios para probar historias adecuadamente o aprovechar nuevas tecnologías que aceleran significativamente los procesos de extracción, transformación y carga (ETL) para permitir pruebas en grandes volúmenes de datos. Las historias de nuestros colaboradores en este capítulo le mostrarán algunas de las formas en que las personas se han adaptado.

Las pruebas unitarias para DW/BI se ven diferentes de las pruebas unitarias de aplicaciones. Una prueba unitaria típica podría garantizar que los recuentos de filas y los totales coincidan desde el origen hasta el destino, y estos tipos de pruebas ETL se pueden automatizar. Sin embargo, el paso de transformación cambia la forma en que se representan los datos. Por ejemplo, se pueden convertir tipos de datos, aplicar cálculos o agregar valores diarios en totales mensuales. Este tipo de transformaciones representan algunos de los riesgos en la gestión de datos que a menudo dificultan las pruebas unitarias.

Para probar DW/BI, es esencial una excelente comprensión de los valores de límites, rangos y transformaciones de tipos de datos. Es fácil caer en la trampa de tener en ejecución muchas pruebas automatizadas que son tan simplistas que no agregan ningún valor. Las pruebas simples se pueden generar o copiar fácilmente ya que las entradas generalmente se realizan a través de una interfaz de línea de comandos. La dificultad está en comprender las diferentes reglas de decisión y caminos que pueden tomar los datos y ejecutarlas. Los problemas en la manipulación y transformación de datos a menudo ocurren en los límites entre tipos. Para crear pruebas válidas e informativas es necesario comprender la naturaleza de los tipos de datos dentro de los sistemas de bases de datos y las reglas de decisión que se aplican a los datos a medida que se procesan.

Tabla 22-1 Lista de verificación simple para la automatización

Descripción de la prueba	Riesgo	Objetivo
1 Verifique que no se pierdan registros	Alto	Los registros se transfieren por completo.
2 Verifique que no se pierdan campos	Alto	Los registros se transfieren por completo.
3 Verificar que los totales de origen y destino coincidan	Alto	Los registros se transfieren por completo.
4 Verifique que no haya errores en el tamaño de los datos	Medio	Los registros se procesan correctamente
5 Verificar que no haya errores de tipo de datos	Bajo	Los registros se procesan correctamente
6 Verifique que los cambios de formato de datos sean realizados correctamente	Bajo	Los registros se procesan correctamente

Colaborar con los programadores y propietarios de productos permite a los evaluadores orientar las pruebas de manera efectiva. Una combinación de comunicación y exploración puede ayudar a identificar las pruebas adecuadas para automatizar y reducir los riesgos de pruebas ineficaces.

La automatización es necesaria, pero tiene una apariencia diferente a la que analizamos en la Parte VI, "Automatización de pruebas". A menudo se presenta en forma de scripts simples que comparan campos. Una lista de verificación simple puede ayudar a identificar algunos de los problemas y resaltar lo que se puede automatizar para probar aplicaciones de almacén de datos. Un colega de Janet empezó con una lista de verificación sencilla, similar a la tabla 22-1.

Es posible que deba aplicar formato o enmascaramiento de datos para realizar una verificación consistente, almacenando el estado original de los datos en cada etapa de la prueba para ayudar a rastrear dónde se ha producido cualquier problema. Además, capturar todos los registros y rastros del proceso ayudará a diagnosticar cualquier problema más adelante.

Intente utilizar un proceso automatizado para ejecutar pruebas, crear e inyectar datos y luego examinar los resultados y utilizar pruebas exploratorias para investigar más a fondo. Los equipos de Adam Knight utilizan arneses automatizados para ejecutar sus pruebas exploratorias, ya que de esa manera es más fácil manipular los datos. Tenga en cuenta que, debido a la gran cantidad de datos involucrados, es posible que necesite monitorear la actividad en producción para garantizar que las condiciones del borde se manejen correctamente.



Usando principios ágiles

Probar diferentes enfoques de prueba de forma iterativa ayuda a los equipos a descubrir buenas formas de demostrar la calidad de los datos y el valor que pueden agregar al negocio. Los principios ágiles básicos, como la simplicidad, la retroalimentación rápida y la división de elementos en pequeños fragmentos, se pueden aplicar a los sistemas BI/DW.

Aprender a probar BI

mike heinrich
almacén de datos
tardó en coger ritmo.

acciones para probar BI
en un
Prueba pionero, pero no le hizo falta

Durante el proceso de construcción de nuestro almacén de datos y entorno de inteligencia empresarial, aprendimos muy rápidamente que el negocio era No está interesado en los pasos intermedios necesarios para preparar y transformar los datos a través de múltiples bases de datos. La afirmación: "Pero puedo hacer Esto en Excel mucho más rápido" era un estribillo común. Con eso en mente, Se hizo muy evidente que necesitábamos demostrar que el negocio era utilizable, información modelada temprana y frecuentemente. Por supuesto, para organizar todos los datos, federarlo y transformarlo en algo que la empresa pueda evaluar a través de procesos ETL tradicionales es costoso, requiere mucho tiempo y aversión al cambio.

Esto nos llevó a encontrar formas de abstraer los modelos lógicos de datos y considerar lo que le importaba a la empresa, en lugar de concentrarse en el medio de almacenamiento. Usar nuestras herramientas de informes para realizar tareas rudimentarias La federación y la transformación de datos acortaron el ciclo de retroalimentación y permitió a los empresarios utilizar una herramienta con la que estaban familiarizados para refinar sus requisitos de datos. Al darle a nuestro negocio algo tangible para explorar, redujimos la cantidad de problemas de calidad de los datos y Incluso ayudó a refinar de dónde deberían obtenerse los datos.

A medida que se comprendieron mejor los requisitos de los datos, el equipo de desarrollo pudo determinar si ETL y la persistencia de los datos eran necesarias o si la solución existente era satisfactoria. Mi papel como evaluador a lo largo de todo esto se volvió menos sobre verificar la funcionalidad de scripts ETL y más sobre cómo intermediar la comunicación entre la empresa y el equipo de desarrollo en torno a la calidad de los datos y las expectativas de rendimiento.

A medida que avanzamos hacia las pruebas de aceptación del usuario (UAT), la iterativa Este enfoque fue una vez más necesario ya que aprendimos que el concepto de una "UAT con sello de goma", donde el equipo de pruebas ya había descubierto

y comunicado todos los problemas, sería completamente infructuoso y garantizar que nunca llegaríamos a producción. La UAT se convirtió en proceso intensivo para el cual necesitábamos una comunicación aún más frecuente entre la empresa y el equipo de entrega cuando finalmente trabajaron a través de su modelo general completo. Aceptando eso que Habíamos llamado que UAT era en realidad solo la iteración final de "desarrollo y prueba". del producto fue quizás el mayor obstáculo mental que tuvimos que superar.

El concepto de desarrollo impulsado por modelos, o de poblar una zona vagamente Un modelo lógico definido (y cambiante) para permitir que la empresa comprenda los datos antes no era necesariamente un enfoque popular. Sin embargo, los ciclos de retroalimentación acortados permitieron que un proyecto que había costado millones y languideció en desarrollo durante años para finalmente entregar información valiosa sobre los clientes y los costos para respaldar la rentabilidad de la organización.

Mike utilizó muchos principios ágiles para aprender a probar su primer proyecto BI/ DW. Al comprender lo que el cliente consideraba valioso, el equipo pudo adaptar su enfoque para satisfacer esas necesidades. Los rápidos ciclos de retroalimentación le dieron al equipo la confianza para experimentar y ver qué funcionaba mejor. Janet tuvo la suerte de hablar con Mike durante el proyecto y uno de los muchos problemas que tuvieron fue la mala información. El equipo sufría un viejo problema: GIGO (basura que entra, basura que sale). Con los rápidos ciclos de retroalimentación, pudieron comenzar a limpiar los datos a medida que desarrollaban, en lugar de tener que esperar hasta que todo estuviera construido.

Para garantizar un desarrollo iterativo fluido, identifique los requisitos de su entorno de prueba al inicio de cada actividad para que no se bloquee esperando el hardware y los entornos donde pueda ejecutar sus pruebas.

Datos: el activo crítico

Los almacenes de datos generalmente se rigen por reglas estrictas. Un ejemplo es que los datos no pueden ser volátiles: nunca deben sobrescribirse ni eliminarse. Una vez que se confirman las transacciones de la base de datos, los datos son estáticos, de solo lectura y se retienen solo para informes futuros. Esto puede implicar enormes cantidades de datos únicos con un ciclo de vida de valor limitado, razón por la que hoy en día escuchamos tanto sobre Big Data. Probar con tantos datos se convierte en un problema en sí mismo, y lo cubriremos más adelante en este capítulo.

Otro tema es la privacidad y la seguridad. No siempre es posible utilizar datos de producción para realizar pruebas. La mayoría de los países tienen una regulación de privacidad que protege los datos personales. Esto significa que los datos de producción deben ser "borrados" o "despersonalizados". El administrador de la base de datos del equipo anterior de Lisa creó un procedimiento utilizando claves con un cálculo matemático que reemplazó información confidencial como las identificaciones dentro del sistema. Si fuera necesario, los ID podrían descifrarse para identificar los valores en producción. Si un enfoque unidireccional es aceptable, simplemente puede utilizar números generados aleatoriamente. Algunos productos de bases de datos tienen procedimientos integrados que se pueden utilizar. Asegúrese de que los valores sustituidos cumplan con las mismas restricciones a las que están sujetos los datos de producción.

En inteligencia de negocios, los datos son el activo más importante. Dado que las decisiones comerciales se basan en esos datos y en cómo se representan, la forma en que manejamos los datos para las pruebas es fundamental.

Resolver el problema de los datos de prueba incorrectos

Lynn Winterboer, un formador y consultor ágil para BI/DW soluciones, cuenta cómo su equipo resolvió el problema que enfrentaron al probar datos apropiados.

Todos los que trabajan en el almacenamiento de datos y la inteligencia empresarial.

La industria comprende cómo la calidad de los datos puede afectar los resultados de BI. Esto es cierto tanto para pruebas como para cargas de producción y puede ser particularmente doloroso cuando las reglas de negocio que se aplican en el ETL son complejas y superposición. Un equipo de BI puede ahorrarse muchos dolores de cabeza si empieza con un conjunto limpio de registros de pruebas específicos para pruebas unitarias y de regresión para los principales escenarios del proyecto.

Por ejemplo, trabajé con una empresa que había crecido más de 20 años, en gran medida mediante adquisiciones. Cuando se adquirieron empresas, sus sistemas por personas no familiarizadas con la gestión de datos. eran solo preocupado por lograr que los datos encajen de alguna manera en el sistema operativo objetivo para que los negocios combinados puedan continuar operando. Lamentablemente, esto provocó muchas anomalías en los datos del sistema ERP.

En un proyecto de asignación de ingresos que requería datos de pedidos de este sistema ERP, nuestro equipo de DW/BI trabajó para implementar reglas comerciales complicadas. Para probar los ETL que aplicaron estas reglas, utilizamos aleatoria

muestras de datos de producción. En repetidas ocasiones experimentamos problemas debido a la calidad de nuestros datos de prueba. Los problemas se veían así:

1. Codifique y pruebe el Producto A.
2. Codifique y pruebe el Producto B.
3. Prueba de regresión del Producto A con el mismo registro que utilizamos para probar la primera vez, pero esta vez la prueba falla.
4. Al trabajar con el líder empresarial del proyecto, eventualmente Descubrimos que el registro que habíamos usado para el Producto A tenía un anomalía en el mismo que provocó un falso negativo en la prueba de regresión cuando las reglas para el Producto B también estaban en la ETL. en una limpieza registro para el Producto A, la prueba habría pasado.

Debido a estos problemas, tuvimos dificultades para codificar y probar con éxito. nuestros escenarios de camino feliz, y mucho menos los casos de esquina. el equipo estaba frustrado.

La líder empresarial decidió dedicar su tiempo a crear un conjunto pequeño y preciso de datos de prueba que pudiera usarse para confirmar que teníamos todos nuestros datos.

Lógica de codificación correcta para los principales escenarios en nuevos pedidos. Identificó 16 órdenes limpias que podríamos usar para realizar pruebas y pruebas de regresión. el ETL a medida que se agregó cada escenario y luego calcular manualmente el resultados de asignación de ingresos adecuados para cada uno.

Esto permitió a nuestro equipo volver a centrarse en los resultados más importantes. Podíamos confiar en que cuando una prueba fallaba, se debía a un problema de código real. dentro de un escenario específico relacionado con el registro que no pasó la prueba, y No son malos datos de prueba. Esto ahorró mucho tiempo perdido. Nuestro liderazgo empresarial Ahora tenía tiempo para investigar las anomalías de los datos y decidir si fijarlos en la fuente, proporcionar reglas de negocio para abordarlos, o determinar si deben incluirse en la carga inicial. Ella También pudo aclarar los estándares de entrada de datos en el sistema fuente. en el futuro, para que podamos evitar este problema con el próximo inevitable adquisición o integración de datos.



¿Puedes identificar algunas de las prácticas ágiles que Lynn utilizó en su historia? Guiar el desarrollo con ejemplos es lo primero que notamos. No es el formato normal en el que pensamos, pero la práctica es la misma. Proporcione ejemplos reales y luego codifique para que las pruebas pasen. La segunda práctica que nos llamó la atención fue la creación de datos de prueba por parte del líder empresarial en función de los escenarios más importantes. Utilizaron pequeños experimentos con retroalimentación rápida para evolucionar su enfoque de prueba y codificación.

Gestión de datos para pruebas

En esta siguiente historia, Jeff "Cheezy" Morgan comparte que encontró datos útiles cuando llegó tarde. intento y para automatizar popu-prueba

Hace unos años estaba trabajando con un equipo que estaba desarrollando un sistema de datos. Aplicación de almacenamiento mediante ETL. La fuente era una base de datos muy grande con aproximadamente 140 tablas y el objetivo era un esquema en estrella de gran tamaño. base de datos (Wikipedia, 2014n).

Cuando llegué por primera vez, los evaluadores estaban probando la aplicación ejecutando alrededor de 500.000 registros en el sistema cada noche. Cuando ellos vino a trabajar a la mañana siguiente, realizaron consultas contra la fuente para encontrar registros que coincidieran con la condición que deseaban probar. Una vez encontraron un registro apropiado, buscaron el mismo registro en el objetivo e interrogó los datos para asegurarse de que la regla comercial se activó y que modificó correctamente los datos. No hace falta decir que, Este proceso fue extremadamente laborioso y propenso a errores.

De inmediato quise automatizar el proceso, pero para comenzar la automatización necesitaba una forma de configurar los datos en la fuente. Él Fue muy intimidante pensar en hacer que cada prueba insertara datos en tantas mesas. La complejidad iba a ser enorme, así que comencé buscando una manera de simplificar la gestión de datos de prueba.

Pronto descubrí que cada regla de negocio que realiza el proceso ETL examinó sólo un pequeño subconjunto de los datos generales que pasaban a través de él para determinar si debe activar la regla o no. Me di cuenta de que para cada prueba, simplemente podría completar las tablas con valores predeterminados. Entonces para En cada prueba específica, solo pude reemplazar los pocos valores que cambiaron el resultado de una prueba específica con los valores que quería para esa prueba. Esto simplificó todo dramáticamente. Me tomó cuatro días crear un marco que proporcionaba esta capacidad. Después de eso, cada evaluador pudo proporcionar valores específicos para un pequeño subconjunto de datos necesarios para verificar un caso de prueba particular y dejar que la prueba use los valores predeterminados para todo demás. La Figura 22-2 muestra esta configuración.

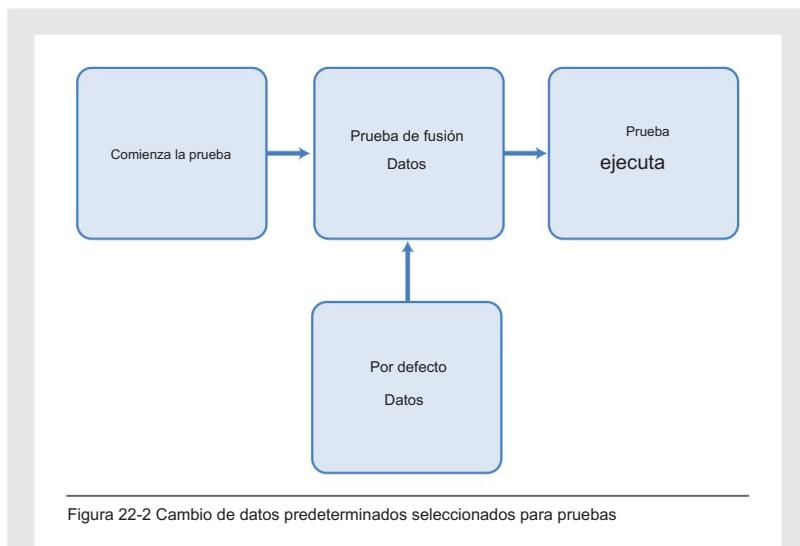


Figura 22-2 Cambio de datos predeterminados seleccionados para pruebas

El siguiente proyecto en el que trabajé fue una aplicación web. Pronto descubrí que se aplicaba el mismo patrón. Se necesitaban muchos datos para cada flujo de trabajo de prueba, pero la cantidad de datos que realmente cambiaron el resultado de las pruebas fue un pequeño subconjunto. Al especificar sólo el subconjunto de datos y permitiendo que un marco llenara el resto, pude simplificar las pruebas.

El descubrimiento final que hice fue cuando comencé a ejecutar las pruebas en paralelo para lograr que terminen en una cantidad más corta de tiempo. Cuando haces esto, pierdes control sobre el orden en el que tus pruebas ejecutadas. Si dos pruebas que se ejecutan al mismo tiempo intentan utilizar el mismo dato, es probable que cada uno modifique los datos que el otro necesita, y una o ambas pruebas fallarán. Para evitar este tipo de falla, usted Puede utilizar la automatización para aleatorizar los datos utilizados por las pruebas, de modo que Cada prueba tiene sus propios valores de entrada únicos y se pueden realizar varias pruebas de forma segura. ejecutarse simultáneamente.

La gestión de datos puede resultar dolorosa. Escribir scripts para crear conjuntos de datos de forma iterativa puede ser preferible a extraer y borrar grandes cantidades de datos de una base de datos de producción. Un enfoque consiste en construir de forma iterativa un generador de datos. Comience con registros que prueben las reglas más simples y luego agregue soporte para diferentes reglas a medida que las identifique. Por ejemplo,

Los valores, los registros huérfanos y los registros de alta cardinalidad pueden proporcionar pruebas útiles para la integridad de las transformaciones de datos y los informes. Crear un generador en lugar de crear registros manualmente le permite a su equipo escalar sus datos una vez que esté satisfecho con su cobertura funcional.

Grandes datos

Big Data es un desafío para muchas organizaciones. Los conceptos de volumen, velocidad y variedad son realmente los que definen Big Data, y es posible que los enfoques tradicionales de almacenamiento de datos no se escale lo suficiente para realizar pruebas. Cuando el rendimiento y el volumen de datos son consideraciones de alta prioridad, los equipos ágiles necesitan soluciones de prueba creativas.

Pruebas de rendimiento y escala en Agile

En las **iteraciones de prueba del** **Productos que operan en los mayores desafíos** **grande escala de datos,** Adán Caballero **para las pruebas.**

Proyectos ágiles: garantizar que el software cumpla con los requisitos de rendimiento iteraciones y escala en el corto plazo. de rendimiento disponibles para pruebas.

Uno de los mayores desafíos al probar un data warehouse o data El sistema de procesamiento garantiza que el sistema pueda satisfacer las demandas de rendimiento y escalabilidad que se le imponen. Rendimiento y La escala son características de calidad importantes de cualquier desarrollo de aplicaciones. Sin embargo, son una prioridad particular para los sistemas que gestionan grandes cantidades de datos.

Esto presenta una especie de desafío para los equipos ágiles. Cuando se trabaja en equipos pequeños, probando suficientemente los objetivos de rendimiento dentro del Las iteraciones de sprints cortos pueden parecer una tarea desalentadora. En producción En estos entornos, pueden ser necesarios muchos días o semanas para acumular el volumen de datos que constituye la capacidad operativa de los sistemas Big Data. Rara vez tenemos el lujo de estos plazos dentro de los sprints ágiles.

Escala en Velocidad

En el pasado, un enfoque de fuerza bruta para realizar pruebas a escala puede haber sido una opción como parte de una fase no funcional de un proyecto de prueba por etapas. En En un contexto ágil, debemos ser creativos al probar estos importantes características a la escala requerida y a la velocidad necesaria para ágil

desarrollo. A lo largo de los años de abordar exactamente este tipo de problemas, mi equipo ha ideado una serie de enfoques para ayudarnos:

- Capas de escalamiento: para procesar grandes volúmenes de datos en paralelo, los grandes sistemas de datos generalmente incorporan capas de escalabilidad de metadatos dentro de ellos. Estos suelen tomar la forma de índices de metadatos.

o bases de datos. Si podemos comprender y controlar estas capas, podemos generar rápidamente instalaciones que exhiban las mismas características que los sistemas de producción en términos de procesamiento de metadatos, pero que requieran una fracción del tiempo y espacio para generarlas. Trabajar con los programadores para comprender las estructuras de metadatos de la aplicación y cómo manipularlas es un enfoque excelente para ayudar a probar la escalabilidad de los distintos componentes del sistema de forma aislada. • Copia de seguridad y restauración: Haciendo uso de nuestra propia copia de seguridad y

mecanismos de restauración, podemos generar rápidamente entornos en los que podemos realizar pruebas, al mismo tiempo que creamos confianza adicional en nuestras capacidades de recuperación ante desastres.

- Instalaciones estáticas: un enfoque eficaz es mantener una serie de entornos poblados con datos a escala de producción con los que podemos programar comprobaciones de objetivos clave de rendimiento. En el enfoque más simple, mantenemos un sistema estático donde ejercitamos nuevas versiones de nuestros motores de consulta. En pruebas más avanzadas, mantenemos una prueba continua para realizar ciclos continuos de datos dentro y fuera de un sistema completamente poblado. De esta manera, probamos el rendimiento de las funciones a una escala realista sin tener que crear los datos desde cero para cada ejecución de las pruebas.

Automatización para Escala

Además de los enfoques para administrar el software y los datos, existen una serie de técnicas dentro de nuestros arneses de automatización que nos ayudan específicamente a alcanzar los objetivos de escalabilidad y rendimiento de las pruebas:

- Iteración: si diseñamos nuestros paquetes de prueba cuidadosamente, nuestros arneses de automatización respaldan la capacidad de ejecutarlos de forma iterativa. Esto nos permite escalar hasta grandes volúmenes de cargas de datos y realizar un seguimiento del rendimiento de otras operaciones del sistema. Podemos utilizar la automatización iterativa en pruebas exploratorias para permitirnos examinar características con actividad creciente, o como parte de pruebas continuas, como ejecutar pruebas iterativas en instalaciones estáticas como se mencionó anteriormente.
- Paralelización: la mayoría de los sistemas de procesamiento de datos admiten el procesamiento paralelo en múltiples núcleos de CPU o máquinas para

gestionar los volúmenes de datos involucrados. parece inevitable que nuestra automatización de pruebas también debe ejecutarse en paralelo si queremos ejercer la aplicación de manera realista. La capacidad de ejecutar Las pruebas en paralelo permiten a los evaluadores manejar cargas de trabajo mixtas realistas. que ejercitan de manera determinista los múltiples procesos que esperaría ver al ejecutar una aplicación en clúster. Es sensato tanto parallelizar la actividad dentro de una sola ejecución de prueba como para Parallelizar ejecuciones de pruebas automatizadas en diferentes máquinas. Esto permite diferentes cargas de trabajo y tipos de pruebas que se ejecutarán en diferentes entornos, optimizando la información que obtenemos dentro una iteración. Pruebas nocturnas más pequeñas para las pruebas más importantes, combinado con pruebas de fin de semana a gran escala en docenas de servidores, nos permiten desarrollarnos rápidamente y con confianza. Nosotros También son capaces de mantener conjuntos de pruebas a gran escala que brindan confianza en características de mayor escala.

- Monitoreo: Además de los resultados requeridos para proporcionar comprobaciones, capturamos información sobre el software en ejecución y su entorno mientras se ejecutan las pruebas. A medida que ejecutamos pruebas de forma iterativa y en paralelo, utilizamos la información recopilada para graficar características del sistema y modelar el comportamiento del sistema para identificar posibles problemas de escalabilidad.

Cuando se trabaja de manera ágil, es fácil pasar por alto las pruebas de rendimiento y escalabilidad del sistema. Sin embargo, estas son cualidades esenciales. Características de las aplicaciones de almacenamiento y análisis de datos. Adoptando Una filosofía de hacer que el rendimiento sea parte integral de nuestras pruebas y uso. algunas ideas creativas tanto en la exploración como en la verificación de enfoques, puede lograr una sorprendente cantidad de confianza en estas características, incluso en iteraciones cortas.

Big Data es un desafío nuevo y creciente para muchos equipos y requiere un enfoque diligente y multifacético como el que describe Adam. Se requieren pruebas de rendimiento en el almacenamiento de datos para establecer si la aplicación o el sistema es capaz de manejar la ingesta de datos y la carga de trabajo de consultas. Si, por ejemplo, el hardware o el software de la base de datos no son capaces de procesar la carga de trabajo, los intentos de ajustar el rendimiento no ayudarán. Sin embargo, si el hardware y la base de datos pueden adaptarse a la escala del trabajo, el equipo puede tomar medidas para ajustar el rendimiento, medir los resultados y volver a ejecutarlos para evaluarlos. Las pruebas pueden ayudar a identificar escenarios clave, pero hacer que los evaluadores trabajen en conjunto con los programadores de manera iterativa vale la pena para garantizar la calidad de los datos.

Resumen

Las pruebas en un almacén de datos o un sistema de inteligencia empresarial son un buen ejemplo de un contexto que amplía los límites de la agilidad para un equipo de desarrollo. En este capítulo compartimos ideas que esperamos ayuden a los equipos en dificultades.

- Considere las habilidades especializadas que su equipo necesitará para probar un almacén de datos o un sistema de inteligencia empresarial y llenar los vacíos con capacitación o incorporando personas con esas habilidades profundas.
- Se necesitan habilidades técnicas profundas, junto con un amplio conocimiento del dominio empresarial, para tener éxito en las pruebas de sistemas BI y DW.

- Principios y valores ágiles, como realizar pruebas tempranas y utilizar soluciones rápidas. Los comentarios se aplican al software que genera datos para ayudar con las decisiones comerciales.
- Para enfrentar el desafío de verificar el rendimiento y la escalabilidad cuando se manejan grandes cantidades de datos, busque formas creativas de utilizar metadatos, parallelizar el procesamiento de pruebas automatizado y trabajar en capas.

- Monitorear los datos del sistema en prueba y producción para evaluar la escalabilidad y verificar la respuesta del sistema a los casos extremos.
- Recuerde las preocupaciones de privacidad y seguridad al probar DW y Sistemas de BI.
- La integridad de los datos es fundamental para la inteligencia empresarial. Si los datos son malo, las decisiones basadas en esos datos serán malas. Asegúrese de que los datos de la prueba reflejen datos reales.
- Tenga en cuenta que es necesario abordar la ampliación y el rendimiento desde el principio. Cree su estrategia de prueba para escalar con Big Data.

Capítulo 23

Pruebas y DevOps

	Una breve introducción a DevOps
23. Pruebas y DevOps	DevOps y calidad
	Cómo los evaluadores agregan valor de DevOps

Hemos trabajado durante muchos años con miembros del equipo que realizan actividades que ahora se denominan DevOps. Como dice Jez Humble, los miembros del equipo involucrados en DevOps construyen “una plataforma que permite a los desarrolladores autoservicio de entornos para pruebas y producción (e implementaciones en esos entornos)” (Humble, 2012). Proporcionan herramientas que permiten a los equipos de entrega crear, probar, implementar y ejecutar sus sistemas. Según nuestra experiencia, todos los miembros de equipos ágiles realizan actividades de DevOps, pero uno o más miembros del equipo aportan habilidades especializadas profundas. Darle una etiqueta al conjunto de habilidades ha llamado la atención sobre su importancia para incorporar calidad al software. DevOps es un componente integral de las pruebas ágiles exitosas.

Una breve introducción a DevOps

El término DevOps se popularizó por primera vez en la conferencia DevOps Days celebrada en Bélgica en 2009. Desde entonces, ha habido algunas publicaciones excelentes dedicadas a DevOps y cómo encaja en el desarrollo ágil. Continuous Delivery (Humble y Farley, 2010) y DevOps for Developers (Hüttermann, 2012) son dos buenas guías para DevOps.

DevOps incluye prácticas y patrones que mejoran la colaboración entre diferentes roles en los equipos de entrega, simplificando el proceso de entrega de software de alta calidad. Estas prácticas y patrones ayudan a los equipos a escribir código comprobable y mantenable, empaquetar el producto, implementarlo y respaldar el código implementado.

Un área de interés para DevOps es acortar los tiempos de ciclo. Algunas empresas pueden llevar esto al extremo y entregar código nuevo a producción.

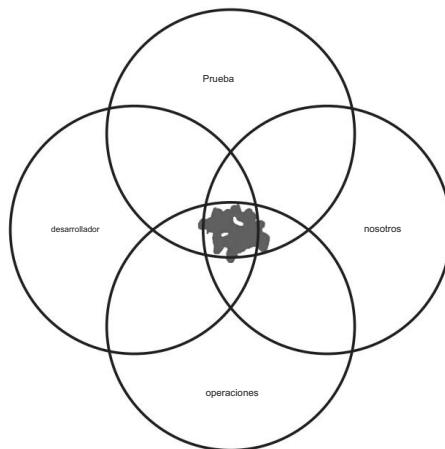


Figura 23-1 Intersección de desarrollo, pruebas, operaciones y el negocio: ¡el enfoque de todo el equipo!

muchas veces al día. Los miembros del equipo de entrega encuentran formas confiables y repetibles de pasar de una idea al valor comercial entregado, con un tiempo de ciclo que no solo es corto, sino también seguro. Esto ayuda a prevenir defectos y mejorar la calidad.

Las actividades de DevOps están diseñadas para mejorar la capacidad de mantenimiento y la velocidad de las pruebas y la implementación automatizadas. Los ejemplos incluyen mantener entornos de prueba, proporcionar retroalimentación rápida de la integración continua (CI) y garantizar confiabilidad y frecuencia (quizás incluso continua) despliegue. Muchos equipos ahora prueban su infraestructura continuamente para protegerse contra fallas espurias e intermitentes en las pruebas de regresión debido a fallas del servidor, la implementación o la configuración. Las partes interesadas del negocio también participan, ayudando a contratar a las personas adecuadas, obtener el hardware y software necesarios y recopilar información para guiar el desarrollo futuro. La figura 23-1 ilustra que DevOps es la intersección de la programación, las pruebas, las operaciones y el negocio. Es otra forma de ver el enfoque de todo el equipo para las pruebas ágiles.

DevOps y calidad

Las habilidades de los miembros del equipo ágil a menudo incluyen secuencias de comandos, administración de sistemas, codificación, configuración de CI, herramientas y habilidades de colaboración y comunicación que les permiten realizar actividades de DevOps. Estas actividades son fundamentales para realizar pruebas ágiles y desarrollar continuamente calidad en nuestros productos.

La historia de Janet

Hace años, antes de que existiera la idea de integración continua (al menos en mi mundo), trabajé con una persona que considero uno de los primeros DevOps practicantes. El papel de Darcy incluía la implementación de nuestro producto en el cliente. sitios, soporte de primer nivel para los clientes, mantener nuestros servidores en funcionamiento, así como ser nuestro evaluador técnico, configurar nuestros entornos de prueba y mantenerlos corriendo. Supo aportar lo que sabía del mundo de los clientes. en nuestras pruebas, incluidas las pruebas de recuperación. No me di cuenta del valor que tiene. aportado al desarrollo de nuestro producto hasta que trabajé en una organización donde ese rol cruzado no existía.

En otra organización teníamos un equipo de prueba de infraestructura. Ellos trabajaron con los programadores y los equipos que soportaron el hardware y comunicaciones. Trabajaron con cada uno de los equipos ágiles para ayudarlos. comprender los impactos de los cambios en la infraestructura en el nuevo características.

Personas con habilidades especializadas en operaciones y administración de sistemas ayudan al equipo a mejorar la calidad en muchos frentes diferentes. Pueden ayudar a configurar entornos de desarrollo, prueba y preparación adecuados, optimizar la CI y perfeccionar los procesos de implementación para estos entornos. También pueden ayudar a encontrar o crear e implementar marcos de automatización y otras herramientas que se adapten a las necesidades del equipo. Según nuestra experiencia, son excelentes para diagnosticar problemas y tienen muchas herramientas útiles a su disposición. Pueden ayudar con la infraestructura para respaldar pruebas automatizadas, como la generación de datos de prueba.

La Figura 23-2 ilustra la idea de que DevOps cruza los cuatro cuadrantes de pruebas ágiles. Ayuda a guiar el desarrollo con muchos tipos diferentes de actividades de prueba, proporciona herramientas y entornos para evaluar el producto y crea la tecnología que permite probar diversos atributos de calidad, como rendimiento, confiabilidad y seguridad.

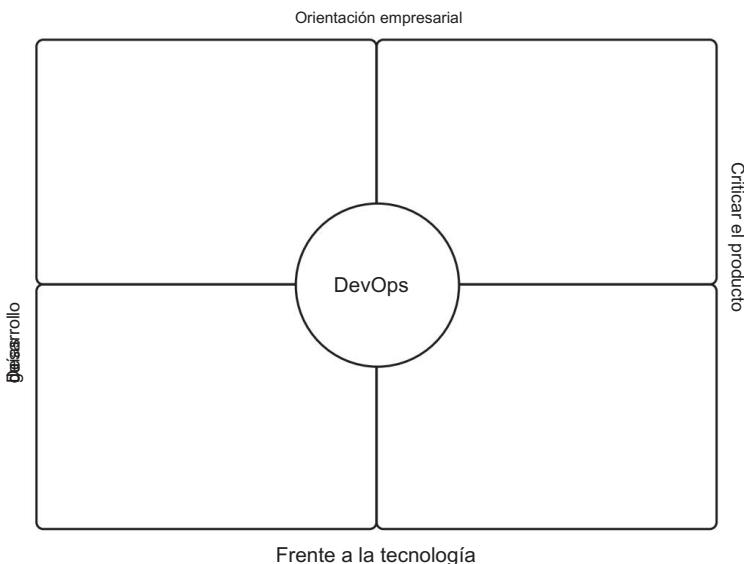


Figura 23-2 Las actividades de prueba se cruzan en DevOps.

La historia de Lisa

En los días de mainframe, cuando trabajaba como programador/analista, colaboraba estrechamente con los operadores de la sala de máquinas. Juntos aprendimos formas de resolver y prevenir problemas con el procesamiento de trabajos por lotes.

A principios de la década de 1990 tuve la suerte de aprender habilidades de administración de sistemas UNIX. de expertos de mi equipo. Aprendí la importancia de la liberación repetible. procesos y formas de verificar el producto envasado. En la década de 2000, mis compañeros de equipo con profundos conocimientos de administración de sistemas me ayudaron a encontrar mejores herramientas de prueba. marcos de automatización y formas de configurar trabajos de CI para obtener comentarios de prueba más rápidos. Me mostraron cómo usar archivos de registro y monitoreo para ayudar con depurar y explorar.

Mi equipo actual es el primero con el que he trabajado que utiliza el término . Nosotros DevOps Tenemos una acumulación de historias de usuarios de DevOps para mejorar todos los aspectos de nuestra infraestructura, incluidos nuestros entornos de prueba, CI y proceso de implementación. Los miembros del equipo interesados se unen a una reunión semanal de DevOps para priorizar el trabajo, y estas actividades se planifican junto con las características del producto. Los programadores trabajan con la empresa que aloja nuestro sitio de producción y mantiene nuestro software de motor de búsqueda. Hacemos nuestras propias implementaciones de producción y podemos Responder rápidamente a los problemas operativos. Experimentamos con formas de mejorar continuamente nuestro proceso de entrega.

Algunas empresas mantienen operaciones o departamentos de TI separados. Los miembros de estos silos a menudo tienen que apoyar a varios equipos, haciendo malabarismos con las necesidades de desarrollo y el mantenimiento de la producción. Es posible que pierdan contacto con los equipos de desarrollo y sus necesidades.



Una vez, Lisa facilitó un taller para una gran empresa donde los participantes de equipos que trabajaban en diferentes áreas de productos identificaron problemas que impedían varios aspectos de sus pruebas. Los evaluadores se quejaron de que las otras partes del sistema tenían incompatibilidades con las suyas y no tenían idea de cómo solucionarlas. Uno de los participantes del taller era del departamento de operaciones. Una vez que vio los obstáculos enumerados, dijo: "No tenía idea de que estabas luchando con esto."

Mi equipo puede encargarse de estos problemas. ¡De ahora en adelante, contáctame directamente! A veces, resolver un problema de prueba espinoso solo requiere reunir a personas de diferentes departamentos en la misma sala.

Ver el conjunto: agregar infraestructura al alcance de las pruebas, Estilo DevOps

Michael Hüttermann, autor de *DevOps para desarrolladores* (Hüttermann, 2012), comparte la historia que tuvo con un éxito Enfoque de equipo completo para respaldar la infraestructura y las pruebas. a través de DevOps.

En un proyecto más grande con unos 100 desarrolladores, tuvimos que hacer frente a objetivos agresivos de tiempo de comercialización, protegiendo las ventajas competitivas, mucha complejidad técnica, así como altas exigencias de disponibilidad y capacidad. Estos fueron los principales impulsores para implementar DevOps acercarse. Este enfoque incluía aplicar "la infraestructura como paradigma de código", que comienza con poner los manifiestos de Puppet en versión control, en nuestro caso Git. [Consulte la sección "Herramientas" del bibliografía para enlaces a todas las herramientas mencionadas en este capítulo.]

Se había implementado un proceso de entrega de extremo a extremo que abarcaba todos los departamentos, comenzando con la etapa 0, el espacio de trabajo del desarrollador, y cerrando con máquinas de prueba superiores, un espejo de producción y producción. Desde el Sistema de control de versiones, las líneas base se crearon continuamente. Aquellos Las líneas de base contenían diferentes unidades de configuración que componían una lanzamiento, incluido el código comercial, pruebas unitarias, pruebas de integración e información de infraestructura. Sobre la base de estas líneas de base, creamos liberar candidatos continuamente. Candidatos de lanzamiento cuidadosamente seleccionados

fueron promovidos para ser lanzados a producción. Esas versiones eran a la vez aptas para su propósito (al contener el alcance funcional) y aptas para su uso (al cumplir con los requisitos no funcionales).

Después de aplicar los conceptos de DevOps, los procesos y herramientas tanto de desarrollo como de operaciones se alinearon entre sí. Cada uno utilizó los mismos enfoques para aprovisionar máquinas, incluida la instalación y configuración de la infraestructura, el middleware y la aplicación empresarial. El desarrollo y las operaciones colaboraron continuamente para garantizar el máximo conocimiento y el intercambio abierto de información. Tuvimos tiempo libre para aprender y experimentar, y generamos respeto mutuo. Usamos kanban para gestionar el flujo y revisamos el diseño al principio del proceso. Esto permitió un aprendizaje temprano y frecuente, y un “fallo rápido”.

Al aplicar prácticas de DevOps, los requisitos de nivel de servicio y las capacidades del servicio podrían definirse y verificarse en las primeras etapas del proceso.

Las máquinas de desarrollo y las máquinas de prueba eran similares a las de producción, por lo que brindaban comentarios rápidos y significativos sobre la instalación y configuración, así como requisitos no funcionales como seguridad y monitoreo.

Como parte de la plataforma de entrega continua, Puppet se utilizó junto con Vagrant y Jenkins para configurar y eliminar máquinas virtuales para realizar pruebas reproducibles del proceso de aprovisionamiento en sí, así como del resultado definido en la máquina de destino.

Para encontrar errores de manera temprana y frecuente, comenzamos verificando categorías de fallas más básicas. Como parte de una construcción continua, el código de infraestructura en forma de manifiestos de Puppet se validó al principio del proceso. Cuando se encontraron inmediatamente errores sintácticos en los manifiestos, el proceso se canceló y no se produjeron archivos binarios para su uso posterior. Encontrar esta categoría de errores es fácil de lograr simplemente aplicando una validación del analizador de marionetas como parte de un paso de compilación dedicado en la compilación continua.

Después de la configuración real del entorno de prueba de destino con Puppet, se agregó otra etapa posterior para verificar el aprovisionamiento correcto. Introdujimos manifiestos de prueba que fueron aplicados por `puppet apply --noop` para verificar errores de compilación de Puppet y luego verificamos el registro de eventos resultante. Una prueba de humo automatizada básica comparó los resultados reales y deseados.

Cuando el proceso de entrega estuvo lo suficientemente maduro, agregamos verificaciones de estilo de código. Nos resultó útil acordar un formato compartido para todos nuestros artefactos. Se realizaron muchas comprobaciones del código comercial con SonarQube, pero también verificamos el cumplimiento de la guía de estilo del código de infraestructura con Puppet-lint.

La aplicación de prácticas de DevOps nos enseñó que era crucial fomentar la espíritu de "un solo equipo" formado por miembros de desarrollo, codificadores, probadores, negocios y operaciones, junto con redes, sistemas, e ingenieros de bases de datos. Todos los expertos involucrados se convirtieron en desarrolladores de la solución.

El equipo combinado compartió objetivos comerciales como reducir el tiempo del ciclo. Procesos compartidos, como usar el mismo concepto de aprovisionamiento para Todas las máquinas y herramientas compartidas, como Puppet, también alinearon al equipo combinado. Muchos incidentes de producción son causados por cambios en el Infraestructura de TI o por trabajo no planificado, como la producción de extinción de incendios. Incidentes debido a procesos rotos o soluciones mal probadas. Marioneta nos permitió definir especificaciones ejecutables de la infraestructura de destino comportamiento, haciendo que la infraestructura sea comprobable. Esto hizo que la automatización confiable, lo que acortó los ciclos de retroalimentación. La documentación sobre las máquinas están siempre actualizadas, facilitando las conversaciones entre desarrolladores y personal de operaciones.

Era elemental distinguir entre rendición de cuentas y responsabilidad. Los colegas del "equipo único" de desarrolladores y operaciones se reunieron responsable, pero sólo una persona del departamento de desarrollo era responsable de las máquinas de desarrollo y una persona del El departamento de operaciones era responsable de las máquinas de operaciones. Otros colegas que no eran responsables ni debían rendir cuentas fueron consultados o informados, temprana y frecuentemente.

En general fue un gran éxito para enfatizar todo el flujo de trabajo. desde el inicio (inicio) hasta el final (operación), extender las prácticas de prueba de desarrollo ágil a las operaciones e incluir intensivamente al personal de operaciones comenzando con las primeras fases del desarrollo de software.

Para lograr la automatización de pruebas confiable y los ciclos de retroalimentación más cortos que disfrutó el equipo de Michael, es necesario comprender su proceso de construcción. Sepa qué pruebas se ejecutan en qué entorno. El concepto de canalizaciones de construcción funciona bien para hablar de entornos de prueba. Crear buenos entornos de prueba ha sido un obstáculo para muchos de los evaluadores y equipos con los que ha trabajado Janet. En el Capítulo 5, "Conciencia técnica", la Figura 5-2 era un ejemplo de un proceso de construcción muy simple. En términos generales, hay un envío automatizado al entorno de desarrollo si todas las pruebas de CI pasan. Sin embargo, una de las prácticas que recomendamos es utilizar un sistema pull para el entorno de prueba. Esto les da a los evaluadores control sobre la ejecución de sus pruebas. En nuestra opinión, no hay nada peor que estar a mitad de camino de un ciclo de prueba exploratorio, sólo para tener

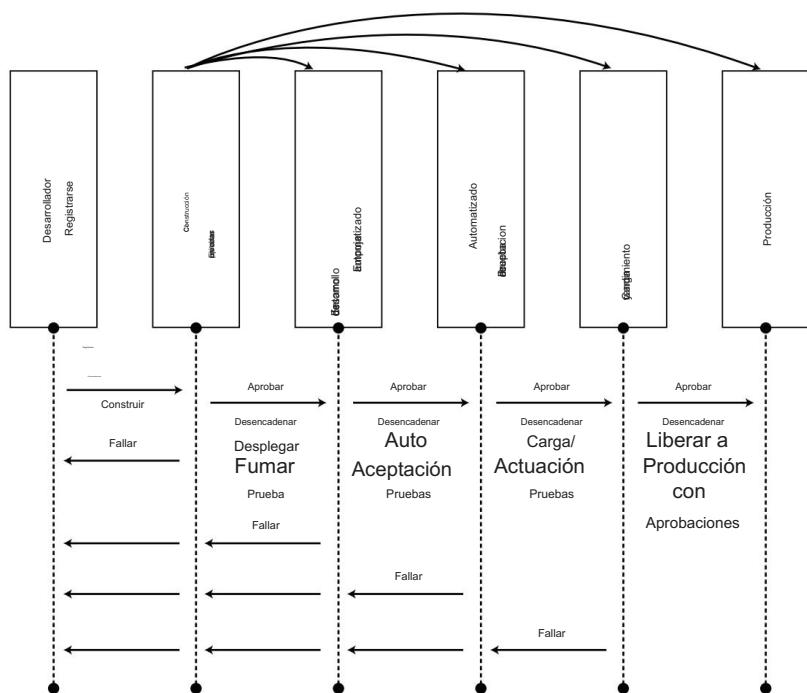


Figura 23-3 Canalización de compilación automatizada

estabas probando sobreescrito. Esto a menudo significa comenzar las pruebas desde cero.

La Figura 23-3 es un diagrama de un proceso de construcción automatizada. Si lo compara con el proceso de compilación simple de la Figura 5-2, verá que los entornos de prueba y de preparación de este ejemplo no son parte de la automatización. Esos son dos entornos en los que se realizarían pruebas exploratorias.

En el Capítulo 18, “Pruebas ágiles en la empresa”, presentamos la estrategia de prueba de Dell para su solución empresarial. La siguiente historia brinda más detalles sobre la infraestructura necesaria.

Pruebas de verificación de compilación automatizadas

Kareem Fazal, a ingeniero de software Dell, habla sobre la verificación de construcción y cómo la adaptaron después de sus primeros intentos.

El Desafío de matriz de pruebas de hardware

Uno de los desafíos que enfrentamos en prácticamente todos nuestros entornos de prueba de software en Dell Enterprise Solutions Group (ESG) es la amplia matriz de compatibilidad de hardware que se espera que admita nuestro software. Esto es más evidente en nuestro firmware Server Systems Management y proyectos de software que respaldan nuestros servidores de duodécima generación. Cuando Primero aplicamos prácticas ágiles a nuestros servidores 12G, establecimos una CI y entorno de prueba de verificación de compilación automatizada (aBVT). Este CI y El entorno aBVT apoyó un proyecto con 15 equipos Scrum distribuidos en India y Estados Unidos. Desde que se verificó el nuevo código varias veces por hora, había una demanda constante de frecuentes, Construcciones "bien conocidas". La compilación y las pruebas automatizadas debían finalizar en 90 minutos, ejecutándose en múltiples unidades físicas bajo prueba. (UUT) que se configuraron con combinaciones específicas de tarjeta de interfaz de red (NIC) y RAID.

Los procedimientos que utilizamos inicialmente para aBVT no eran muy flexibles. Él Fue difícil agregar conjuntos de pruebas que no siguieran un formato de interfaz predefinido al conjunto existente. Además, dado que los casos de prueba estaban obligados a servidores, si se requería una configuración específica que no estuviera ya en lugar, fue necesario desconectar un servidor y reconfigurarlo. Esto significaba que teníamos que tener mucho cuidado con las pruebas que se ejecutaban y cómo Cuánto tiempo tomó un escenario de prueba para equilibrar el uso de recursos con requisitos de prueba. En estas condiciones, era más fácil simplemente reducir el número de pruebas ejecutadas y limitar cada ejecución de prueba solo a aquellas casos de prueba que coincidían con la configuración de hardware existente. Para empeorar las cosas, este era un entorno de un solo subproceso, por lo que aunque había múltiples solicitudes de compilación entrantes cada hora, estaban bloqueados detrás de la única verificación de compilación en progreso

prueba de funcionamiento.

El Solución

En preparación para la próxima actualización, el equipo de CI y aBVT tomó medidas para abordar las limitaciones del entorno de prueba. El equipo de construcción imaginó una solución que proporcionara

- Equipos Scrum con “facilidad de uso” para desarrollar y enviar pruebas casos para lograr la máxima cobertura de hardware
- Formar equipos con “uso eficiente de los recursos de hardware” sin Requiere intervención manual entre ejecuciones de prueba.
- Equipos Scrum con la “flexibilidad” para priorizar casos de prueba en configuraciones específicas

En respuesta, nuestro equipo desarrolló el administrador de recursos genérico (GRM). GRM logró la flexibilidad y facilidad de uso que buscábamos al tener todas las configuraciones definidas por un conjunto simple de archivos XML independientes. El espacio de nombres se definió de una manera que no imponía limitaciones arbitrarias sobre la información que podía definirse, y al mismo tiempo se mantenía básicamente simple y fácil de entender. La forma en que se utilizó la configuración también permitió una buena escalabilidad.

El evaluador podía definir tanta o tan poca información para cada escenario de prueba como fuera necesario con muy poco esfuerzo superfluo.

GRM logró un uso eficiente de los recursos del servidor asegurándose de que se usaran en paralelo tanto como fuera posible y asignó recursos según los requisitos de la prueba. Si una prueba tarda más de lo esperado o termina con un error, el GRM se adapta a las condiciones actuales. Los requisitos de recursos que se definen para una prueba particular pueden ser tan amplios o tan limitados como sea necesario sin tener un efecto en otras pruebas en la ejecución de la prueba.

El GRM se puede utilizar para configuraciones de pruebas pequeñas, pero también puede escalar y admitir escenarios complejos con muchos tipos de recursos y miles de pruebas en la combinación. Esto se logra con una metodología de configuración sólida y fácil de entender que utiliza un formato de archivo XML estándar.

Resultados

Con la implementación del GRM en el entorno aBVT, los equipos pudieron maximizar la cobertura de la matriz de prueba para cada ejecución de un conjunto de pruebas.

Debido a que los casos de prueba para una ejecución particular no siempre requerían o consumían todas las configuraciones disponibles, se iniciaron ejecuciones adicionales de aBVT en paralelo para aprovechar al máximo las configuraciones disponibles. GRM brindó al equipo de construcción la flexibilidad de ampliar y agregar configuraciones y casos de prueba sin deshabilitar ninguna ejecución de prueba en progreso, lo que alivió las preocupaciones de mantenimiento.

Como puede verse en la historia de Kareem sobre sus experiencias, no existe una "talla única". Dell ESG creó a medida una solución flexible que funcionó para equipos de todo el mundo. Observe sus propios entornos y esfuércese por simplificar el ciclo de construcción, implementación y prueba para que sea consistente y mantenable.

Cómo los evaluadores agregan DevOpsValue

Un área de DevOps donde los evaluadores pueden aportar sus habilidades especializadas es ayudar a probar la infraestructura. Pueden ayudar a identificar lugares donde la infraestructura podría fallar simplemente haciendo preguntas como: "¿Cómo probamos qué sucede si la red falla?" Es posible que los evaluadores no sepan todas las respuestas, pero son buenos haciendo preguntas.

Janet trabajó con una empresa cuyo grupo de pruebas de infraestructura trabajó en estrecha colaboración con los equipos de funciones individuales, pero de forma más formal. Crearon un acuerdo con los equipos que se parecía a este.

Los "Niveles de Servicio" que los probadores de infraestructura están preparados para proporcionar son:

- Sin participación: si se requieren pruebas, serán planificadas, ejecutadas y monitoreadas por un representante de un equipo apropiado para la actividad (por ejemplo, DBA, Red, Infraestructura).
- Orientación: Los evaluadores de infraestructura brindarán orientación para ayudar a planificar una prueba adecuada. La prueba será ejecutada por un representante de un equipo apropiado para la actividad (por ejemplo, DBA, Red, Infraestructura). Los evaluadores pueden revisar los resultados de las pruebas y ayudar con la evaluación de riesgos para una decisión de liberación/aceptación.
- Planificación: los evaluadores de infraestructura prepararán un plan de prueba basado en riesgos. que define el enfoque y el alcance de las pruebas, enumera las pruebas que se realizarán y los criterios de aceptación, e identifica los riesgos asociados con el proyecto. Este plan se entregará a un representante de un equipo apropiado para la actividad (por ejemplo, DBA, Red, Infraestructura) para su ejecución. Los evaluadores de infraestructura pueden revisar el progreso y los resultados de las pruebas y ayudar con la evaluación de riesgos para una decisión de liberación/aceptación.
- Pruebas de infraestructura: los evaluadores prepararán un plan de pruebas basado en riesgos. que define el enfoque y el alcance de las pruebas, enumera las pruebas que se realizarán y los criterios de aceptación, e identifica los riesgos asociados con el proyecto. Los probadores de infraestructura ejecutarán las pruebas y brindarán retroalimentación a

al equipo del proyecto sobre el estado de ejecución. Los problemas se registrarán, rastrearán, tomarán medidas y escalarán utilizando el sistema estándar de seguimiento de defectos.

Los evaluadores de infraestructura presentarán los resultados de las pruebas y la evaluación de riesgos para una decisión de liberación/aceptación.

Esas definiciones son más formales de lo que normalmente usaríamos en equipos ágiles. Sin embargo, muestra cómo los especialistas en pruebas pueden contribuir en diferentes situaciones, ayudando a gestionar el riesgo y proporcionando retroalimentación sobre la infraestructura, incluidos los entornos de prueba y producción. Anteriormente en este capítulo aprendimos cómo los equipos de Michael Hüttermann probaron su infraestructura utilizando herramientas de código abierto como Puppet, Vagrant y Jenkins. Veremos más ejemplos.

Infraestructura de prueba

Stephan Kämper , a probado en Alemania, explica más detalles sobre cómo prueba la infraestructura.

En mi contexto, la infraestructura es el hardware y software adicional. que necesita para ejecutar pruebas. Por lo general, incluye un sistema de CI, incluidos los trabajos que ejecuta el sistema; crear herramientas como make, rake o Maven; y las máquinas en las que se ejecuta la CI. Si el (inalámbrico) La red o el firewall también deben considerarse parte de sus pruebas. La infraestructura depende de su propio contexto.

En mi equipo actual (a finales de 2013), realizamos comprobaciones automáticas cada compromiso con el sistema de control de fuente, crear nuevos artefactos de software (elementos de compilación implementables) con regularidad y ejecutar de forma independiente pruebas de integración contra estos artefactos en máquinas virtuales recién construidas. Estos artefactos se someten a una prueba de integración del sistema y finalmente llevados al sistema de producción a través del sistema CI.

Todo el flujo de trabajo, desde un compromiso único hasta la producción, está orquestado y respaldado por el sistema CI. Desde que escribimos el código para hacer todo esto, me pareció muy buena idea probarlo también. La habilidad para La puesta en marcha con cambios depende de ese código.

¿Qué pasa si la Infraestructura Prueba

No sugiero que pruebe su sistema CI como tal, y tampoco lo hago.

Le recomendamos que pruebe su sistema de compilación. Sin embargo, creo que el El código que usted escribe alimenta a estos sistemas debe ser probado.

Como ejemplo, supongamos que está utilizando Jenkins como sistema de CI y rastillo como herramienta de construcción. Usamos Jenkins Job DSL/Plugin, que permite escribir trabajos de Jenkins como código, en contraste con la forma habitual de configurar subirlos a través de la interfaz gráfica de usuario. De esa manera podemos tener nuestro Código Jenkins bajo control de versiones.

Dado que las tareas de rake son código Ruby, las tareas delegan el trabajo a Ruby. clases, módulos o métodos, que a su vez pueden desarrollarse y probado como cualquier otro código. Sin duda, lo mismo es posible para otros combinaciones de sistemas de construcción y lenguajes de programación.

Uno Advertencia

Algunos trabajos de Jenkins realmente cambian el "mundo", es decir, implementan software a los sistemas de producción. Esto es difícil de probar en un laboratorio. ambiente ya que el objetivo de un sistema de producción es, bueno, dejando la seguridad de un entorno de prueba.

Pero incluso en este caso, puede parametrizar los trabajos de Jenkins para que puede implementar en un sistema de prueba o de preparación y en producción utilizando el mismo guión. De esa manera, la implementación en un sistema de prueba es, de hecho, la prueba para implementar en producción. Tenga en cuenta que incluso cuando esto funciona bien, el La implementación en producción aún puede estar interrumpida, ya que es posible que implemente al servidor equivocado.

Stephan enfatiza el valor de probar implementaciones automatizadas. Otros tipos de pruebas de infraestructura que los evaluadores pueden realizar suelen formar parte de las pruebas del Cuadrante 4, incluidas la conectividad, la confiabilidad, la conmutación por error o las copias de seguridad y restauraciones.

A menudo se pide a los equipos que realicen pruebas en producción, monitoreando la actividad o el rendimiento del sitio. El emparejamiento para monitorear archivos de registro de producción tiene ventajas similares a las de la programación y prueba en pares. Es fácil concentrarse en una cosa y pasar por alto otra por completo, por lo que un segundo par de ojos es invaluable.

Hemos hablado sobre lo que hace DevOps para las pruebas y la calidad y cómo los evaluadores pueden ayudar con las actividades de DevOps. En la parte final de esta sección, cubriremos la intersección de ambos.

Aprovisionamiento automatizado de estados base de configuración

Ben Frempong , prueba de ingeniero, continúa Dell

historia, almacenamiento que describe cómo automatizaron la base de configuración estados para que las pruebas sean mucho más fáciles.

A mediados de 2012, nuestro equipo de pruebas de almacenamiento enfrentó desafíos presupuestarios y de personal. Tuvimos que idear una solución sostenible a una realidad que enfrenta la mayoría de las organizaciones de prueba en un momento u otro: cómo hacer más con menos recursos y al mismo tiempo mejorar continuamente la calidad de los productos que entregamos a nuestros clientes. La solución nos obligó a reinventarnos e implementar estrategias de automatización sostenibles e innovadoras en nuestros proceso de prueba para lograr tres objetivos clave:

1. Maximizar la utilización de nuestros recursos de hardware.
2. Mejorar la eficiencia de nuestro personal de ejecución de pruebas.
3. Capture métricas de automatización.

La primera tarea fue establecer algunos estándares para nuestra iniciación en la automatización. Elegimos Python como nuestro lenguaje de programación estándar y seleccionamos un marco de automatización centralizado y desarrollado internamente para control remoto implementación de guiones. A continuación, nos centramos en dos equipos que tenían características similares. Necesidades: una local y otra remota. Después de analizar su diario tareas, organizamos sus actividades de prueba en el siguiente flujo de trabajo secuencia:

1. Configure el hardware base.
2. Instale el sistema operativo.
3. Instale actualizaciones (firmware/BIOS/controladores de dispositivo OS).
4. Instale los periféricos de la solución/producto.
5. Instale y configure paquetes de soluciones/productos específicos.
6. Ejecute un conjunto de casos de prueba manuales y automatizados.
7. Realizar pruebas exploratorias.
8. Desmonte y reprovisione el sistema para otra configuración.

Observé que los primeros tres pasos, el estado base del aprovisionamiento de flujos de trabajo, eran a menudo manuales, repetitivos y requerían mucho tiempo. secuencia de tareas con el mismo sistema operativo, en el mismo hardware o familias de plataformas. A veces había configuraciones menores. cambios como periféricos de almacenamiento adicionales, pero estos cambios no impactar el estado base de cualquier configuración. Además, el estado base

Las configuraciones a menudo tenían que volver a implementarse para validar la compilación del software. Lanzamientos o correcciones de errores durante los ciclos de prueba de un programa. Se podría utilizar un proceso automatizado para crear imágenes de disco, archivos que contengan el contenido y estructura de un volumen de disco o dispositivo de almacenamiento de datos, con las configuraciones base necesarias. Estas imágenes podrían usarse para automáticamente reprovisionar el estado base de cada configuración.

Las soluciones de imágenes existentes en ese momento funcionaban muy bien en entornos de implementación grande, pero no tan bien en entornos de prueba, donde uno es reprovisionando continuamente el mismo hardware. Además, tomaron hasta 90 minutos o más, dependiendo del sistema operativo, para capturar una imagen. Es posible que se necesiten varias horas para restaurar imágenes que no sean de Windows como el sistema operativo Linux. Esto explica por qué los evaluadores prefirieron el manual. Intentamos automatizar las soluciones de imágenes disponibles en el mercado, pero ninguna solución admitía la captura y el procesamiento automatizados de imágenes. restauraciones para los tres entornos de sistema requeridos que necesitábamos soporte: Windows, Linux y ESXi5. Teniendo en cuenta la matriz de hardware que debía validarse, tendríamos que comprar más hardware y asignar más configuraciones por probador, o agregar más probadores. Debido a nuestros desafíos presupuestarios, ninguna de las soluciones era una opción.

En cambio, priorizamos la automatización de los flujos de trabajo de aprovisionamiento con algunas ideas innovadoras mediante el desarrollo de nuestra propia imagen totalmente automatizada/ solución de redistribución. Primero, reempaquetamos el kernel de la herramienta de imágenes de código abierto Clonezilla para que funcione con nuestro marco de automatización. A continuación, escribimos secuencias de comandos de imágenes y redespliegue totalmente automatizadas para imágenes de Windows, Linux y ESXi5. Estos scripts capturan automáticamente y volver a implementar imágenes de estado base en 10 a 15 minutos o menos.

Los resultados han sido positivos. Uno de los equipos del proyecto normalmente necesitaba hasta tres probadores por iteración de prueba para cubrir de 12 a 18 configuraciones. Despues de implementar la creación de imágenes y restauración automatizadas tareas, el líder del equipo pudo ejecutar él solo la iteración de prueba sin recursos adicionales. Incluso tuvo más tiempo para pruebas exploratorias porque el marco de ejecución de pruebas le permitió para iniciar una secuencia de tareas automatizadas, que podrían ejecutarse durante la noche o durante el almuerzo.

Con nuestro nuevo enfoque para la automatización de pruebas de flujo de trabajo, ahora podemos hacer más con menos recursos y al mismo tiempo dar a nuestro personal cualificado más tiempo para pruebas exploratorias. Esto nos permite continuar entregando productos de calidad. También podemos reducir el gasto y maximizar la utilización de nuestros recursos de hardware. Por último, desde una perspectiva de proceso, Podemos utilizar la información sobre nuestras capacidades de automatización. en nuestra planificación de pruebas, estrategia de pruebas y decisiones comerciales.



Los evaluadores, programadores y expertos en operaciones pueden colaborar en el proceso de CI para hacer que los trabajos de construcción sean más sólidos, los resultados de las pruebas sean más confiables y las implementaciones en diversos entornos sean oportunas y sensatas. Si todo el equipo comprende sus entornos operativos, pueden ahorrar muchas horas dedicadas a investigar fallas en las pruebas. Por ejemplo, DevOps puede ayudar a crear comprobaciones de conectividad o configuración antes de la implementación, lo que elimina la pérdida de tiempo rechazando compilaciones. Alternativamente, los evaluadores y programadores pueden analizar conjuntos de pruebas para identificar pruebas duplicadas, pruebas que no agregan valor y formas de mejorar la confiabilidad. También pueden buscar lagunas en la automatización de las pruebas de regresión y trabajar para llenarlas.

Las prácticas de DevOps ayudan a nuestros equipos a crear código de prueba de regresión de manera eficiente e implementarlo según sea necesario en entornos que admitan pruebas exploratorias efectivas. Los profesionales cualificados de DevOps nos ayudan a implementar herramientas para respaldar las pruebas de todos los atributos de calidad necesarios, como seguridad, rendimiento y confiabilidad. Estas capacidades son esenciales para que los equipos multifuncionales entreguen software de alta calidad que proporcione valor empresarial con frecuencia y a un ritmo sostenible.

Resumen

DevOps es un ejemplo perfecto del enfoque de todo el equipo hacia la calidad en el trabajo. Reúne a especialistas en generalización con diferentes habilidades en forma de T para ayudar a garantizar varios aspectos de la calidad. Incluso cuando el desarrollo y las operaciones son departamentos separados, pueden trabajar juntos para lograr objetivos compartidos. Aquí hay algunos puntos sobre DevOps que cubrimos en este capítulo:

- DevOps es una combinación de desarrollo, pruebas y operaciones. participó en prácticas que agilizan el proceso de entrega, brindan retroalimentación oportuna para mejorar el tiempo del ciclo y la calidad del software, y facilitan la colaboración entre todos los roles del equipo.
- Los miembros del equipo con habilidades de administración de sistemas y operaciones colaboran con otros miembros del equipo para construir una CI y una infraestructura de implementación que admite tiempos de ciclo cortos.
- Los equipos de desarrollo, incluidos los evaluadores, pueden ayudar a implementar una enfoque guiado por pruebas para construir esa infraestructura y garantizar que la infraestructura continúe funcionando de manera efectiva.

- Los profesionales de DevOps ayudan a crear e implementar controladores, bibliotecas y marcos de automatización de pruebas, y herramientas de generación de datos de prueba.
- Comprenda los entornos de prueba y canalización de compilación de su equipo para aprovechar dónde puede utilizar la automatización para complementar sus esfuerzos de prueba exploratoria.
- Cada organización necesita experimentar con diferentes configuraciones de hardware para encontrar aquellas que proporcionen flexibilidad, coherencia y facilidad de mantenimiento.
- Los evaluadores pueden contribuir a las pruebas de infraestructura haciendo buenas preguntas y ayudando con un enfoque basado en riesgos, así como participando activamente en la configuración e implementación de pruebas.
- Automatizar el aprovisionamiento de estados base de configuración permite a los equipos hacer más con menos recursos, utilizando estrategias de automatización sostenibles que proporcionan métricas útiles y permiten más tiempo de prueba exploratoria.
- DevOps puede ayudar al equipo a encontrar formas de superar impedimentos como pruebas automatizadas frágiles o lentas.

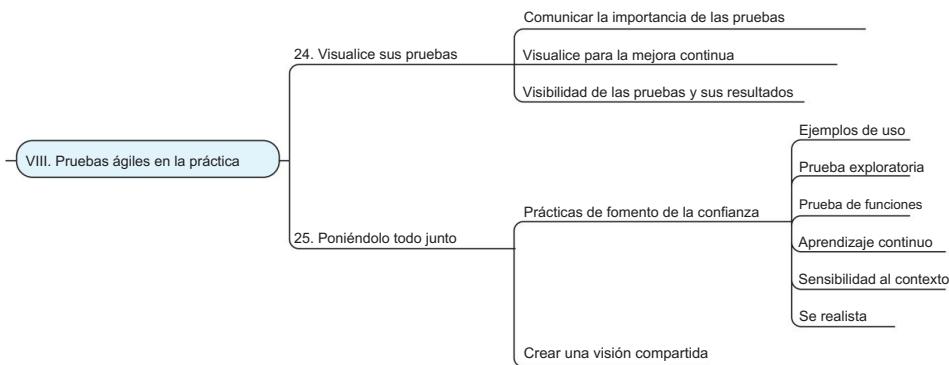
Esta página se dejó en blanco intencionalmente.

Parte VIII

Pruebas ágiles en la práctica

Concluimos More Agile Testing con recomendaciones que creemos que serán de gran ayuda para ayudarle a usted y a su equipo a incorporar calidad en sus productos de software. Una práctica crucial es hacer que las actividades de prueba sean transparentes y proporcionar información que sea accesible y fácil de entender. La visibilidad hace que los impedimentos sean inmediatamente obvios, promueve la discusión y la colaboración, y proporciona guías a lo largo del camino, mientras nos esforzamos por mejorar continuamente nuestro proceso y nuestro software. En esta parte, nosotros y nuestros colaboradores compartimos formas de hacer que las actividades de prueba sean altamente transparentes.

Finalmente, compartimos prácticas que generan confianza en la entrega de productos de software que aportan valor a nuestros clientes y a nuestro negocio. El uso de ejemplos para guiar el desarrollo, las pruebas exploratorias, las pruebas de funciones, el aprendizaje continuo, la sensibilidad al contexto y mantenerlo real durante las pruebas ayudan a los equipos a ganar coraje y mejorar continuamente. Los pilares de las pruebas, explicados por David Evans, nos brindan la base desde la cual aumentar nuestra confianza a la hora de publicar cambios de software.



- Capítulo 24, “Visualice sus pruebas” ■ Capítulo 25, “Juntándolo todo”

Capítulo 24

Visualice sus pruebas

Comunicar la importancia de las pruebas
24. Visualice sus pruebas
Visualice para la mejora continua
Visibilidad de las pruebas y sus resultados

Una de las razones por las que el desarrollo ágil funciona tan bien es que hace que el proceso de desarrollo sea transparente. Ingrese al área de trabajo de cualquier equipo ágil y probablemente verá algunos gráficos visibles de gran tamaño. Puede saber de un vistazo en qué está trabajando el equipo, qué impedimentos pueden haber en su camino, cuánto trabajo se ha completado y si las pruebas de regresión están pasando actualmente o no. Dedicamos una sección de Pruebas ágiles (págs. 354 a 66) a prepararnos para la visibilidad. Aquí, exploraremos más formas de aumentar la visibilidad de las pruebas en un equipo ágil.

Comunicar la importancia de las pruebas

Los equipos ágiles representan el trabajo planificado, en progreso y completado de manera visual, con notas adhesivas o tarjetas en filas y columnas en una pared, o una herramienta de seguimiento de proyectos en línea que simula un tablero físico. Cualquiera que mire el tablero debería poder comprender de un vistazo qué actividades de prueba están planificadas, cuáles están en progreso y qué historias han sido probadas y aceptadas.

Hacer que las tareas de prueba sean muy visibles alienta a los miembros del equipo a asumir tareas no completadas, incluso aquellas que quedan fuera de su área de especialidad. Los impedimentos también pueden representarse claramente, de modo que se aborden a tiempo. Estos tableros deben diseñarse para ayudar al equipo a concentrarse en completar una historia a la vez, o al menos un pequeño conjunto de historias. Ayudan a los equipos a recordar completar las actividades de prueba de una historia antes de apresurarse a comenzar la siguiente.

Kanban y pruebas

Steve Rogalsky , un agilista

Los tableros proporcionan unidireccionales
visibilidad.

Protegra, explica cómo kanban en

hacer que las actividades de prueba sean altamente

Alguno Fondo

Empezaré diciendo que mis primeras experiencias ágiles fueron tipo Scrum. Una vez que comencé a comprender la metodología ágil, rápidamente me incliné por los sistemas basados en flujo como kanban. Me tomó un tiempo sentirme cómodo con los límites de trabajo en progreso (WIP) de kanban. Durante los primeros proyectos, ignoramos los límites de WIP y, en cambio, prestamos atención al tablero: nuestro objetivo era el equilibrio en lugar de los límites. Una vez que nos cansamos de intentar mantener el tablero en equilibrio nosotros mismos, cambiamos a límites de WIP.

Tenga en cuenta que combinamos nuestro enfoque basado en el flujo kanban con algunos eventos que ocurren cada dos semanas: demostraciones, reuniones de planificación/diseño y retrospectivas. Además, actualmente no contamos con evaluadores capacitados en nuestro equipo. Nuestros analistas de negocios y programadores comparten la carga de pruebas. Tenemos clientes que realizan una prueba de aceptación final, pero están capacitados como expertos comerciales y recientemente comenzaron a realizar pruebas debido a este proyecto. ¡Están aprendiendo rápido y nos ayudan mucho!

Nuestro Junta

Nuestros tableros iniciales eran bastante básicos. Las columnas "Listo", "WIP" y "Listo" pronto fueron reemplazadas por "Listo", "Análisis", "Desarrollo".

Columnas "Prueba" y "Listo". Después de conocer el juego "Obtener Kanban", agregamos una columna "Listo para el siguiente estado".

Nuestra placa ha pasado por algunas iteraciones desde entonces, pero la Figura 24-1 muestra lo que estamos usando actualmente.

NEXT F6	Analysis + Design	Development			QA	
WIP LIMIT = 10	WIP LIMIT = 5	WIP LIMIT = 3	WIP LIMIT = 4	WIP LIMIT = ∞	Final Approval	Done!!
Due: Jan. 7, 2014	WIP Ready 4 dev Design ready for review Review done Review done	WIP Ready 4 dev Design ready for review Review done Review done	WIP Ready 4 dev Design ready for review Review done Review done	WIP Ready 4 dev Design ready for review Review done Review done	QA Done!!	
	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10
	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10

Figura 24-1 Tablero kanban de Steve

La columna "Análisis y diseño" incluye subcolumnas de "WIP" y "Listo para desarrollo". El objetivo de esta columna "WIP" es escribir casos de prueba (ejemplos) con nuestros clientes para asegurarnos de que entendemos lo que esperan de esta historia. La definición de "Listo para desarrollo" son casos de prueba (ejemplos) escritos y revisados por el cliente con al menos una prueba fallida. (Por supuesto, al principio, es probable que todos ellos fallen las pruebas).

La columna "Desarrollo" incluye subcolumnas de "WIP", "Listo para la prueba de humo", "Prueba de humo" y "Listo para el control de calidad". "Listo para la prueba de humo" indica que la historia se implementó en nuestro entorno de prueba, se revisó el código y el desarrollador cree que todas las pruebas pasan.

Definimos prueba de humo en el sentido de que un analista, programador o evaluador ha probado la función y cree que todos los casos de prueba pasan. Una historia califica como "Lista para control de calidad" cuando pasan todas sus pruebas de humo.

Las subcolumnas de la columna "QA" son "WIP" y "Listo para la aprobación final". Aquí es donde nuestro cliente da el último paso. En este punto, nos sorprendería mucho que encontraran un defecto o un caso de prueba omitido. Si encuentran algo, intentamos solucionarlo de inmediato y agregar las pruebas que faltan si es necesario. La definición de "Listo para la aprobación final" es que el cliente cree que todos los casos de prueba pasan y que la historia está completa.

La aprobación final recae en nuestro principal cliente. Está bastante ocupada, por lo que no prueba todo, pero quiere echar un último vistazo antes de pasar cualquier historia a la columna "Completa".

¿Cómo definimos "completo"? ¡El cliente está contento y está listo para implementarse!

Cómo Él Se relaciona con Pruebas y Probadores

Hemos visto varios beneficios al utilizar un sistema basado en flujo. Proporciona una visualización potente. Todos, incluidos los evaluadores, pueden ver lo que se avecina y prepararse para ello. Dado que nuestras definiciones en la mayoría de las columnas kanban incluyen las frases "aprobar pruebas" o "reprobar pruebas", comunica la importancia de las pruebas a todo el equipo y respalda un enfoque de calidad de todo el equipo.

Las columnas "Listo para" ofrecen mucha libertad en la asignación de tareas.

Todos hacemos nuestro propio trabajo en lugar de que un gerente nos asigne el trabajo. Casi sin excepciones, una vez que movemos una tarjeta a "Completar", no regresa. Cuando el tablero se desequilibra, apoya el desarrollo de habilidades multifuncionales. Si los evaluadores tienen poco trabajo, ayudan a los programadores en lo que pueden o, más a menudo, ayudan a los analistas a facilitar y recopilar los casos y ejemplos de prueba. Cuando los programadores

o los analistas tienen poco trabajo, generalmente buscan ayuda con las pruebas primero. Esto crea una mayor comprensión en el equipo sobre el papel y la importancia de las pruebas. Su experiencia como evaluadores ha impulsado tanto a programadores como a analistas a prestar mayor atención a los casos de prueba.

Aunque todavía no estamos haciendo ningún emparejamiento significativo, seguimos viendo pequeños casos de emparejamiento, incluido el emparejamiento entre desarrollo y cliente durante el control de calidad. Esperamos que esa tendencia continúe.

En general, nos encanta este enfoque. Nos da mucho orgullo y confianza. Tenemos la sensación de que Deming nos sonríe cada vez que una historia pasa a "Completar".

Steve señala que la visualización que proporciona el tablero del equipo es un beneficio clave. Sin embargo, Mike Talks nos recordó una posible desventaja de los tableros kanban. Le permiten flexibilidad, pero si agrega demasiados estados para cubrir todas las eventualidades, terminará agotando toda la simplicidad de lo que comenzó como una solución elegante. Revise constantemente su flujo de trabajo y formule una pregunta que sugiere Liz Keogh (Keogh, 2014b): "¿Es esta una representación útil de la realidad?" Si parece que las tareas pasan por encima de varios estados, puede ser una señal de alerta que indique que esos estados pueden no ser relevantes.



Figura 24-2 Un tablero vertical estilo kanban utiliza el único espacio disponible en la pared.

Utilice su creatividad y experimente con diferentes formas de mantener las actividades de prueba en primer plano. Lisa trabajaba en un equipo donde el único espacio en la pared estaba en una columna en el medio de la sala y dos de los cinco miembros estaban remotos. El equipo colocó un tablero estilo kanban en formato vertical en la columna (ver Figura 24-2) y publicó fotos del mismo cada vez que se movía una tarjeta en la wiki del equipo. Lisa, que trabajaba de forma remota, guardaba su propia copia en la puerta de un armario de la oficina de su casa. Podía ver de un vistazo en qué trabajar a continuación.

Un guión gráfico simple, como el ejemplo de la Figura 24-3, con sólo cuatro columnas: "Por hacer" (o "Listo"), "En progreso", "Por revisar" y "Terminado", es una manera efectiva de visualizar el progreso de las pruebas. Todas las tareas, incluidas las de codificación y prueba, avanzan a lo largo del tablero. Cuando el equipo establece límites de WIP a nivel de historia, digamos solo dos historias en progreso a la vez, los miembros del equipo deben ayudar a otros a completar al menos una historia antes de comenzar la siguiente. Cuando se utiliza un código de colores para diferenciar las actividades de prueba, es fácil ver cuándo están apiladas, ya s



Figura 24-3 Guión gráfico simple

“En progreso” o “Para revisar”. Esta visibilidad recuerda a los miembros del equipo que deben ayudar en las tareas para terminar cada historia. Quizás un programador pueda ayudar a revisar las pruebas o realizar pruebas exploratorias.

Es posible que los equipos dispersos y distribuidos prefieran utilizar un tablero en línea para realizar un seguimiento del progreso de las pruebas junto con otras actividades de desarrollo, pero alentamos a esos equipos a encontrar una manera de mantenerlo frente a todos tanto como sea posible. Algunos equipos distribuidos utilizan monitores o proyectores de gran tamaño para mantener a la vista su tablero de seguimiento de proyectos en línea y su panel de integración continua (CI) en cada área del equipo.

Proporcionamos más ejemplos de guiones gráficos en el Capítulo 15 de Pruebas ágiles, y una búsqueda en línea de “gráficos ágiles grandes y visibles” o “guiones gráficos ágiles” proporcionará más ejemplos para alimentar su imaginación.

Visualice para la mejora continua

Uno de los diez principios para los probadores ágiles que describimos en Pruebas ágiles es practicar la mejora continua. Siempre deberíamos estar buscando formas de hacer un mejor trabajo. Utilice gráficos grandes y visibles para mostrar los objetivos de mejora, los experimentos en curso y los resultados de los experimentos completados.

Según nuestra experiencia, si su equipo tiene un obstáculo en su camino, es útil hacerlo visible. Cuando lo tienes “en la cara”, recuerdas hacer algo al respecto. Cuando identifique áreas de mejora en sus retrospectivas, establezca objetivos INTELIGENTES (específicos, mensurables, alcanzables, relevantes, con un límite de tiempo) o encuentre alguna manera de hacer que el problema sea más visible. Esto prepara el escenario para la mejora continua.

Usando Kanban para mejorar continuamente

María Walsh , pruebas en Paddy Power Irlanda, comparte las experiencias de su equipo con el uso de mejoras. el sistema kanban a un visualizar áreas de

Llevamos más de un año utilizando kanban en nuestro departamento y hemos realizado grandes mejoras en nuestro proceso. Lo usamos para visualizar áreas que podríamos mejorar, ajustar nuestro proceso y ver dónde podemos experimentar. Es una herramienta que todo el equipo posee y

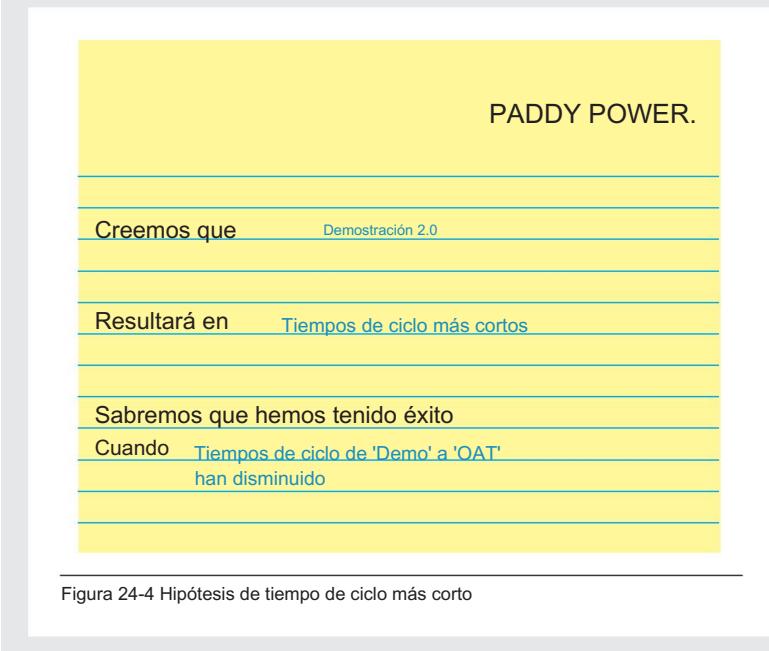
que utilizamos para la mejora continua. Si notamos que el tablero no ha cambiado en un tiempo, sabemos que es posible que estemos flojos en nuestros experimentos.

Uno de nuestros objetivos de mejora más recientes fue intentar reducir los tiempos de nuestros ciclos en tres días. Nuestro tablero kanban y sus límites de WIP nos ayudaron a identificar un área de posible mejora. Vimos que había un cuello de botella en nuestra columna "Esperando UAT" y comenzaron las conversaciones sobre la duplicación de esfuerzos en esta área.

Varias historias de usuarios que aportaban valor empresarial tardaron más de tres días en aceptarse. ¡Todo ese valor comercial podría haber llegado a nuestro cliente tres días antes!

El cuello de botella en la aceptación afectó a los propietarios de productos, desarrolladores, analistas de negocios, personal de operaciones, evaluadores... en resumen, ¡a todos! Los propietarios de productos dedicaron más tiempo a preparar, ejecutar pruebas y documentar la actividad. Los clientes tuvieron que esperar más para que se les entregara valor comercial. Los tiempos de ciclo largos significaron un ciclo de retroalimentación más largo. El desarrollo se ralentizó. La repetición de pruebas ya realizadas durante las pruebas exploratorias ralentizó las cosas.

Nuestros propietarios de productos sugirieron que intentáramos "mejorar la demostración", en otras palabras, centrarnos en satisfacer a los propietarios de productos con la demostración del trabajo recién completado. Investigamos un poco y decidimos intentar introducir la Demo 2.0. Una historia de hipótesis para esto podría leerse algo como lo que se muestra en la Figura 24-4.



Algunos miembros del equipo se encargaron de trabajar con nuestros propietarios de productos y garantizar que la Demo 2.0 satisficiera sus necesidades.

Escribieron un guión de demostración y lo enviaron junto con el informe de la prueba de aceptación en la invitación a la demostración, que se programó de antemano con el propietario del producto. Durante la demostración, repasamos las pruebas de aceptación de cada función y ejecutamos el script de demostración. Les dimos tiempo a los propietarios de productos para que hicieran preguntas y realizaran pruebas adicionales según fuera necesario.

Eso es todo. Parece bastante simple, ¿verdad? Al momento de escribir este artículo, ha pasado casi un mes desde la primera Demo 2.0. Mirando hacia atrás en nuestra historia de hipótesis, de hecho hemos tenido éxito al reducir los tiempos de ciclo desde la demostración hasta las pruebas de aceptación operativa (OAT).

Veamos las métricas. En la Figura 24-5, podemos ver que el tiempo de ciclo promedio (la línea inferior) desde la demostración hasta OAT en septiembre fue de tres días.

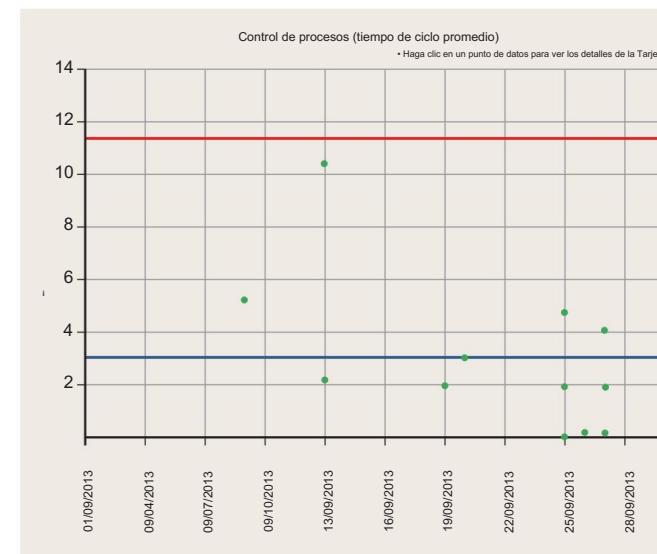


Figura 24-5 Tiempos de ciclo promedio desde la demostración hasta la OAT antes de la Demo 2.0

La Figura 24-6 muestra los tiempos del ciclo desde la demostración hasta el OAT a partir del 1 de octubre, el día de la primera Demostración 2.0. Podemos ver que el ciclo promedio

El tiempo desde la demostración hasta OAT en este mes se ha reducido a menos de uno. Nota: La escala en el y El eje se ajustó para proporcionar una mejor (visualización de las diferencias).

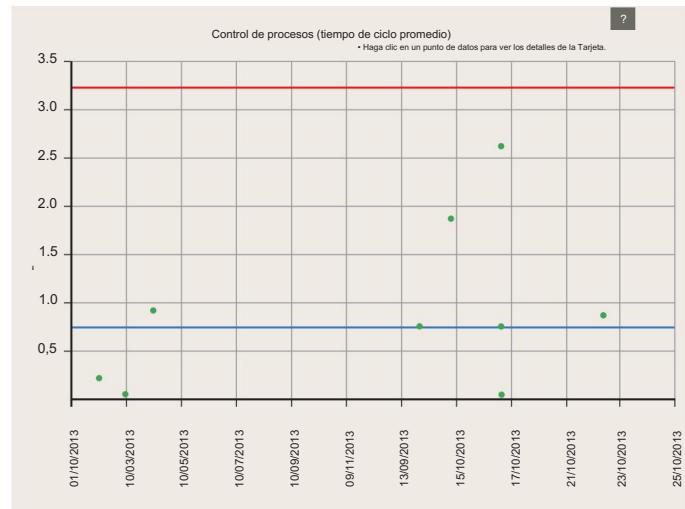


Figura 24-6 Tiempos de ciclo promedio desde la demostración hasta la OAT después de la Demo 2.0

Podemos ver la disminución del tiempo a través de métricas al tomar los tiempos de ciclo de la demostración a OAT del mes anterior de septiembre y compararlos con el mes actual de octubre, donde presentamos la Demo 2.0. Entonces, ¿lo hemos logrado? Sí, ¡recortamos dos días completos de nuestro ciclo con esta iniciativa!

Los críticos podrían preguntarse si la demostración más larga aumentó los tiempos de ciclo desde el inicio de una historia hasta su aceptación, contrarrestando el ahorro de dos días, pero las métricas muestran que la demostración no ha aumentado los tiempos de ciclo. De hecho, el tiempo empleado en la línea de demostración ha disminuido en el último mes. Esto puede deberse al hecho de que reducir el desperdicio está en la mente del equipo.

Todo el equipo puso mucho esfuerzo para que esto despegara, incluida la organización inicial de cómo se vería la nueva demostración, ponerla en acción, obtener comentarios y mejorarl a lo largo del camino. Considerándolo todo, creo que la Demo 2.0 ha sido un gran éxito. En el futuro, podríamos considerar si necesitamos siquiera la fase de aceptación.

Las historias de Steve y Mary tienen mucho en común. El mensaje no es que todo el mundo debería utilizar un sistema kanban. Más bien, es esencial que un equipo pueda visualizar su trabajo de prueba y su nivel de calidad y hacer que esta información sea transparente para el negocio. Puede probar cualquier tipo de historia, tablero de tareas o sistema de seguimiento en línea para ver si eso le brinda la visibilidad que necesita. Podría implicar un monitor grande que muestre resultados de CI en vivo en la sala de su equipo o algo tan simple como un dibujo en una pizarra, una hoja de cálculo, una lista de verificación o una página wiki. El criterio principal es ponerlo en el centro de atención de todo el equipo y de todas las partes interesadas. Esta transparencia ayuda a los equipos a identificar fácilmente el mayor problema en el que trabajar a continuación. Pueden realizar una lluvia de ideas sobre experimentos para solucionar ese problema y mejorar continuamente sus esfuerzos de prueba.

No espere hasta el final de la iteración para comenzar a demostrar pequeños incrementos del trabajo completado a sus partes interesadas. La colaboración frecuente le ayuda a mantener el rumbo y evitar perder el tiempo.

Visibilidad de las pruebas y sus resultados

A menudo hemos escuchado a los equipos de desarrollo quejarse: "¡No sabemos qué están haciendo los evaluadores!". Para nosotros, esto se traduce en: "No tenemos confianza en lo que se está probando". Hay muchas maneras de mostrar lo que se está probando.

Hablamos sobre el uso de mapas mentales de prueba para realizar una lluvia de ideas con el equipo en el Capítulo 7, "Niveles de precisión para la planificación". La Figura 24-7 muestra un mapa mental que Lisa y otro evaluador crearon en una pizarra móvil para visualizar un nuevo proceso de cobro de tarifas de cuenta por lotes, que se integraría en un sistema de procesamiento de transacciones existente. Mientras creaban el mapa mental, pensaron en preguntas para hacerle al propietario del producto y al programador. El equipo utilizó el mapa mental en varias iteraciones mientras trabajaban en las nuevas funciones. Cada vez que surgían preguntas, los miembros del equipo se reunían frente al mapa mental para discutirlas. A medida que se completaban las pruebas para cada nodo del mapa mental, lo marcaban en la pizarra, haciendo visible el progreso.

Los equipos pueden ampliar las pruebas de aceptación iniciales para brindarle a todo el equipo visibilidad de las pruebas. Hacerlos accesibles a todos fomenta

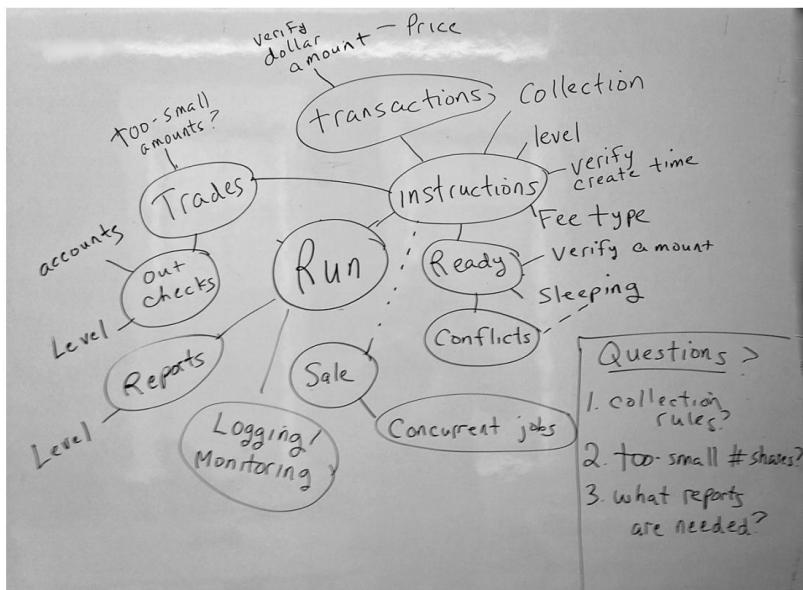


Figura 24-7 Un mapa mental de prueba

colaboración y anima a los miembros del equipo de desarrolladores y clientes a comentar y sugerir nuevas ideas.

En el Capítulo 7, “Niveles de precisión para la planificación”, la Figura 7-6 mostró un ejemplo de una matriz de prueba. Esta es una forma sencilla de hacer que las pruebas de alto nivel sean visibles para las partes interesadas fuera del equipo. También ofrece una perspectiva diferente sobre las pruebas para ayudar a evitar el problema de quedar atrapado en historias y olvidarse del panorama general.

Las pruebas consisten en proporcionar información lo más rápido posible. Piense en cómo puede hacer visible la información aprendida en las pruebas.

¿Cómo puedes comunicarlo? ¿Cómo lo escucharán y verán las personas en otros roles? Como analizamos en Pruebas ágiles (págs. 357 a 66), los resultados visibles de las pruebas son una de las formas más importantes en las que podemos medir el progreso.

Esta información adopta muchas formas. La integración continua proporciona una instantánea crítica del estado actual de sus compilaciones y quizás de sus ejecuciones de pruebas automatizadas. Los resultados de las sesiones de pruebas exploratorias pueden ser

consolidado en un documento compartido, una wiki de equipo o un panel de prueba simple. Las pruebas y los resultados pueden anotarse en la historia o aparecer en una herramienta de seguimiento en línea. Las correcciones de errores se pueden capturar como pruebas automatizadas y los defectos que no se corrijen de inmediato se pueden rastrear de la manera más simple y visible posible. Lo más importante es que los resultados de las pruebas deben discutirse en las reuniones del equipo. Utilice los comentarios para guiar sus próximos pasos.

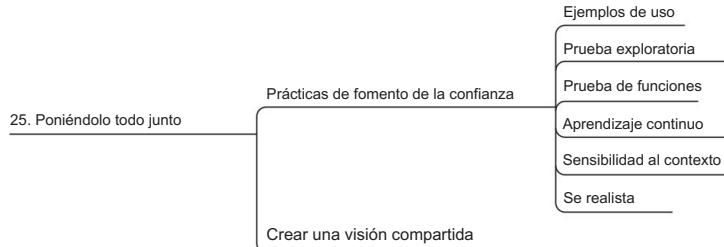
Resumen

La transparencia que proporciona la metodología ágil es uno de sus mayores beneficios para una organización. Las pruebas consisten en proporcionar información, y cuando hacemos que esa información sea fácil de ver, todos pueden hablar de ella. No basta "sólo" con ser visible: es necesario que sea útil.

- Buscar formas de hacer más visibles diversos aspectos de las pruebas.
en su historia, tarea o tablero estilo kanban. Si su proceso tiene una definición de "listo" y una historia no está en el tablero hasta que cumpla con esa definición, es fácil ver cuándo una nueva historia está lista para las tareas de prueba que se abordarán.
- Experimente con diferentes estilos de guiones gráficos físicos y, si es necesario, herramientas de seguimiento en línea para ver qué proporciona la mayor visibilidad del progreso de las pruebas para su equipo.
- Demostrar el nuevo trabajo a los clientes tempranamente y con frecuencia para hacer el progreso sea visible, obtenga comentarios rápidos y reduzca los tiempos del ciclo de la historia.
- Si su equipo encuentra obstáculos para completar con éxito las actividades de prueba, encuentre una manera de hacer que esos obstáculos sean visibles. Puede ser tan simple como una tarjeta bloqueadora de color rojo brillante en un guión gráfico.
- Considere técnicas como mapas mentales que ayuden a los equipos a visualizar su trabajo y progreso. La representación visual ayuda a los equipos a mantenerse conectados a la realidad.
- Las pruebas generan información sobre la calidad y el riesgo. Usar una Variedad de formas de hacer que los resultados de las pruebas y el estado actual sean visibles para todos.

Capítulo 25

Poniéndolo todo junto



Agile Testing concluyó con siete factores clave de éxito para las pruebas en proyectos ágiles. De hecho, creemos que son fundamentales para el éxito de cualquier proyecto, ya sea que lo llames ágil o no. Ellos son:

1. Utilice el enfoque de calidad de todo el equipo.
2. Adopte una mentalidad de prueba ágil.
3. Automatizar las pruebas de regresión.
4. Proporcionar y obtener comentarios.
5. Construir una base de prácticas ágiles básicas.
6. Colaborar con los clientes.
7. Mire el panorama general.

Nosotros y nuestros colaboradores hemos compartido en este libro lo que hemos aprendido desde que se escribió Agile Testing , abordando desafíos que son cada vez más comunes para los equipos. Los factores de éxito originales se aplican a entornos de grandes empresas, equipos distribuidos globalmente, pruebas en diferentes contextos, como software móvil e integrado, almacenes de datos y productos implementados continuamente. Sin embargo, tenemos más y mejores herramientas y técnicas disponibles para ayudar con los desafíos actuales.

Prácticas de fomento de la confianza

Si bien no creemos que existan “mejores prácticas”, sí creemos que existen prácticas de prueba básicas que benefician a la mayoría de los equipos ágiles. Estas prácticas dan a los equipos confianza para ofrecer productos de calidad que deleiten a sus clientes. Con esa confianza viene el coraje de probar cosas nuevas y mejorar continuamente.

Ejemplos de uso

Aprenda a guiar el desarrollo con ejemplos, utilizando el desarrollo impulsado por pruebas de aceptación (ATDD), la especificación por ejemplo (SBE) o el desarrollo impulsado por el comportamiento (BDD). Haga sus ejemplos lo más concretos posible utilizando medios visuales siempre que sea posible (consulte la Figura 25-1).

Entre las prácticas de desarrollo principales que recomendamos se encuentran el desarrollo basado en pruebas (TDD) y la integración continua (CI). Estos contribuyen en gran medida a incorporar calidad al software y aprender de la retroalimentación rápida. Sin embargo, por sí solos no son suficientes. Cuando ofrecer una nueva función es complicado, guiar el desarrollo con enfoque empresarial

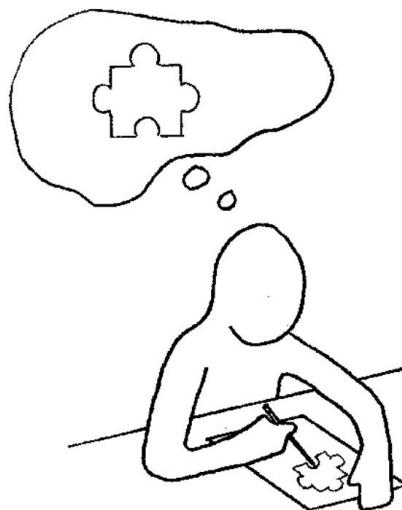


Figura 25-1 Utilice ejemplos reales.

Los ejemplos ayudan a evitar que las historias sean rechazadas varias veces.

Cuando las características propuestas tienen una gran incertidumbre o son tan complejas que es difícil incluso encontrar ejemplos del comportamiento deseado, es posible que necesite algunos ciclos cortos de picos y aprendizaje antes de poder resumir la información en ejemplos concretos.

Podemos ayudar a nuestros expertos en negocios a identificar las características de software más valiosas para implementar combinando mentalidades de prueba y análisis de negocios (BA). Las técnicas ágiles, como el mapeo de historias y el mapeo de impacto, ayudan a los equipos de negocios y de desarrollo a enfocar el esfuerzo del equipo en los objetivos correctos. Las técnicas de BA, como las 7 dimensiones del producto descritas en el Capítulo 9, “¿Estamos construyendo lo correcto?”, promueven nuestros esfuerzos por crear productos valiosos. Aunque no queremos perder el tiempo con grandes análisis iniciales de requisitos en constante evolución, existen formas buenas y rápidas de desarrollar los detalles de la historia antes de comenzar a codificar una característica o historia en particular.

Esta información se puede convertir en pruebas que guíen el desarrollo, muchas de las cuales pueden automatizarse y formar parte de las suites de regresión.

Ayuda a garantizar que entregamos valor útil y nos ayuda a crear ciclos cortos de retroalimentación para aprender más a medida que desarrollamos y entregamos de manera incremental. Todavía tenemos más pruebas por hacer una vez que creamos que la codificación está terminada, pero hemos minimizado las posibilidades de rechazo sin un gran costo inicial. La Parte IV, “Prueba del valor empresarial”, describió técnicas para ayudar a extraer comportamientos deseados y malos comportamientos de los clientes y aprender a guiar el desarrollo con ejemplos.

Si la CI es el latido del corazón de un equipo ágil, guiar el desarrollo con ejemplos es el monitor cardíaco que muestra que estamos en el camino correcto.

Prueba exploratoria

Aunque algunos de los primeros equipos ágiles tenían evaluadores que realizaban algunas pruebas exploratorias, el valor que aportan los evaluadores exploratorios a través de su capacidad de ver más allá de la funcionalidad de la historia se ha vuelto más visible en los últimos años. Probar el valor empresarial (probar tempranamente) consiste en prevenir defectos y, si se prefiere, explorar ideas. Sin embargo, nos referimos a las pruebas exploratorias como probar el software: si no puede evitar los defectos, encuéntrelos lo antes posible (consulte la Figura 25-2).

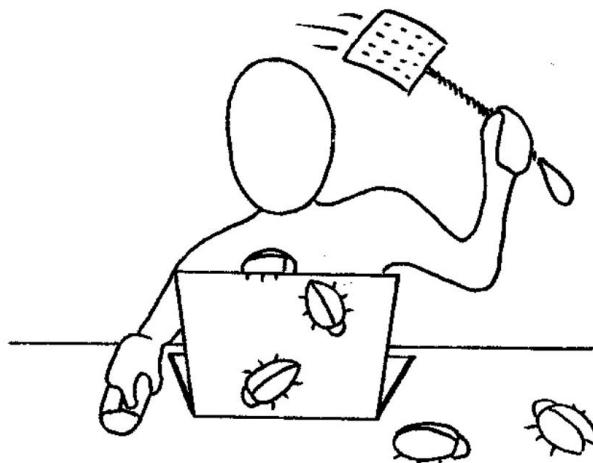


Figura 25-2 Explorar.

El Capítulo 12, “Pruebas exploratorias”, analizó varias técnicas de pruebas exploratorias. Le recomendamos que utilice los enlaces proporcionados en la bibliografía de la Parte V para experimentar y descubrir qué funciona mejor para su situación. Si algunos miembros de su equipo no están familiarizados con las pruebas exploratorias, organice un taller o un dojo de pruebas para que puedan aprender y practicar técnicas.

Algunas técnicas exploratorias son ideales para colaborar con otras disciplinas del equipo. Involucre a sus diseñadores de experiencia de usuario (UX) para crear personas de usuario. Reúna a evaluadores, programadores y analistas para crear y priorizar cartas de prueba. Comparta la información aprendida en sesiones exploratorias con los equipos de desarrolladores y clientes, haciendo que la información sea visible y procesable. Experimente con diferentes enfoques y vea cuál agrega más valor a su equipo.

Prueba de funciones

El desarrollo incremental e iterativo es la piedra angular de la metodología ágil. Dividir las historias en un tamaño pequeño y consistente ayuda a los equipos a lograr un progreso predecible, pero en estos pequeños componentes de las historias, podemos perder el panorama general. Mirar el panorama general es un factor clave de éxito. Necesitamos técnicas

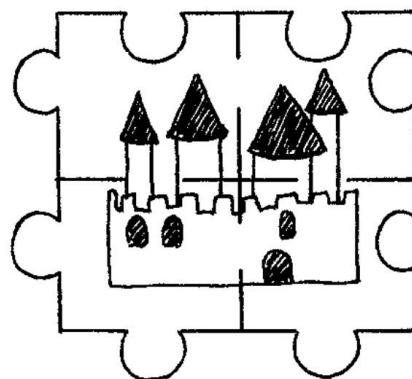


Figura 25-3 Recuerde el panorama general.

para realizar pruebas a nivel de función una vez que los componentes básicos estén completos (consulte la Figura 25-3).

Janet a menudo compara las pruebas de funciones con armar un rompecabezas. No necesariamente es necesario armar todo el rompecabezas para obtener valor de él. Se centra en las partes de la imagen que son más significativas para ella: rostros de las personas en la imagen, hermosas secciones del paisaje (Gregory, 2011).

Cuando probamos características de software, debemos identificar las áreas de mayor valor para los usuarios finales, las partes que “tienen que funcionar” y centrar nuestras pruebas allí. Es posible que estos no estén en la funcionalidad real; el rendimiento, la seguridad o la comodidad pueden ser lo que más valora el cliente.

Mientras su equipo analiza su próximo gran conjunto de funciones, piense en los diferentes tipos de pruebas explicados en el Capítulo 13, “Otros tipos de pruebas”, así como en diferentes modelos, como los cuadrantes de pruebas ágiles. Escriba historias con actividades de prueba que se centren en aspectos de la característica que los usuarios más valoran. Considere los atributos y limitaciones de calidad que afectan las pruebas durante las conversaciones del equipo.

Aprendizaje continuo

Tomarse el tiempo para experimentar, utilizar ciclos cortos de retroalimentación para evaluar qué funciona y qué no, tener una sensación de seguridad personal ante el fracaso.

no será castigado: todos estos son componentes críticos de un entorno que fomente el aprendizaje (ver Figura 25-4). A medida que un equipo sobrevive a su transición inicial hacia el desarrollo ágil y continúa a lo largo de los años, surgirán nuevos problemas y desafíos. Para tener éxito a largo plazo, debemos seguir mejorando. No podemos hacer eso si no aprendemos.

Por supuesto, esto se aplica a todas las disciplinas de un equipo ágil. Pero, según nuestra experiencia, los evaluadores y las pruebas a menudo se olvidan en la prisa por implementar un desarrollo ágil. Los evaluadores a menudo no reciben ninguna formación especial. También pueden quedar atrapados en una situación de minicascada, donde las historias no se entregan para la prueba hasta el último día de la iteración, de modo que las pruebas siempre son para ponerse al día.

Cuando nos concentramos en ofrecer nuevas funciones, es difícil dar un paso atrás y tomarnos el tiempo para aprender. Sin embargo, si no dedicamos tiempo al aprendizaje, es posible que no podamos mantener contentos a nuestros clientes.

Si ocupa un puesto directivo o de liderazgo, tiene la oportunidad de fomentar una cultura de aprendizaje. Independientemente de su función, intente que su equipo analice áreas de pruebas en las que el equipo carece de experiencia y piense en maneras en que cada miembro del equipo pueda desarrollar sus “habilidades en forma de T” (como se describe en el Capítulo 3, “Funciones y competencias”). ”) para cubrir las capacidades faltantes.



Figura 25-4 Aprendizaje

Sensibilidad al contexto

Tenga en cuenta que es posible que necesite aplicar principios ágiles de manera diferente según su contexto. Esta es una de las áreas principales que exploramos en este libro, aunque estamos seguros de que no abordamos todas las diferentes situaciones que existen; Cada día surgen nuevos desafíos. Comprenda su contexto (consulte la Figura 25-5) y esfuérzese por aplicar principios y prácticas ágiles a su entorno. (Nota: utilizamos la palabra contexto según su definición de Merriam-Webster como "la situación en la que algo sucede: el grupo de condiciones que existen donde y cuando algo sucede".

No estamos hablando de "pruebas basadas en el contexto" (Kaner, 2012), aunque es posible que también quieras aprender sobre eso).

Por ejemplo, si trabaja en una gran corporación, es posible que deba adaptar las pruebas a los controles organizacionales existentes. Es posible que necesite roles adicionales para ayudar a coordinar las actividades de prueba y administrar las dependencias en toda la organización, o tal vez incluso tener equipos adicionales para actividades como pruebas en todo el sistema que abarquen múltiples aplicaciones.

Un mayor apoyo del nivel ejecutivo puede ayudar a efectuar cambios culturales para cambiar la mentalidad de prueba de la detección de errores a la prevención de errores.

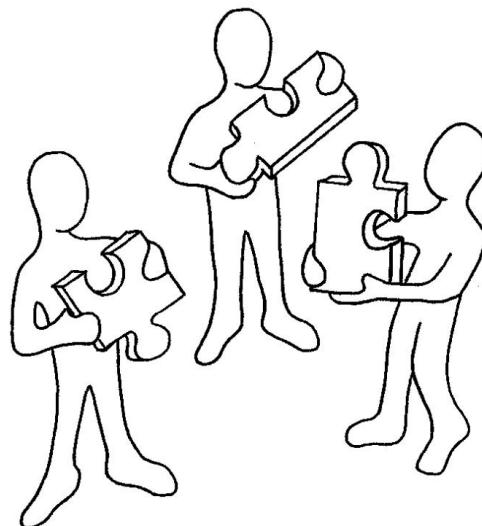


Figura 25-5 Conozca su contexto.

Algunos equipos de software deben cumplir requisitos reglamentarios para documentación específica. Si los equipos piensan en la forma más sencilla de proporcionar la evidencia necesaria requerida por los auditores, es posible que puedan reducir al mínimo tanto la cantidad de documentación como el tiempo dedicado a ella. Puede parecer contradictorio al principio, pero los valores y prácticas de prueba ágiles funcionan bien en este contexto para cumplir con las regulaciones y reducir el riesgo para el producto.

Los equipos distribuidos requieren enfoques innovadores para permitir una colaboración y comunicación adecuadas en su contexto. Es difícil para un equipo homenajeado para vincularse si los miembros del equipo nunca tienen la oportunidad de reunirse en persona o hablar directamente entre ellos. Si su equipo está disperso geográficamente, busque formas creativas de generar confianza y permitir la colaboración para que pueda beneficiarse de un enfoque de prueba de todo el equipo. El uso de pruebas para ayudar a definir las especificaciones es una forma de obtener una mejor comprensión compartida de qué construir.

Los equipos de software móvil e integrado enfrentan desafíos de prueba adicionales debido a la cantidad de plataformas y dispositivos compatibles y muchas otras variables, como cómo y dónde se utilizarán los dispositivos. Estos equipos pueden mejorar la calidad del software mediante la colaboración entre roles, aprovechando la automatización y utilizando ciclos de retroalimentación rápidos.

Los equipos de inteligencia empresarial y almacén de datos tienen la tarea de garantizar que los expertos empresariales dispongan de datos de alta calidad para planificar y tomar decisiones empresariales. Puede resultar difícil descubrir cómo construir estos sistemas de forma incremental e iterativa. Es posible que se necesiten habilidades técnicas especializadas y experiencia en el dominio empresarial para tener éxito con las pruebas ágiles en este contexto.

Se han incorporado a los equipos de entrega nuevos conjuntos de habilidades especializadas que se centran en las pruebas y la calidad desde un punto de vista operativo. Todo el equipo participa en actividades de DevOps para agilizar la entrega, acortar los ciclos de retroalimentación, mejorar la efectividad de las pruebas automatizadas y garantizar que las infraestructuras de CI, pruebas e implementación admitan tiempos de ciclo cortos.

Nuestro producto de software es más que solo la aplicación bajo prueba, y DevOps ayuda a ampliar nuestras pruebas para abarcar todas las facetas del producto y el valor que brinda a los clientes. Cualquiera que sea el contexto de su equipo, DevOps puede ayudarlo a mejorar continuamente la calidad del software.

Se realista

En algún momento todos aspiramos a tener superpoderes, ¡y nuestros gerentes a menudo parecen pensar que los poseemos! Pero como nos recuerda Bob Martin en Clean Coder (Martin, 2011), la mejor manera de trabajar en equipo es informar a la gerencia lo que podemos lograr de manera realista y ayudarlos a priorizar el trabajo de mayor valor.

Incluso si su equipo no utiliza un proceso basado en flujo, el concepto de limitar el trabajo en progreso (WIP) es esencial. Cada uno de nosotros tiene una cantidad limitada de ancho de banda. Si los evaluadores del equipo pueden probar sólo un máximo de dos historias de usuario a la vez, no tiene sentido que los programadores sigan creando historias. Si los programadores detienen los nuevos desarrollos y ayudan con las pruebas, el equipo puede avanzar mucho más fácilmente.

Haga que las pruebas formen parte de cada historia y de cada característica.

No sucumba al deseo de hacer felices a los clientes diciendo: "Lo intentaremos" cuando sabe que es imposible. ¡Se necesita coraje para decir que no! Experimente formas de hacer visible el límite de WIP de la prueba. En lugar de encogerse de hombros y decir: "Tenemos un cuello de botella en las pruebas", busque formas de eliminar la restricción, como hacer que todo el equipo ayude con las actividades de prueba para mantener los elementos de trabajo avanzando a un ritmo constante. Ayude a su equipo a comprobar la realidad cuando sea necesario (consulte la Figura 25-6).

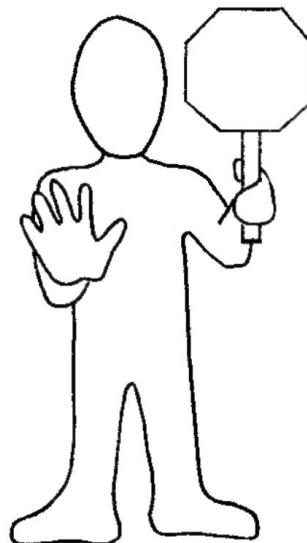


Figura 25-6 Verificación de la realidad

Crear una visión compartida

Su propio equipo, su entorno, sus limitaciones y su visión compartida son únicos.

No existe un conjunto mágico de prácticas de pruebas ágiles que garanticen el éxito. Sin embargo, los valores y principios ágiles nos guían. Probar pequeños experimentos y crear ciclos cortos de retroalimentación nos ayudan a mantener el rumbo y hacer correcciones en el camino.

Hemos descrito algunas prácticas de prueba básicas que, según nuestra experiencia, son valiosas para la mayoría de los equipos. Nos resulta útil dar un paso atrás y recordar el propósito de las pruebas.

Pilares de las pruebas

David Evans, un entrenador ágil de calidad del Reino Unido, nos recuerda sobre las pruebas [y el propósito](#). resume sus "Pilares de las pruebas" quedan maravillosamente en

Las pruebas no producen calidad; produce información sobre la calidad. A alto nivel, el resultado que queremos de las pruebas es confianza. en nuestras decisiones de publicar cambios de software. Mejores pruebas producen mayor confianza en esas decisiones. Por lo tanto, el valioso producto de las pruebas efectivas es la confianza.

En mi modelo de "Pilares de las pruebas" (consulte la Figura 25-7), trazo las conexiones y dependencias de una serie de factores que son críticos para aumentando esta confianza. Represento el modelo como un "templo", con una techo, cuatro pilares, cimientos y base de roca.

En la cima de nuestro modelo, la confianza se basa en el coraje y la seguridad. El coraje sin seguridad conduce a la temeridad; Seguridad sin coraje conduce al estancamiento y la inacción.

Los cuatro pilares que aumentan nuestro nivel de confianza en las pruebas son

- Evidencia de prueba más sólida: ¿Qué tan bien entienden nuestras partes interesadas lo que se ha probado y cuándo?
- Mejor diseño de pruebas: qué tan apropiado, significativo y efectivo Cuáles son los casos de prueba que utilizamos?
- Mayor cobertura de pruebas: ¿Qué parte del sistema se prueba en comparación con nuestros modelos de riesgo?

- Comentarios de prueba más rápidos: ¿Con qué rapidez identificamos problemas y sus acciones correctivas?

Cada pilar representa cualidades que existen en una escala; todas existen hasta cierto punto, pero siempre se pueden mejorar y elevar. Cuantas más de estas cualidades tengamos, más confianza crearemos. También debemos ser conscientes del equilibrio y evitar invertir excesivamente en cualquier pilar en detrimento de otros.

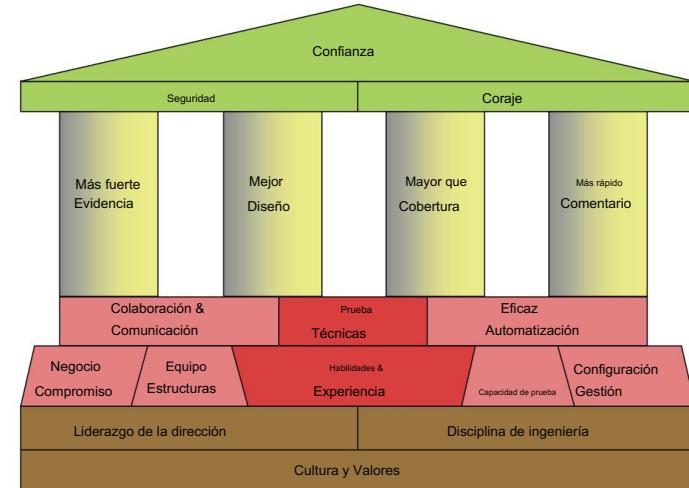


Figura 25-7 Pilares de las pruebas

Los pilares respaldan los fundamentos, que representan tres áreas clave: fundamentos de equipo, fundamentos de capacidad y fundamentos técnicos. Cada área incluye varios bloques de construcción interconectados:

- Fundamentos del equipo
 - Colaboración y comunicación: representa el grado en que un equipo es capaz de aprovechar al máximo una variedad de perspectivas y habilidades, tanto dentro como fuera del equipo, y encontrar los modos de comunicación más efectivos e informativos.
 - entre todos los interesados.

- Compromiso empresarial: la colaboración eficaz depende de un compromiso significativo y productivo entre los equipos técnicos y los empresarios a los que apoyan.
 - Estructuras de equipo: la colaboración también depende de garantizar que existan las estructuras de equipo más apropiadas para respaldar el trabajo en equipo efectivo, conectando a las personas con las habilidades adecuadas y capacitándolas para trabajar como equipos autoorganizados y multifuncionales.
- Fundamentos de capacidad
- Técnicas de prueba: aplicación de pruebas apropiadas y efectivas.
Las técnicas, tanto en el diseño de casos de prueba formales como en la exploración dinámica del producto, contribuirán a un mejor diseño de pruebas y una mayor cobertura.
 - Habilidades y experiencia: la efectividad de esas técnicas de prueba (y otros fundamentos) depende en gran medida de las habilidades y experiencia dentro del equipo.
- Fundamentos Técnicos
- Automatización efectiva: la capacidad de realizar rápida y eficientemente
Automatizar la mayoría de las pruebas definidas es fundamental para respaldar los pilares de una retroalimentación más rápida y una mayor cobertura.
Esto, a su vez, depende de la capacidad de prueba de la arquitectura y de una sólida gestión de la configuración.
 - Gestión de configuración: la disciplina de gestionar versiones.
siones de pruebas y otros códigos, la configuración controlada de entornos y datos de prueba, y la identificación de todos los artefactos relevantes en el proceso de desarrollo y prueba de software.
 - Capacidad de prueba: el grado en que la arquitectura del sistema
El sistema bajo prueba está diseñado para admitir pruebas automatizadas.
Las características de un sistema comprobable se alinean con buenos objetivos arquitectónicos, como un acoplamiento flexible, inyección de dependencia, separación estricta de preocupaciones, un modelo de dominio bien definido y uso ubicuo del lenguaje de dominio.

Todas las áreas de la Fundación se basan en la base organizacional, que representa el estilo de liderazgo y la cultura de la organización:

- Gestión y liderazgo: la capacidad de contratar e inspirar a personas excelentes, eliminar o superar limitaciones e impedimentos organizacionales que les impiden hacer su mejor trabajo, ver y explotar las oportunidades para que las personas sobresalgan y motivar a las personas para que alcancen logros. lo que es mejor para el equipo.

- Disciplina de Ingeniería: La orientación y el apoyo brindado a Garantizar que los equipos técnicos tengan las herramientas y conocimientos técnicos adecuados. medio ambiente a su disposición, están facultados para aprovechar al máximo de ellos, y se les brinda la educación y el desarrollo personal para garantizar que sus habilidades estén siempre actualizadas y sean apropiadas.
- Cultura y Valores: En definitiva, todo el entorno y La estructura para el éxito se basa en la cultura y los valores de la organización.

Las pruebas forman una parte importante de la capacidad de cualquier equipo para entregar valor comercial con frecuencia a un ritmo sostenible. Sabemos que no podemos probar la calidad; tenemos que incorporarlo desde el principio. Cuando comenzamos el desarrollo de software con pruebas, es más probable que generemos el valor adecuado para nuestro cliente, y cuando completamos el desarrollo y la entrega con pruebas, ayudamos a garantizar que hayamos creado ese valor correctamente.

Resumen

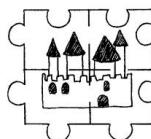
Alentamos a las personas a recordar los siete factores clave de éxito para las pruebas ágiles: utilizar el enfoque de todo el equipo, mantener una mentalidad de prueba ágil, automatizar las pruebas de regresión, centrarse en la retroalimentación, utilizar prácticas básicas, colaborar y tener en mente el panorama general. . Con el espíritu de aprendizaje continuo, animamos a los equipos a experimentar con las prácticas de fomento de la confianza que analizamos en este libro:

- Impulsar el desarrollo con ejemplos



- Pruebas exploratorias





■ Pruebas de funciones



■ Aprendizaje continuo



■ Sensibilidad al contexto



■ Manteniéndolo real

Esfuérzese por lograr confianza aplicando los pilares de las pruebas. Es lo mejor que podemos hacer por nuestros clientes.

Apéndice A

Objetos de página en la práctica: Ejemplos

En el Capítulo 16, "Patrones y enfoques de diseño de automatización de pruebas", Tony Sweets explicó los conceptos detrás del uso del patrón de objetos de página para diseñar pruebas automatizadas. Aquí hay fragmentos de código y detalles técnicos para que pueda probarlo usted mismo.

Un ejemplo con Selenium 2: WebDriver

El siguiente fragmento de código es un objeto de página para la página de inicio de Wikipedia. Está escrito en Java usando Selenium 2 (WebDriver). Este objeto de página en particular tiene dos métodos públicos (o acciones) que puede realizar. Podemos hacer clic en el enlace de privacidad o podemos buscar en Wikipedia utilizando el término "Nissan Motor Company". Estos dos métodos (más el método inicializador, que se conoce como constructor) forman la interfaz pública de este objeto de página. Esta es la interfaz que desea mantener lo más estable posible y es lo que utilizará su script de prueba para navegar por el sitio y realizar pruebas.

Si cree que carece de las habilidades de programación necesarias para implementar este objeto, los programadores del equipo podrían escribir el código para los evaluadores y dejar los scripts de prueba para los miembros del equipo que realmente están especificando las pruebas.

Una cosa que debes notar acerca de este código es que encapsulamos los elementos de la página y los hicimos privados. Esto ayuda a evitar pruebas frágiles, ya que no tienes acceso directo a ellas y son las que tienen más probabilidades de cambiar con el paso del tiempo. La única manera de acceder a ellos es

a través de la interfaz pública, que es intencionalmente vaga. Cuando su script de prueba llama a `searchForNissan()`, no tiene idea de cómo el objeto de página realizará ese trabajo, ni debería tenerla. Dado que ocultamos los detalles de implementación de la prueba real, podemos cambiar la implementación sin cambiar la prueba.

Por ejemplo, digamos que Wikipedia realizó una actualización del sistema y la búsqueda de "Nissan Motor Company" ya no arroja la página real de Nissan. En su lugar, devuelve una página bloqueadora, con una imagen del fundador Jimmy Wales pidiendo una donación, y esa página tiene un enlace a la página real de Nissan. Sin cambiar tu prueba, puedes modificar la página Nissan Objeto de página para hacer clic en el enlace debajo de la solicitud de donación antes de devolver el objeto `NissanPage` a la prueba.

```
// Esta es la clase de página de inicio que cuando se crea (cuando // se ejecuta la prueba) se
// convierte en un objeto de página
Página de inicio de clase pública {
    // Esta es una utilidad de registro. Se utiliza para iniciar sesión en un archivo // y/o la
    // pantalla
    Registro de registrador final estático privado = LoggerFactory.
    getLogger(Página de inicio.clase);

    // Este es el objeto que actúa como navegador web cuando // se usa la unidad HTML

    // o lo que "impulsa" un navegador web real como
    // Firefox
    WebDriver privado WebDriver;

    // Estos son elementos (enlaces, cuadros de entrada, etc.) en // esta página que
    // necesito usar
    enlace de política de privacidad de WebElement privado;
    cuadro de búsqueda de WebElement privado;
    botón de envío de elemento web privado;

    // Constructor de este objeto de página
    Página de inicio pública (WebDriver webDriver) {
        log.info("Página = {}", webDriver.getTitle());
        log.info("URL = {}", webDriver.getCurrentUrl());
        this.webDriver = webDriver;

        // Probar que la página en la que estamos es en realidad // la página de
        // inicio de Wikipedia.
        // Sabemos que estamos en la página de inicio si // el título coincide
        // con el single
        // palabra "Wikipedia" y nada más
        if (!webDriver.getTitle().equals("Wikipedia")) {
            throw new IllegalStateException("Página de inicio solicitada: página
actual - webDriver.getTitle()");
        }
    }
}
```

```
//Inicializar elementos web
enlace de política de privacidad =
webDriver.findElement(By.partialLinkText("Política de privacidad"));
    searchBox = webDriver.findElement(Por.
id("entrada de búsqueda"));
        enviarBotón = webDriver.findElement(Por.
nombre("ir"));
    }

página de política de privacidad pública haga clic en enlace de política de privacidad () {
    privacidadPolicyLink.click();

    devolver nueva PrivacyPolicyPage(webDriver);
}

Búsqueda pública de NissanPageForNissan() {
    searchBox.sendKeys("Nissan Motor Company");
    enviarBotón.enviar();

    devolver nueva NissanPage(webDriver);
}
}
```

Veamos un script de prueba (uso ese término libremente, ya que se trata de un programa/prueba Java real). Este es código Java y está implementado como JUnit. El uso del marco JUnit nos permite aprovechar todos los complementos y el ecosistema aptos para pruebas de forma gratuita. Por ejemplo, la salida de JUnit se podrá mostrar mediante sistemas de compilación continua, de forma predeterminada, sin necesidad de trabajo adicional. También puedes ejecutar esto con un simple comando de Maven como cualquier otra prueba.

Considero que esto es un guión, porque es un diseño de arriba hacia abajo. En realidad no está orientado a objetos, pero utiliza objetos de página que usted ha creado para representar su sistema bajo prueba. Debe crear un objeto WebDriver que sea parte de Selenium 2, inicializarlo con el navegador web que va a controlar (HtmlUnit en este caso) y apuntarlo a una URL. Luego simplemente se lo entrega al objeto de página que representa esa URL específica, en este caso la página de inicio . También estoy registrando el proceso porque si su prueba falla, querrá saber en qué paso se encontraba para poder investigar manualmente lo que sucedió.

```
@Prueba
prueba pública vacíaWikipedia() {
    WebDriver webDriver = nuevo HtmlUnitDriver();

    log.info("Solicitando página de inicio");
    webDriver.get("http://www.wikipedia.org");
    Página de inicio Página de inicio = nueva Página de inicio (webDriver);
    Assert.assertNotNull(página de inicio);
```

```
    log.info("Solicitando política de privacidad");
    PrivacyPolicyPageprivacyPolicyPage = página de inicio.
    haga clic en EnlacePolíticaDePrivacidad();
    Assert.assertNotNull(privacyPolicyPage);

    // Si vuelvo debería estar en la página de inicio nuevamente
    log.info("Presionando el botón Atrás");
    webDriver.navigate().back();
    Página de inicio homePageAfterBack = nueva
    página de inicio (webDriver);
    Assert.assertNotNull(homePageAfterBack);

    // Búsqueda de Nissan
    log.info("Buscando Nissan Motor Company");
    NissanPage nissanPage = homePageAfterBack.
    buscarNissan();
    Assert.assertNotNull(nissanPage);

    // Ir a la lista de autos Nissan
    log.info("Haciendo clic en la lista de automóviles Nissan");
    NissanCarListPage nissanCarListPage = nissanPage.
    seleccionarListaDeCoches();
    Assert.assertNotNull(nissanCarListPage);

    // Finalmente ve a la página del Nissan Skyline GT-R
    log.info("Haciendo clic en Skyline GT-R");
    NissanSkylinePágina nissanSkylinePágina =
    nissanCarListPage.selectNissanSkyline();
    Assert.assertNotNull(nissanSkylinePágina);

    webDriver.quit();
}
```

JUnit usa la clase `Assert` para verificar aspectos de su código. Puede comprobar si hay nulos, igualdad y verdadero/falso, entre otros. Si la aserción falla, el script se detiene en ese punto y se registra automáticamente como falla de prueba. Esto puede aparecer en su sistema de compilación, en la línea de comando o en un informe. Al final del script de prueba, enviamos un comando de salida al objeto `WebDriver`, que limpia cualquier recurso que se esté utilizando. Esto es especialmente útil cuando se utiliza un navegador real como Internet Explorer o Firefox.

Usando la clase PageFactory

Lo que ya hemos visto son simplemente objetos de página implementados con Selenium 2. Podríamos ir un paso más allá con Selenium 2 y usar la clase auxiliar `PageFactory` incorporada que proporciona. Esta es una implementación de un patrón de diseño ampliamente utilizado conocido como patrón Factory.

PageFactory le ayuda ofreciéndole anotaciones fáciles de leer para localizar elementos de la página web . Las anotaciones son una característica relativamente nueva de Java y siempre comienzan con el símbolo (@). La fábrica de páginas crea el objeto de página por usted e inicializa los elementos en función de las anotaciones que escribió para ubicar el elemento. Aquí está nuevamente el objeto HomePage con la implementación de PageFactory :

```
Página de inicio de clase pública {  
    Registro de registrador final estático privado = LoggerFactory.  
getLogger(Página de inicio.clase);  
    WebDriver privado WebDriver;  
  
    @FindBy(how = How.PARTIAL_LINK_TEXT, usando = "Política de privacidad")  
    enlace de política de privacidad de WebElement privado;  
    @FindBy(how = How.ID, usando = "searchInput")  
    cuadro de búsqueda de WebElement privado;  
    @FindBy(cómo = Cómo.NOMBRE, usando = "ir")  
    botón de envío de elemento web privado;  
  
    Página de inicio pública (WebDriver webDriver) {  
        log.info("Página = {}", webDriver.getTitle());  
        log.info("URL = {}", webDriver.getCurrentUrl());  
        this.webDriver = webDriver;  
        if (!webDriver.getTitle().equals("Wikipedia")) {  
            throw new IllegalStateException("Página de inicio solicitada: página  
actual - + webDriver.getTitle());  
        }  
  
    }  
  
    página de política de privacidad pública haga clic en enlace de política de privacidad () {  
        privacidadPolicyLink.click();  
        devolver PageFactory.initElements(webDriver, PrivacyPolicyPage.class);  
    }  
  
    Búsqueda pública de NissanPageForNissan() {  
        searchBox.sendKeys("Nissan Motor Company");  
        enviarBotón.enviar();  
        devolver PageFactory.initElements(webDriver, NissanPage.class);  
    }  
}
```

Tenga en cuenta que no ha cambiado mucho, pero es suficiente como para que probablemente no desee mezclar los dos estilos, ya que sus scripts de prueba también son ligeramente diferentes. Su script de prueba ya nunca crea un objeto de página por sí solo. Delega en la fábrica. No estás comprando mucho con

este cambio; sin embargo, es un poco más limpio a la vista y las anotaciones le facilitan la escritura y administración del objeto de página.

El guión de prueba también debe cambiar. Cuando creamos el objeto de página nosotros mismos, se veía así:

```
log.info("Solicitando página de inicio");
webDriver.get("http://www.wikipedia.org");
Página de inicio Página de inicio = nueva Página de inicio (webDriver);
Assert.assertNotNull(página de inicio);
```

Ahora, con PageFactory creando el objeto, se ve así (los cambios están resaltados):

```
log.info("Solicitando página de inicio");
webDriver.get("http://www.wikipedia.org");
Página de inicio página de inicio = PageFactory.
initElements(webDriver, HomePage.clase);
Assert.assertNotNull(página de inicio);
```

El código completo de este ejemplo se encuentra en mi cuenta de GitHub (Sweets, 2013). El archivo README tiene toda la información necesaria para ejecutar este código en su caja local.

Si está en una Mac con las herramientas de desarrollo instaladas, debería poder ejecutar estos comandos para descargar y ejecutar este conjunto de pruebas:

```
> cd/tmp
> clon de git https://github.com/tsweets/page-objects.git
> cd página-objetos/
> prueba de limpieza mvn
```

Esto funcionará en cualquier plataforma siempre que tenga instalado Java, Git y Maven. El directorio temporal es específico del sistema, sin embargo, con /tmp siendo parte de Mac y otras cajas basadas en UNIX/Linux. Esto ejecutará tres versiones de la misma prueba. Ejecutará una versión frágil, la versión de objeto de página y la versión de fábrica de páginas.

apéndice B

Entrantes de provocación

Carol Vaage utilizó una lista de preguntas para que sus alumnos de primer grado comenzaran a pensar en el mundo y en cómo interactuaban con él. Estas son preguntas que pueden usarse para pensar sobre su producto y cómo puede usarlas para obtener ejemplos o descubrir suposiciones. Consulte la bibliografía de la Parte II, “Aprender para realizar mejores pruebas”, para obtener el enlace.

Me pregunto . . .

Podrías mostrarme . . .

Averigüemos cómo podría ser eso. . .

En otras palabras, si usted. . . entonces . . . ?

¿Qué pasaría después?

¿Pero por qué sería eso?

Entonces, ahora nuestra pregunta es. . .

¿Te diste cuenta?

¿Qué has descubierto? . . . ?

Sé que puedes resolver esto: trabaja con tu equipo y ve qué ideas podrían funcionar.

Esta (persona/grupo) se ha topado con un problema. ¿Qué se nos ocurre para ayudarles a solucionarlo?

¿Por qué cree que pasó?

Describe lo que ves.

¿Qué te parece eso?

¿Cómo sucede eso?

¿Por qué funcionó de esa manera?

¿Cómo puedes saberlo?

¿En qué se diferencia esto?

¿Qué piensas tú que sucederá?

¿Cómo puedes saberlo?

Pruébalo y cuéntame cómo resulta.

¿Puedes mostrarme la evidencia?

¿Cuál es tu teoría?

Glosario

Prueba de aceptación Las pruebas de aceptación son pruebas que definen el valor comercial que cada historia (o característica) debe ofrecer y ayudan a definir el alcance de la historia (o característica). Pueden verificar requisitos funcionales o requisitos no funcionales, como el rendimiento o la confiabilidad. Aunque se utilizan para ayudar a guiar el desarrollo, se realizan a un nivel superior a las pruebas de nivel unitario utilizadas para el diseño de código en el desarrollo basado en pruebas. Este es un término amplio que puede incluir pruebas tanto orientadas a los negocios como a las tecnológicas.

Desarrollo basado en pruebas de aceptación (ATDD) Los equipos que realizan ATDD especifican pruebas de aceptación ejecutables utilizando ejemplos obtenidos de conversaciones con clientes. Cada ejemplo se analiza y, si es posible, se especifica en un formato ejecutable. Los programadores utilizan las pruebas como guía para desarrollar la función. Una vez que se escribe el código para pasar la prueba, se puede mantener como una prueba de regresión automatizada, documentando el comportamiento de las funciones.

Gestión del ciclo de vida de la aplicación (ALM) ALM se refiere a la gestión del ciclo de vida de un producto de aplicación por parte de personas de múltiples disciplinas que trabajan en las diversas actividades involucradas en la entrega del producto. Agile ALM abarca actividades como la integración y entrega continua, la colaboración con los clientes para describir ejemplos que guían el desarrollo, la codificación, las pruebas y la gestión de lanzamientos.

Desarrollo impulsado por el comportamiento (BDD) Los ejemplos de comportamiento deseado de las conversaciones con las partes interesadas se convierten en pruebas ejecutables en un formato determinado_cuándo_entonces. Los programadores automatizan las pruebas y

luego escriba el código para pasar cada prueba. BDD se puede utilizar a nivel de unidad para el diseño basado en pruebas y se puede utilizar para pruebas empresariales que abarcan múltiples componentes y capas de la aplicación y verifican la calidad empresarial.

Construir-medir-aprender (BML) Construir-medir-aprender es un término de lean startup en el que un equipo convierte una idea en un producto entregable, mide la respuesta del cliente y aprende si la idea debe llevarse a cabo tal como está y cambiarse de alguna manera. , o abandonado.

Canal de compilación Cuando el código se registra en el sistema de control de código fuente, la confirmación inicia un canal de compilación. Para equipos más maduros, esto significa ejecutar un conjunto rápido de pruebas de regresión a nivel unitario para proporcionar retroalimentación rápida, seguido de una serie de conjuntos de pruebas en los niveles de API y de interfaz de usuario para detectar fallas de regresión. Las canalizaciones de compilación se pueden configurar para implementar automáticamente artefactos de compilación en entornos de prueba, ensayo y producción.

Integración continua (CI) En equipos que utilizan la práctica de integración continua, los miembros del equipo verifican los cambios de código con frecuencia a lo largo del día. Cada registro desencadena un proceso de compilación automatizado en el que las pruebas de regresión automatizadas brindan retroalimentación rápida para verificar si los cambios en el código han introducido fallas de regresión.

Estas compilaciones producen artefactos que pueden implementarse automática o manualmente en diferentes entornos, como prueba, preparación y producción.

Equipo de clientes Los equipos de clientes incluyen a todas las partes interesadas fuera de el equipo de entrega de software, como propietarios de productos, gerentes de productos, expertos comerciales, expertos en la materia y usuarios finales.

Tiempo de ciclo En sentido genérico, es el tiempo que transcurre desde que algo empieza hasta que termina. Específicamente, nos referimos al tiempo transcurrido desde que se inicia una historia hasta que es aceptada por el propietario del producto o las partes interesadas. A menudo también significa el tiempo desde que existe una historia por primera vez hasta el momento en que se entrega a los usuarios finales.

Equipo de entrega, equipo de desarrollo El equipo de entrega o desarrollo entrega el software que cumple con los objetivos comerciales y proporciona valor.

A los consumidores. Todos los involucrados en la entrega de software son desarrolladores, incluidos programadores, evaluadores, analistas de negocios, expertos en bases de datos, administradores de sistemas, redactores técnicos, arquitectos, expertos en diseño de experiencias de usuario y personal de operaciones.

DevOps DevOps, un término que combina desarrollo y operaciones, se refiere a prácticas y patrones que mejoran la colaboración entre diferentes roles en los equipos de entrega, simplificando el proceso de entrega de software de alta calidad. Estas prácticas y patrones ayudan a los equipos a escribir código comprobable y mantenible, empaquetar el producto, implementarlo y respaldar el código implementado.

Fin del juego El final del juego es el momento antes del lanzamiento cuando el equipo de entrega aplica los toques finales al producto. No es un período de corrección de errores o "endurecimiento", es una oportunidad para trabajar con grupos fuera del desarrollo para ayudar a llevar el software a producción. Ejemplos de actividades finales del juego incluyen pruebas adicionales de migraciones e instalación de bases de datos.

Soluciones empresariales Software utilizado para satisfacer las necesidades de una organización en lugar de las de usuarios individuales, que se encuentran principalmente en grandes organizaciones. El software está destinado a resolver un problema de toda la empresa, en lugar de un problema departamental. El software de nivel empresarial tiene como objetivo mejorar la productividad y la eficiencia de la empresa proporcionando funcionalidad de soporte de lógica de negocios (Wikipedia, 2014d).

Pruebas exploratorias (ET) Las pruebas exploratorias combinan el diseño de pruebas con la ejecución de pruebas y se centran en aprender sobre la aplicación bajo prueba. Consulte el Capítulo 12, "Pruebas exploratorias", para obtener una guía detallada.

Extraer, transformar y cargar (ETL) Extraer, transformar y cargar se refiere a un proceso en el uso de bases de datos, especialmente para almacenes de datos, que extrae datos de fuentes externas, los transforma para satisfacer las necesidades operativas y los carga en una base de datos de destino (Wikipedia, 2014e).

Datos federados La tecnología de federación de datos es un software que brinda a una organización la capacidad de agregar datos de fuentes dispares en una base de datos virtual para que pueda usarse para inteligencia empresarial u otros análisis (Search Data Management, 2010).

Heurística La heurística se refiere a técnicas basadas en la experiencia para la resolución de problemas, el aprendizaje y el descubrimiento que brindan una solución que no es válida.

Garantizado que será óptimo. Cuando una búsqueda exhaustiva no es práctica, Los métodos heurísticos se utilizan para acelerar el proceso de encontrar una solución satisfactoria a través de atajos mentales para aliviar la carga cognitiva de tomar decisiones. una decisión. Ejemplos de este método incluyen el uso de una regla general, una conjetura fundamentada, juicio intuitivo, estereotipos o sentido común (Wikipedia, 2014g).

Infraestructura como servicio (IaaS) La infraestructura como servicio es un Modelo de provisión en el que una organización subcontrata el equipo. Se utiliza para respaldar operaciones, incluido almacenamiento, hardware, servidores y componentes de red. El proveedor de servicios es propietario del equipo y es responsable de albergarlo, administrarlo y mantenerlo. El cliente normalmente paga por uso (SearchCloudComputing, 2010a).

Kanban El término kanban es japonés y tiene el significado literal de “tarjeta” o “letrero”. En el Sistema de Producción Toyota, designa un sistema utilizado para el control de inventarios “justo a tiempo” (Agile Alliance, 2012). En el desarrollo ágil de software, el “método kanban” se caracteriza por límites al trabajo en progreso, la idea de “tirar” del trabajo en lugar de tenerlo asignado por gerentes, y un equipo autodirigido que gestiona sus propios carga de trabajo sostenible. Los tableros Kanban se utilizan para visualizar proyectos y ayudar a hacer cumplir los límites del trabajo en progreso (WIP) (Hiranabe, 2008).

Sistema heredado Un sistema heredado es aquel que no es compatible con pruebas de regresión automatizadas. Introducir cambios en el código heredado o refactorizarlo puede ser arriesgado porque no existen pruebas para detectar errores no deseados. cambios en el comportamiento del sistema.

Producto mínimo viable (MVP) El término producto mínimo viable Fue acuñado por Frank Robinson y popularizado por Eric Ries para productos de software (Wikipedia, 2014i). Un producto mínimo viable acaba de suficientes características principales para permitir que el producto se implemente en una pequeña subconjunto de clientes potenciales. Estos usuarios forman parte del ciclo de retroalimentación “construir-medir-aprender” o “aprendizaje validado”, lo que guía aún más desarrollo y toma de decisiones. Un MVP también se utiliza para representar la menor cantidad de cambio que se puede entregar al cliente que añade valor empresarial.

Pruebas de aceptación operativa (OAT) La OAT se utiliza para llevar a cabo la preparación operativa (prelanzamiento) de un producto, servicio o sistema como parte de un sistema de gestión de calidad. OAT es un tipo común de no funcional pruebas de software, utilizadas principalmente en proyectos de soporte y mantenimiento de software.

Plataforma como servicio (PaaS) Una consecuencia del software como servicio, PaaS es un modelo de prestación de servicios que permite al cliente alquilar servidores virtualizados y servicios asociados para ejecutar aplicaciones existentes. o desarrollar y probar otros nuevos (SearchCloudComputing, 2010).

Lanzamiento del producto El lanzamiento de un producto de software puede consistir en un producto nuevo o nuevas características para un producto existente. En el desarrollo ágil, una función puede consistir en una o más historias de usuario completas.

Refactorización Refactorización es cambiar el código, sin cambiar su funcionalidad, para hacerlo más fácil de mantener, más fácil de leer, más fácil de probar. y más fácil de ampliar.

Prueba de regresión Una prueba de regresión verifica que el comportamiento del sistema bajo prueba no ha cambiado. Las pruebas de regresión generalmente se escriben como pruebas unitarias para impulsar la codificación o pruebas de aceptación para definir el comportamiento deseado del sistema. Una vez que las pruebas pasan, pasan a formar parte de una prueba de regresión. suite, para protegerse contra la introducción de cambios no deseados. Las pruebas de regresión deben automatizarse para garantizar una retroalimentación continua.

Candidato de lanzamiento Un candidato de lanzamiento es una versión o compilación de un producto que potencialmente puede lanzarse a producción. El candidato de liberación puede someterse a pruebas adicionales o complementarse con documentación. u otros materiales.

Retorno de la inversión (ROI) Retorno de la inversión, un término prestado del mundo de las inversiones financieras, es una medida de la eficiencia de una inversión. El ROI se puede calcular de diferentes maneras, pero básicamente es la diferencia entre la ganancia de una inversión y el costo. de esa inversión, dividido por el costo de esa inversión. En las pruebas, El ROI es el beneficio obtenido de una actividad de prueba, como la automatización de un prueba, ponderada con el costo de producir y mantener esa prueba.

Desarrollo basado en conjuntos Tal como se aplica al desarrollo de software, en el desarrollo basado en conjuntos, dos pares o equipos de desarrollo de forma independiente idear una solución o característica al problema. Las partes interesadas, ya sea la empresa u otros miembros del equipo de desarrollo, evalúan los resultados y elegir el que prefieran.

Software como servicio (SaaS) En el modelo de software como servicio, las aplicaciones son alojadas por un proveedor o proveedor de servicios. En lugar de instalar el software en su propio hardware, como lo harían con el "software como producto", los clientes acceden al software en los servidores del proveedor a través de un red, normalmente Internet (SearchCloudComputing, 2010c).

Sistema de control de código fuente También conocido como control de versiones, control de revisión, o gestión de código fuente, control de código fuente Los sistemas permiten que varios miembros del equipo trabajen en la misma base de código. documentos u otro tipo de archivos sin pisarse unos a otros. actualizaciones. Las características comunes incluyen fusionar cambios en el mismo archivo, revertir cambios, comparar versiones, ver un historial de cambios, y recuperar versiones específicas. Hoy en día, los servicios de alojamiento para distribuidos repositorios de código como GitHub, que utiliza Git como control de revisión sistema, son populares (Wikipedia, 2014j).

Especificación por ejemplo (SBE) Especificación por usos de ejemplo Patrones de proceso que ayudan a los equipos de entrega de software a colaborar. equipos de clientes para definir el alcance del trabajo para lograr los objetivos comerciales. SBE ilustra las especificaciones con ejemplos concretos, refina las especificaciones a partir de ejemplos clave y luego las automatiza para validar las especificaciones con frecuencia durante el desarrollo. El ejecutable validado. las especificaciones se mantienen como documentación viva (Adzic, 2011).

Deuda técnica Ward Cunningham fue el primero en introducir esta metáfora, aplicando el concepto de deuda financiera al software. cuando el software Los equipos toman algunos atajos y sacrifican la calidad para ayudar a la empresa a aprovechar una oportunidad, incurren en deudas en forma de software que no está bien diseñado, no está protegido por pruebas de regresión automatizadas o carece de otras características de calidad. Si el equipo no Si se toma tiempo para "pagar" la deuda refactorizando el código o agregando pruebas de regresión automatizadas, se vuelve más difícil ofrecer nuevas funciones y la deuda "compuesta" frena al equipo.

Desarrollo basado en pruebas (TDD) En el desarrollo basado en pruebas, el programador escribe y automatiza una pequeña prueba unitaria, que inicialmente falla, antes de escribir la cantidad mínima de código que hará que la prueba pase. El código se refactoriza según sea necesario para cumplir con estándares aceptables.

El código de producción está diseñado para funcionar una prueba a la vez. TDD, también conocido como diseño basado en pruebas, es más una práctica de diseño de código que una actividad de prueba y ayuda a crear código sólido y fácil de mantener.

Prueba unitaria Una prueba unitaria verifica el comportamiento de una pequeña parte del sistema general, como una sola rama de una única función. Puede ser tan pequeño como un único objeto o método que sea consecuencia de una o más decisiones de diseño. En un contexto ágil, las pruebas unitarias automatizadas guían la codificación a un nivel bajo. Una prueba unitaria suele tener menos de una docena de líneas de código. Algunos profesionales prefieren otros términos como microtest (Hill, 2009) o prueba xunit (Fowler, 2014).

Esta página se dejó en blanco intencionalmente.

Referencias

Adzic, Gojko, Cerrando la brecha de comunicación: especificación con ejemplos y pruebas de aceptación ágiles, Neuri Limited, 2009.

_____, "Cómo prueba Google la ingeniería", [http://gojko.net/2010/10/15/cómo-google-does-test-engineering/](http://gojko.net/2010/10/15/c%C3%B3mo-google-does-test-engineering/), 2010a.

_____, "Cómo implementar pruebas de interfaz de usuario sin dispararse en el Pie", <http://gojko.net/2010/04/13/how-to-implement-ui-testing-without-shooting-yourself-in-the-foot-2/>, 2010b.

_____, Especificación con ejemplo: cómo los equipos exitosos ofrecen el software adecuado, Manning, 2011.

_____, Mapeo de impacto: lograr un gran impacto con productos y proyectos de software, Provoking Thoughts Ltd., 2012.

_____, "La revolución de febrero", <http://gojko.net/2013/02/13/the-revolucion-febrero>, 2013.

Adzic, Gojko, Declan Whelan y otros, diagrama de "Especificación por ejemplo", <https://docs.google.com/drawings/d/1cbfKq-KazcbMVCnRfih6zMSDBdtf90KviV7l2oxGyWM/edit?hl=en>, taller Agile Alliance aa-ftt, 2010.

Agile Alliance, "Junta Kanban", <http://guide.agilealliance.org/guide/kanban.html> , 2012.

Andrea, Jennitta, "Desarrollo basado en pruebas de aceptación: no es tan opcional como usted piensa", www.stickyminds.com/article/acceptance-test-driven-development-not-optional-you-think , StickyMinds, boletín informativo Iterations, 2010.

Appelo, Jurgen, Management 3.0: Desarrolladores ágiles líderes, Desarrollo de líderes ágiles, Addison-Wesley, 2011.

_____, "Envoltura de retroalimentación", www.management30.com/workout/feedback-wrap/, 2013.

Ariely, Dan, "¿Tenemos el control de nuestras propias decisiones?" www.ted.com/talks/dan_ariely_asks_are_we_in_control_of_our_own_decisions, 2008.

Bach, Jon, "Gestión de pruebas basada en sesiones", www.satisfice.com/articles/sbtm.pdf, Revista Software Testing and Quality Engineering , ahora revista Better Software , 2000.

Beck, Kent, Desarrollo basado en pruebas: con el ejemplo, Addison-Wesley, 2002.

Bell, Rob, "Una guía para principiantes sobre la notación O grande", <http://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/>, 2009.

Brodwell, Johannes, "Programación de pares remotos", www.slideshare.net/jhannes/2013-0807-agile-2013-remote-pair-programming, agosto de 2013.

Instituto CMMI, Carnegie Mellon, "Soluciones", <http://cmmiinstitute.com/cmmi-soluciones/>, 2014.

Cockburn, Alistair, "Caracterización de las personas como componentes no lineales de primer orden en el desarrollo de software", <http://alistair.cockburn.us/Caracterizar+a+las+personas+como+componentes+no-lineales%2c+de+primer-orden+en+el+desarrollo+de+software>, 1999.

_____, "Arquitectura hexagonal: el patrón: puertos y adaptadores", <http://alistair.cockburn.us/Hexagonal+architecture>, 2005.

Crispin, Lisa y Janet Gregory, Pruebas ágiles: una guía práctica para evaluadores y equipos ágiles, Addison-Wesley, 2009.

Cunningham, Ward, "La metáfora de la deuda", www.youtube.com/watch?v=pqeJFYwnkjE, 2009.

De Bono, Edward, Pensando para la acción, DK Publishing, 1998.

Derby, Esther, Johanna Rothman y Gerald M. Weinberg, "Curso de liderazgo para la resolución de problemas", www.estherderby.com/problem-solving-liderazgo-psl , 2014.

Dinwiddie, George, "A Lingua Franca Between the Three (o More) Ami-gos" , <http://blog.gdinwiddie.com/2010/02/25/a-lingua-franca-between-the-tres-or-más-amigos/>, 2010.

Eliot, Seth, "Pruebas de la A a la Z en producción: metodologías , técnicas y ejemplos de TiP en STPCon 2012", www.setheliot.com/blog/a-to-z-testing-in-production-tip-methodologies-techniques-and-ejemplos-en-stpcon-2012, 2012.

Emery, Dale, "Redacción de pruebas de aceptación automatizadas mantenibles", dhemery.com/pdf/writing_maintainable_automated_acceptance_tests.pdf, 2009.

Evans, David, "Visualising Quality", <http://prezi.com/yych7sndetph/visualising-quality-atd> , 2013.

Fowler, Martin, "UnitTest", <http://martinfowler.com/bliki/UnitTest.html>, 2014.

Freeman, Steve y Nat Pryce, Desarrollo de software orientado a objetos, guiado por pruebas, Addison-Wesley, 2009.

Gärtner, Markus, "La deuda técnica aplicada a las pruebas", www.shino.de/2009/01/08/technical-debt-applied-to-testing/ , 2009.

_____, "Prueba Pomodoro", www.shino.de/tag/pomodoro-testing, 2011.

_____, ATDD con el ejemplo: una guía práctica para el desarrollo basado en pruebas de aceptación, Addison-Wesley, 2012.

_____, comunicación personal, 2014.

Gawande, Atul, "Slow Ideas", The New Yorker, 29 de julio de 2013.

Gilb, Tom y Kai Gilb, "Planguage", www.gilb.com/definitionPlanguage&estructura=Glosario&page_ref_id=476, 2013.

Gottesdiener, Ellen, Requisitos por colaboración: talleres para definir necesidades, Addison-Wesley, 2002.

_____, El Jogger de memoria de requisitos de software: una guía de bolsillo para ayudar a los equipos comerciales y de software a desarrollar y gestionar requisitos, Goal QPC Inc., 2005.

- Gottesdiener, Ellen y Mary Gorman, Descubrir para entregar: análisis y planificación ágil de productos, 2012.
- Gregory, Janet, "Acerca del aprendizaje 2", <http://janetgregory.ca/about-learning-2/>, 2010.
- _____, "Rompecabezas y trozos pequeños", <http://janetgregory.ca/jigsaw-puzzles-small-chunks/>, 2011.
- _____, "Equipos distribuidos: mi experiencia", <http://janetgregory.ca/equipos-distribuidos-mi-experiencia/>, 2014.
- Invitado, David, "La búsqueda del hombre informático del Renacimiento ha comenzado", The Independent (Londres), 1991.
- Hagar, Jon, Ataques de prueba de software para romper dispositivos móviles e integrados, Chapman y Hall/CRC, 2013.
- _____, comunicación personal, 2014.
- Hagberg, Jan Petter, comunicación personal, 2013.
- Haines, Corey y otros, Coderetreat Community Network, <http://coderetreat.org/>.
- Harty, Julian, "Trinity Testing", <http://julianharty-softwaretesting.blogspot.ca/2010/05/trinity-testing.html>, 2010.
- _____, Pruebas y automatización de pruebas para aplicaciones de teléfonos móviles, Chap-man y Hall/CRC, 2015.
- Hassa, Christian, "Story Maps in Practice", www.slideshare.net/chassa/2013-0509story-mapsagilemeetupbp, 2013.
- Hendrickson, Elisabeth, "Pruebas: no una fase, sino una forma de vida", <http://testobsessed.com/2006/11/testing-not-a-phase-but-a-way-of-life/>, 2006.
- _____, "Impulsando el desarrollo con pruebas: ATDD y TDD", <http://testobsessed.com/wp-content/uploads/2011/04/atddexample.pdf>, 2008.
- _____, "La prueba ácida ágil", <http://testobsessed.com/2010/12/the-agile-prueba-acida/>, 2010.

- _____, "Hoja de referencia de heuristicas de prueba", <http://testobsessed.com/wp-content/uploads/2011/04/testtheuristicscheatsheetv1.pdf> , 2011.
- _____, "The Thinking Tester: Evolucionado", www.slideshare.net/ehendrickson/el-probador-de-pensamiento-evolucionado, 2012.
- _____, ¡explóralo! Reduzca el riesgo y aumente la confianza con pruebas exploratorias, Programador pragmático, 2013.
- Heusser, Matthew, Cómo reducir el costo de las pruebas de software, Taylor y Francis, 2011.
- Hill, Michael, "Se llaman micropruebas", <http://anarchycreek.com/2009/05/20/theyreceived-microtests/> , 2009.
- Hiranabe, Kenji, "Kanban aplicado al desarrollo de software: de ágil a eficiente", www.infoq.com/articles/hiranabe-lean-agile-kanban, InfoQ, 2008.
- Humble, Jez, "No existe tal cosa como un 'equipo DevOps'", <http://continuedelivery.com/2012/10/theres-no-such-thing-as-a-devops-team/> , 2012.
- Humble, Jez y David Farley, Entrega continua: lanzamientos de software confiables mediante la automatización de compilación, prueba e implementación, Addison-Wesley, 2010.
- Hunt, Andy, Pensamiento y aprendizaje pragmáticos, Pragmatic Bookshelf, 2008.
- Hussman, David, "Agile Journeys", www.slideshare.net/agileee/agile-journeys-by-david-hussman-2029404 , 2009.
- _____, "Cortando un ritmo ágil: una serie de videos educativos gratuitos para principiantes y profesionales", <http://pragprog.com/screencasts/v-dhcag/cortando-un-ritmo-agil>, Pragmatic Bookshelf, 2011.
- _____, "Story Maps, Slices, Customer Journeys y otras herramientas de diseño de productos", <http://agilepalooza.com/austin2013/slidedecks/Story-Maps-Slices-Customer-Journeys.pdf>, 2013.
- Hüttermann, Michael, Agile ALM: herramientas ligeras y estrategias ágiles, Publicaciones Manning, 2011a.

_____, "ALM ágil y desarrollo colaborativo", <http://huettermann.net/perform/AgileALM-AgileRecord-Huettermann.pdf> , Agile Record, 2011b.

_____, DevOps para desarrolladores: integrar el desarrollo y las operaciones de forma ágil, Apress, 2012.

Kahneman, Daniel, Pensar rápido y lento, Farrar, Straus y Giroux, 2011.

Kaner, Cem, "¿Qué son las pruebas basadas en el contexto?" <http://kaner.com/?p=49>, 2012.

Karten, Naomi, "¿Estás escuchando?" www.agileconnection.com/article/are-you-listening , Conexión ágil, 2009.

Kemerling, Ashton, "Generative Testing", www.pivotaltracker.com/community/tracker-blog/generative-testing , Pivotal, 2014.

Keogh, Liz, publicación del grupo Yahoo Agile Testing, agosto de 2010.

_____, "Desarrollo impulsado por el comportamiento", www.slideshare.net/lunivore/desarrollo-impulsado-por-comportamiento-11754474, 2012a.

_____, "El taller de descubrimiento deliberado", <http://lizkeogh.com/2012/09/21/the-deliberate-discovery-workshop/> , 2012b.

_____, "BDD antes de las herramientas", <http://lizkeogh.com/2013/10/24/bdd-antes-de-las-herramientas/>, 2013a.

_____, "Embracing Uncertainty", que incluye enlaces a publicaciones sobre Cynefin, Deliberate Discovery y Real Options, <http://lizkeogh.com/embracing-uncertainty/> , 2013b.

_____, "Capacidades discretas versus continuas", <http://lizkeogh.com/2014/02/10/discrete-vs-continuous-capabilities/> , 2014a.

Keogh, Liz, comunicación personal, 2014b.

Kniberg, Henrik y Spotify Labs, "Spotify Culture", <http://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1/> , 2013.

- Kniberg, Henrik y Anders Ivarsson, "Scaling Agile @ Spotify: con tribus, escuadrones, capítulos y gremios", <http://ucvox.files.wordpress.com/2012/11/113617905-scaling-agile-spotify-11.pdf>, 2012.
- Knight, Adam P., "El hilo de una idea: adopción de un enfoque basado en hilos para las pruebas exploratorias", www.a-sisyphean-task.com/2011/11/Enfoque-basado-en-hilos-para-exploratoria.html, 2011.
- _____, "Pruebas exploratorias fractales", www.a-sisyphean-task.com/2013/01/fractal-exploratory-testing.html, 2013.
- _____, comunicación personal, 2014.
- Kohl, Jonathan, Aproveche las pruebas móviles, LeanPub, 2013.
- Kubasek, Stanley, "La regla de los boy scouts", <http://pragmaticcraftsman.com/2011/03/the-boy-scout-rule/>, The Pragmatic Craftsman, 2011.
- Lambert, Rob, "Los probadores en forma de T y su papel en un equipo", <http://thesocialtester.co.uk/t-shaped-testers-and-their-role-in-a-team/>, 2012.
- Levison, Mark, "La mente del principiante: un enfoque para escuchar", www.infoq.com/news/2008/08/beginners_mind, InfoQ, 2008.
- Lyndsay, James, "Aventuras en pruebas basadas en sesiones", www.workroom-productions.com/papers/AiSBTv1.2.pdf, 2003.
- _____, "Por qué la exploración tiene un lugar en cualquier estrategia", www.workroom-productions.com/papers/Exploration%20and%20Strategy.pdf, 2006.
- _____, "Pruebas en un entorno ágil", www.workroom-productions.com/papers/Testing%20in%20an%20agile%20environment.pdf, 2007.
- _____, comunicación personal, 2014.
- Manns, Mary Lynn y Linda Rising, Cambio intrépido: patrones para introducir nuevas ideas, Addison-Wesley, 2005.
- Marick, Brian, "Instrucciones de pruebas ágiles: pruebas y ejemplos", www.exampler.com/old-blog/2003/08/22/#agile-testing-project-2, 2003.
- _____, Secuencias de comandos cotidianas con Ruby: para equipos, evaluadores y para usted, Prag-matic Bookshelf, 2007.

Martin, Robert C., Código limpio: un manual de artesanía en software, Prentice Hall, 2009.

_____, The Clean Coder: un código de conducta para programadores profesionales, Prentice Hall, 2011.

Matts, Chris y Gojko Adzic, "Inyección de funciones: tres pasos hacia el éxito", www.infoq.com/articles/feature-injection-success, InfoQ, 2011.

Matts, Chris y Olav Maassen, "Las 'opciones reales' subyacen a las prácticas ágiles", www.infoq.com/articles/real-options-enhance-agility, InfoQ, 2007.

McDonald, Kent J., "Crear un contrato de proveedor mientras se mantiene ágil", www.techwell.com/2013/05/create-vendor-contract-while-keeping-agile, Tech-well, 2013.

McDonald, Mark, "¿Es su empresa una empresa? La respuesta importa", http://blogs.gartner.com/mark_mcdonald/2009/06/15/is-your-company-an-enterprise-the-answer-matters/, Gartner Blog Network, 2009.

McMahon, Chris, "Política de teletrabajo", <http://chrismahonsblog.blogspot.com/2009/12/telecommuting-policy.html> , 2009.

Morgan, Jeff "Cheezy", "Pruebas de interfaz de usuario: datos predeterminados", www.cheezyworld.com/2010/11/21/ui-tests-default-dat , 2010.

_____, Pepino y queso: un taller de probadores, LeanPub, 2013.

_____, comunicación personal, 2014.

Morville, Peter, "Diseño de experiencia de usuario", <http://semanticstudios.com/publicaciones/semanitca/000029.php>, 2004.

Nordstrom, "Laboratorio de innovación de Nordstrom: estudio de caso de la aplicación Sunglass para iPad", www.youtube.com/watch?v=szr0ezLyQHY, 2011.

North, Dan, "Presentación de BDD", <http://dannorth.net/introduciendo-bdd/>, Revista Better Software , 2006.

Ottinger, Tim y Jeff Langr, "Arrange-Act-Assert", <http://agileinaflash.blogspot.com/2009/03/arrange-act-assert.html> , 2009a.

_____, Ágil en un instante: software ágil de aprendizaje rápido, estantería pragmática, 2011.

- _____, "Modelo "FURPS", <http://agileinaflash.blogspot.ca/2009/04/furps.html>, 2009b.
- _____, "Refactor-rojo-verde", <http://agileinaflash.blogspot.co.uk/2009/02/rojo-verde-refactor.html>, 2009c.
- Patton, Jeff, "Todo depende de cómo lo divida: diseñe su proyecto en capas de trabajo para evitar versiones incrementales a medio hacer", www.agileproductdesign.com/writing/how_you_slice_it.pdf , Revista Better Software , 2005.
- _____, "Personas pragmáticas", www.stickyminds.com/article/pragmatic-personas , StickyMinds/Techwell, 2010.
- _____, Mapeo de historias de usuario: creación de mejores productos utilizando software ágil Diseño, O'Reilly Media, 2014.
- Rainsberger, JB, "La próxima década de desarrollo de software ágil", www.slideshare.net/jbrains/the-next-decade-of-agile-software-development , 2013.
- Ramdeo, Anand, "Automatización de pruebas: ¿Cómo manejar componentes comunes con el modelo de objetos de página?", www.testinggeek.com/test-automation-how-to-handle-common-components-with-page-object-model , 2013.
- Rasmussen, Jonathan, The Agile Samurai: Cómo los maestros ágiles ofrecen un excelente software, Pragmatic Bookshelf, 2010.
- Ries, Eric, "Estudio de caso: SlideShare se vuelve Freemium", www.startuplessonslearned.com/2010/08/case-study-slideshare-goes-freemium.html , 2010.
- Rising, Linda, "El poder de las retrospectivas", www.stickyminds.com/presentation/power-retrospectives-0 , SQE, 2009.
- Rogalsky, Steve, "Reflexiones sobre más allá de los plazos de Jabe Bloom", <http://winnipegagilist.blogspot.ca/2013/03/thinkts-on-beyond-deadlines-by-jabe.html> , 2012.
- Rothman, Johanna, "Agile no es para todos", www.jrothman.com/blog/mpd/2012/12/agile-is-not-for-everyone.html, 2012a.
- _____, Contratación de geeks que encajen, Pragmatic Bookshelf, 2012b.

Ruhland, Bernice Niel, "Desarrollo de sus habilidades de liderazgo a través de un club de diario", <http://thetestersedge.com/2013/12/28/developing-your-leadership-skills-through-a-journal-club/> , 2013a.

_____, comunicación personal, 2013b.

_____, comunicación personal, 2014.

Red Global Satir, <http://satirglobal.org/>.

Scott, Alister, "Especificación por ejemplo: una historia de amor", <http://watirmelon.files.wordpress.com/2011/05/specificationbyexamplealovestory1.pdf> , 2011a.

_____, "Otra pirámide más de pruebas de software", <http://watirmelon.com/2011/06/10/yet-another-software-testing-pyramid/> , 2011b.

SearchCloudComputing, "Infraestructura como servicio", <http://searchcloudcomputing.techtarget.com/definition/Infrastructure-as-a-Service-IaaS> , 2010a.

_____, "Plataforma como servicio", <http://searchcloudcomputing.techtarget.com/definition/Platform-as-a-Service-PaaS> , 2010b.

_____, "Software como servicio", <http://searchcloudcomputing.techtarget.com/definition/Software-as-a-Service> , 2010c.

Gestión de datos de búsqueda, "Tecnología de federación de datos", <http://searchdatamanagement.techtarget.com/definition/data-federation-technology> , 2010.

Sherry, Rosie, mapa mental de pruebas móviles del Ministerio de Pruebas, www.flickr.com/photos/softwaretestingclub/7159412943/sizes/o/in/photostream/ , Ministerio de Pruebas, 2013.

Sweets, Tony, "Ejemplo de objeto de página", <https://github.com/tsweets/page-objects> , 2013.

Charlas, Mike, El campo minado del software, LeanPub, 2012.

Traynor, Des, "Una entrevista con Andy Budd", <http://insideintercom.io/an-interview-with-andy-budd/> , Intercom, 2011.

- Tung, Portia, "Power of Play", www.slideshare.net/portiatung/the-powerofplay36 , 2011.
- Waters, John K., "Un enfoque ágil de la ciencia espacial", <http://adtmag.com/articles/2004/10/06/an-agile-approach-to-rocket-science.aspx> , Revista Application Development Trends , 16 de octubre de 2004.
- Whittaker, James, "ACC Explicado", <http://code.google.com/p/test-analytics/wiki/AccExplained> , 2011.
- _____, "Visitas de prueba exploratorias", http://msdn.microsoft.com/en-us/biblioteca/jj620911.aspx#bkmk_tours, Microsoft, 2012.
- Whittaker, James A., Jason Arbon y Jeff Carollo, Cómo prueba Google el software, Addison-Wesley, 2012.
- Wikipedia, "Cinco porqués", http://en.wikipedia.org/wiki/5_Whys, 2014a.
- _____, "Investigación contextual", http://en.wikipedia.org/wiki/Contextual_investigation, 2014b.
- _____, "Sistema integrado: Historia", http://en.wikipedia.org/wiki/Sistema_integrado#Historia, 2014c.
- _____, "Software empresarial", http://en.wikipedia.org/wiki/Enterprise_software, 2014d.
- _____, "Extraer, transformar, cargar", https://en.wikipedia.org/wiki/Extracto,_transformaci%C3%B3n,_carga, 2014e.
- _____, "FURPS", <http://en.wikipedia.org/wiki/FURPS>, 2014f.
- _____, "Heurística", <http://en.wikipedia.org/wiki/Heuristics>, 2014g.
- _____, "Diagrama de Ishikawa", http://en.wikipedia.org/wiki/Ishikawa_diagrama, 2014h.
- _____, "Producto mínimo viable", http://en.wikipedia.org/wiki/Minimum_viable_product , 2014i.
- _____, "Control de revisiones", http://en.wikipedia.org/wiki/Revision_controlar, 2014j.

- _____, “Cuestionamiento socrático”, http://en.wikipedia.org/wiki/Socratic_questioning, 2014k.
- _____, “Pruebas de software”, http://en.wikipedia.org/wiki/Software_testing, 2014l.
- _____, “SÓLIDO: Diseño orientado a objetos”, <http://en.wikipedia.org/wiki/SOLID>, 2014m.
- _____, “Esquema en estrella”, http://en.wikipedia.org/wiki/Star_schema, 2014n.

Wikispeed, sitio web, <http://wikispeed.org/>, 2014.

Wynne, Matt y Aslak Hellesøy, The Cucumber Book: Desarrollo impulsado por el comportamiento para evaluadores y desarrolladores, programadores pragmáticos, 2012.

Bibliografía

Parte I: Introducción

Libros

Appelo, Jurgen, Management 3.0: Desarrolladores ágiles líderes, Desarrollo de líderes ágiles, Addison-Wesley, 2011.

Beck, Kent, Desarrollo basado en pruebas: con el ejemplo, Addison-Wesley, 2002.

Benson, Jim y Tonianne DeMaria Berry, Kanban personal: trabajo de mapeo | Navegando por la vida, plataforma de publicación independiente CreateSpace, 2011.

Maassen, Olav, Chris Matts y Chris Geary, Compromiso, Publicaciones Hathaway te Brake, 2013.

Manns, Mary Lynn y Linda Rising, Cambio intrépido: patrones para introducir nuevas ideas, Addison-Wesley, 2005.

Martin, Robert C., Clean Coder: un código de conducta para programadores profesionales, Prentice Hall, 2011.

Poppendieck, Mary y Tom Poppendieck, Implementación del desarrollo de software ajustado: del concepto al efectivo, Addison-Wesley, 2006.

_____, La mentalidad ajustada: haga las preguntas correctas, Addison-Wesley, 2013.

Rasmussen, Jonathan, The Agile Samurai: Cómo los maestros ágiles ofrecen un excelente software, Pragmatic Bookshelf, 2010.

Reinertsen, Donald G., Los principios del flujo de desarrollo de productos: desarrollo de productos ajustados de segunda generación, Celeritas Publishing, 2012.

Sitios web, blogs, artículos, presentaciones de diapositivas

Adzic, Gojko, "La revolución de febrero", <http://gojko.net/2013/02/13/the-february-revolution>, 2013.

Benson, Jim, "Tiempo de finalización", [www.personalkanban.com/pk/uncategorized/time-to-completion/#s\(hash.WdeVZ5i7.dpbs](http://www.personalkanban.com/pk/uncategorized/time-to-completion/#s(hash.WdeVZ5i7.dpbs) , 2011.

Benson, Jim y Jeremy Lightsmith, "Lean Coffee", <http://leancoffee.org>, 2013.

Derby, Esther, sitio web, <http://estherderby.com>.

Dinwiddie, George, "A Lingua Franca Between the Three (o More) Amigos" , <http://blog.gdinwiddie.com/2010/02/25/a-lingua-franca-between-the-tres-ore-mas-amigos/>, 2010.

Gawande, Atul, "Slow Ideas", The New Yorker, 29 de julio de 2013.

Hendrickson, Elisabeth, "Pruebas: no una fase, sino una forma de vida", <http://testobsessed.com/2006/11/testing-not-a-phase-but-a-way-of-life/>, 2006.

Keogh, Liz, "Embracing Uncertainty", que incluye enlaces a publicaciones en Cynefin, Deliberate Discovery y Real Options, <http://lizkeogh.com/embracing-uncertainty/> , 2013.

Kniberg, Henrik, "Propiedad de productos ágil en pocas palabras", www.youtube.com/watch?v=502ILHjX9EE , 2012.

Kniberg, Henrik y Anders Ivarsson, "Scaling Agile @ Spotify: con tribus, escuadrones, capítulos y gremios", <http://ucvox.files.wordpress.com/2012/11/113617905-scaling-agile-spotify-11.pdf>, 2012.

Kniberg, Henrik y Spotify Labs, "Spotify Culture", <http://labs.spotify.com/2014/03/27 spotify-engineering-culture-part-1> , 2013.

Matts, Chris y Olav Maassen, "Las 'opciones reales' subyacen a las prácticas ágiles", www.infoq.com/articles/real-options-enhance-agility, InfoQ, 2007.

Rogalsky, Steve, "Reflexiones sobre más allá de los plazos de Jabe Bloom", <http://winnipegagilist.blogspot.ca/2013/03/thinkts-on-beyond-deadlines-by-jabe.html> , 2012.

_____, "Una guía para el café magro", www.slideshare.net/SteveRogalsky/una-guia-para-el-cafe-magro, 2013.

Rothman, Johanna, "Agile no es para todos", www.jrothman.com/blog/mpd/2012/12/agile-is-not-for-everyone.html, 2012.

_____, "Confianza, gestión ágil de programas y ser eficaz", www.jrothman.com/blog/mpd/2013/08/trust-agile-program-management-being-Effective.html, 2013.

Escocia, Karl, "Presentación de Kanban, flujo y cadencia", <http://agile.dzone.com/news/introtaining-kanban-flow-and>, Dzone, Inc., 2009.

West, Dave, "Analyst Watch: Water-Scrum-Fall es la realidad de Agile", www.sdtimes.com/content/article.aspx?ArticleID=36195&page=1, Tiempos SD, 2011.

_____, "Water-Scrum-Fall es la realidad de Agile para la mayoría de las organizaciones actuales", www.forrester.com/WaterScrumFall+Is+The+Reality+Of+Agile+For+Most+Organizations+Today/fulltext/-/E-RES60109?docid=60109, Forrester Research, 2011.

Wikipedia, "Hábitat", <http://en.wikipedia.org/wiki/Cynefin>, 2014.

Zheglov, Alexei, "El esquivo 20% del tiempo", <http://connected-knowledge.com/2012/05/10/the-elusive-20-time/>, 2013.

Parte II: Aprender para realizar mejores pruebas

Libros

Adkins, Lyssa, Coaching de equipos ágiles: un compañero para ScrumMasters, coaches ágiles y gerentes de proyectos en transición, Addison-Wesley, 2010.

Adzic, Gojko, Especificación con el ejemplo: cómo los equipos exitosos ofrecen el software adecuado, Manning, 2011.

Appelo, Jurgen, Management 3.0: Desarrolladores ágiles líderes, Desarrollo de líderes ágiles, Addison-Wesley, 2010.

Copeland, Lee, Guía para profesionales sobre diseño de pruebas de software, Artech House, 2004.

Davies, Rachel y Liz Sedley, Agile Coaching, Pragmatic Bookshelf, 2009.

De Bono, Edward, Pensando para la acción, DK Publishing, 1998.

Derby, Esther, Don Gray, Johanna Rothman y Gerald M. Weinberg, Lecturas para el liderazgo en la resolución de problemas, LeanPub, 2013, <https://leanpub.com/pslreader>.

Gärtner, Markus, ATDD con el ejemplo: una guía práctica para el desarrollo basado en pruebas de aceptación, Addison-Wesley, 2012.

Gottesdiener, Ellen, Requisitos por colaboración: talleres para definir necesidades, Addison-Wesley, 2002.

Hunt, Andy, Pensamiento y aprendizaje pragmáticos, Pragmatic Bookshelf, 2008.

Kahneman, Daniel, Pensar rápido y lento, Farrar, Straus y Giroux, 2011.

Kaner, Cem, Sowmya Padmanabhan y Douglas Hoffman, The Domain Testing Workbook, Context Driven Press, 2013.

Larman, Craig y Bas Vodde, Escalar el desarrollo eficiente y ágil: herramientas organizativas y de pensamiento para Scrum a gran escala, Addison-Wesley, 2009.

Martin, Robert C., Desarrollo ágil de software: principios, patrones y prácticas, Prentice Hall, 2002.

_____, Código limpio: un manual de artesanía en software, Prentice Hall, 2009.

_____, The Clean Coder: un código de conducta para programadores profesionales, Prentice Hall, 2011.

Morgan, Jeff "Cheezy", Pepino y queso: un taller de evaluadores, LeanPub, 2013.

Ottinger, Tim y Jeff Langr, Agile in a Flash: software ágil de aprendizaje rápido, Pragmatic Bookshelf, 2011.

Patterson, Kerry y otros, Conversaciones cruciales: herramientas para hablar cuando hay mucho en juego, segunda edición, McGraw-Hill, 2001.

Rothman, Johanna, Contratación de geeks que encajen, Pragmatic Bookshelf, 2012.

Satir, Virginia, El modelo Satir: terapia familiar y más, Libros de ciencia y comportamiento, 1991.

Seashore, Charles N., Edith Whitfield Seashore y Gerald M. Weinberg, ¿Qué dijiste? El arte de dar y recibir comentarios, Bingham House Books, 1997.

Sullivan, Wendy y Julie Rees, Lenguaje limpio: revelar metáforas y abrir mentes, Crown House Publishing, 2008.

Tabaka, Jean, Explicación de la colaboración: habilidades de facilitación para líderes de proyectos de software, Addison-Wesley, 2006.

Publicaciones de blog y artículos en línea

Appelo, Jurgen, "Resumen de comentarios", [www.management30.com/workout/
envoltura de comentarios /](http://www.management30.com/workout/envoltura_de_comentarios/), 2013.

Ariely, Dan, "¿Tenemos el control de nuestras propias decisiones?" [www.ted.com/
talks/dan_ariely_asks_are_we_in_control_of_our_own_decisions](http://www.ted.com/talks/dan_ariely_asks_are_we_in_control_of_our_own_decisions), 2008.

Bolton, Michael, "Pruebas de aceptación del usuario" , [www.developsense.com/
presentaciones/User%20Acceptance%20Testing%20-%20STAR%20East%20
2006.pdf](http://www.developsense.com/presentaciones/User%20Acceptance%20Testing%20-%20STAR%20East%202006.pdf).

_____, "Pensamiento crítico para evaluadores", [www.developsense.com/
presentaciones/2012-11-EuroSTAR-CriticalThinkingForTesters.pdf](http://www.developsense.com/presentaciones/2012-11-EuroSTAR-CriticalThinkingForTesters.pdf), 2012.

Cockburn, Alistair, "Caracterización de las personas como componentes no lineales de primer orden en el desarrollo de software", [http://alistair.cockburn.us/
Caracterizar+a+las+personas+como+componentes+no-lineales%2c+de+primer-orden+en+
el+desarrollo+de+software](http://alistair.cockburn.us/Caracterizar+a+las+personas+como+componentes+no-lineales%2c+de+primer-orden+en+el+desarrollo+de+software), 1999.

_____, "Arquitectura hexagonal: el patrón: puertos y adaptadores", [http://
alistair.cockburn.us/Hexagonal+architecture](http://alistair.cockburn.us/Hexagonal+architecture), 2005.

Crispin, Lisa, "Aplicación del modelo Dreyfus de adquisición de habilidades", [http://
lisacrispin.com/2012/06/25/applying-the-dreyfus-model-of-skill-adquisition-to-the-
whole-team-approach/](http://lisacrispin.com/2012/06/25/applying-the-dreyfus-model-of-skill-adquisition-to-the-whole-team-approach/) , 2012.

Gregory, Janet, "Acerca del aprendizaje 2", [http://janetgregory.ca/about-
learning-2/](http://janetgregory.ca/about-learning-2/) , 2010.

Invitado, David, "La búsqueda del hombre informático del Renacimiento ha comenzado", The Independent (Londres), 1991.

Hendrickson, Elisabeth, "La prueba de ácido ágil", <http://testobsessed.com/2010/12/the-agile-acid-test/> , 2010.

Karten, Naomi, "¿Estás escuchando?" www.agileconnection.com/article/are-you-listening , Agile Connection, 2009.

Keogh, Liz, "El taller de descubrimiento deliberado", <http://lizkeogh.com/2012/09/21/the-deliberate-discovery-workshop/> , 2012.

Knight, Adam P., "Probador en forma de T, equipo en forma cuadrada", <http://thesocialtester.co.uk/t-Shaped-tester-square-filled-team/> , 2013.

Lambert, Rob, "Los probadores en forma de T y su papel en un equipo", <http://thesocialtester.co.uk/t-shaped-testers-and-their-role-in-a-team/> , 2012.

Levison, Mark, "La mente del principiante: un enfoque para escuchar", www.infoq.com/news/2008/08/beginners_mind, InfoQ, 2008.

McKee, Lynn, "Inspiración y motivación a través del aprendizaje", www.qualityperspectives.ca/blog/802 , 2010.

McMillan, Darren, "Mapas mentales 101", www.bettertesting.co.uk/content/?p=956, Mejores pruebas, 2011.

Fundación Myers & Briggs, "Conceptos básicos de MBTI", www.myersbriggs.org/mi-tipo-de-personalidad-mbti/mbti-basics.

Ruhland, Bernice Niel, "Desarrollo de sus habilidades de liderazgo a través de un club de diario", <http://thetestersedge.com/2013/12/28/developing-your-leadership-skills-through-a-journal-club/> , 2013.

Tatham, Elizabeth, "Funciones en proyectos de código abierto", <http://oss-watch.ac.uk/recursos/rolesinopensource>, OSS Watch, 2010, actualizado en 2013.

Toastmasters Internacional, www.toastmasters.org/.

Tung, Portia, "Power of Play", www.slideshare.net/portiatung/the-powerofplay36 , 2011.

Vaage, Carol, "Juego y aprendizaje para niños", www.k-3learningpages.net/profesional%20desarrollo.htm.

Wedig, Steve, "Lista de lectura de un desarrollador de software", <http://stevewedig.com/2014/02/03/software-developers-reading-list/> , 2014.

Wikipedia, "Cinco por qués", http://en.wikipedia.org/wiki/5_Whys, 2014.

_____, "Diagrama de Ishikawa", http://en.wikipedia.org/wiki/Ishikawa_diagrama, 2014.

_____, "Cuestionamiento socrático", http://en.wikipedia.org/wiki/Cuestionamiento_socrático, 2014.

Cursos, Conferencias, Comunidades en línea , Podcasts

Campamento de entrenadores ágiles, <http://agilecoachcamp.org/>.

Balamurugadas, Ajay, et al., "Pruebas de fin de semana", <http://weekendtesting.com/>.

Crispin, Lisa y Janet Gregory, moderadores, Lista de correo de pruebas ágiles, <http://tech.groups.yahoo.com/group/agile-testing/>.

Derby, Esther, Don Gray, Johanna Rothman y Gerald M. Weinberg, "Lecturas para el liderazgo en la resolución de problemas", <https://leanpub.com/pslreader>, LeanPub, 2013.

Derby, Esther, Johanna Rothman y Gerald M. Weinberg, "Curso de liderazgo para la resolución de problemas", www.estherderby.com/problem-solving-liderazgo-psl , 2014.

Filipino, Zeljko, WATIR Podcast, <http://watirpodcast.com/>.

Gärtner, Markus, "Testing Dojos", www.testingdojo.org.

Haines, Corey y otros, Coderetreat, <http://coderetreat.org/>.

Kaner, Fiedler and Associates, "Diseño de pruebas: una encuesta sobre técnicas de pruebas de caja negra", www.testingeducation.org/BBST/testdesign/, 2014.

Khoo, Trish y Bruce McLeod, "TestCast", www.testcast.net.

Larsen, Michael, podcasts TWiST, www.mktesthead.com/p/podcasts.html.

Play4Agile, <http://play4agile.wordpress.com/>.

Red Global Satir, <http://satirglobal.org/>.

Sherry, Rosie, et al., comunidad en línea de Software Testing Club, www.softwaretestingclub.com/.

Asociación de profesionales de pruebas de software, podcasts, [www.softwaretestpro.com/List/Podcasts](http://www.softwaretestpro.com>List/Podcasts).

Starter League, escuela de software centrada en principiantes, www.starterleague.com/.

Weirich, Jim y Joe O'Brien, Neo Ruby Koans, <http://rubykoans.com/>.

Parte III: Planificación, para no olvidar el panorama

Libros

Freeman, Steve y Nat Pryce, Desarrollo de software orientado a objetos, guiado por pruebas, Addison-Wesley, 2009.

Galen, Robert, Finales del software: eliminación de defectos, control de cambios y cuenta atrás para la entrega a tiempo, Dorset House, 2005.

Gottesdiener, Ellen y Mary Gorman, Descubrir para entregar: análisis y planificación ágil de productos, 2012.

Hendrickson, Elisabeth, ¡explóralo! Reduzca el riesgo y aumente la confianza con pruebas exploratorias, Programador pragmático, 2013.

Hüttermann, Michael, Agile ALM: herramientas ligeras y estrategias ágiles, Publicaciones Manning, 2011.

Whittaker, James A., Jason Arbon y Jeff Carollo, Cómo prueba Google el software, Addison-Wesley, 2012.

Artículos, publicaciones de blog , presentaciones de diapositivas

Adzic, Gojko, "Cómo prueba Google la ingeniería", <http://gojko.net/2010/10/15/how-google-does-test-engineering/>, 2010.

_____, "La revolución de febrero", <http://gojko.net/2013/02/13/la-revolucion-de-febrero/>, 2013.

_____, "Rompamos los cuadrantes de pruebas ágiles", <http://gojko.net/2013/21/10/rompamos-los-cuadrantes-de-pruebas-agiles/>, 2013.

Crispin, Lisa, "Uso de los cuadrantes de pruebas ágiles", <http://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/>, 2011.

Gärtner, Markus, "Los cuadrantes de prueba: ¡lo hicimos mal!", www.shino.de/2012/07/30/the-testing-quadrants-we-got-it-wrong/, 2012.

Hendrickson, Elisabeth, "Hoja de referencia de pruebas de heurística", <http://testobsessed.com/wp-content/uploads/2011/04/testheuristicscheatsheetv1.pdf>, 2011.

_____, "The Thinking Tester: Evolucionado", www.slideshare.net/ehendrickson/el-probador-de-pensamiento-evolucionado, 2012.

Hüttermann, Michael, "ALM ágil y desarrollo colaborativo", <http://huettermann.net/perform/AgileALM-AgileRecord-Huettermann.pdf>, Registro ágil, 2011.

Keogh, Liz, "Capacidad discreta versus continua", <http://lizkeogh.com/2014/02/10/discrete-vs-continuous-capabilities/>, 2014.

Marick, Brian, "Instrucciones de pruebas ágiles: pruebas y ejemplos", www.exampler.com/old-blog/2003/08/22/#agile-testing-project-2, 2003.

Nisbet, Duncan, "Dissecting the Testing Quadrants: Wrap Up", www.duncannisbet.co.uk/dissecting-the-testing-quadrants-wrap-up, 2014.

Ottinger, Tim y Jeff Langr, modelo "FURPS", <http://agileinaflash.blogspot.ca/2009/04/furps.html>, 2009.

Rising, Linda, "El poder de las retrospectivas", www.stickyminds.com/presentation/power-retrospectives-0, SQE, 2009.

Whittaker, James, "ACC Explicado", <http://code.google.com/p/test-analytics/wiki/AccExplained>, 2011.

Wikipedia, "FURPS", <http://en.wikipedia.org/wiki/FURPS>, 2014.

Parte IV: Prueba del valor empresarial

Libros

Adzic, Gojko, Cerrando la brecha de comunicación: especificación con ejemplos y pruebas de aceptación ágiles, Neuri Limited, 2009.

_____, Especificación con ejemplo: cómo los equipos exitosos ofrecen el software adecuado, Manning, 2011.

_____, Mapeo de impacto: lograr un gran impacto con productos y proyectos de software, Provocating Thoughts, 2012.

_____, 50 ideas rápidas para mejorar sus historias de usuario, LeanPub, 2014.

Chelimksy, David y otros, The RSpec Book: Behaviour-Driven Development with RSpec, Cucumber and Friends, Pragmatic Bookshelf, 2010.

Fowler, Martin, Lenguajes de dominio específico, Addison-Wesley, 2011.

Gärtner, Markus, ATDD con el ejemplo: una guía práctica para el desarrollo basado en pruebas de aceptación, Addison-Wesley, 2012.

Gottesdiener, Ellen, The Software Requisitos Memory Jogger: una guía de bolsillo para ayudar a los equipos de software y de negocios a desarrollar y gestionar los requisitos, Goal QPC Inc., 2005.

Gottesdiener, Ellen y Mary Gorman, Descubrir para entregar: análisis y planificación ágil de productos, 2012.

Heusser, Matthew, Cómo reducir el costo de las pruebas de software, Taylor y Francis, 2011.

Morgan, Jeff "Cheezy", Pepino y queso: un taller de evaluadores, LeanPub, 2013.

Patton, Jeff, Mapeo de historias de usuario: creación de mejores productos utilizando un diseño de software ágil, O'Reilly Media, 2014.

Ries, Eric, The Lean Startup: Cómo los emprendedores de hoy utilizan la innovación continua para crear negocios radicalmente exitosos, Crown Business, 2011.

Ulwick, Antony, Lo que quieren los clientes: utilizar la innovación basada en resultados para crear productos y servicios innovadores, McGraw-Hill, 2005.

Vance, Stephen, Código de calidad: principios, prácticas y patrones de pruebas de software, Addison-Wesley, 2013.

Wynne, Matt y Aslak Hellesøy, The Cucumber Book: Desarrollo impulsado por el comportamiento para evaluadores y desarrolladores, programadores pragmáticos, 2012.

Artículos, publicaciones de blogs , presentaciones de diapositivas y sitios web

Adzic, Gojko, "Cómo prueba Google la ingeniería", <http://gojko.net/2010/10/15/how-google-does-test-engineering/> , 2010.

_____, "La revolución de febrero", <http://gojko.net/2013/02/13/la-revolucion-de-febrero/> , 2013.

_____, "Mapeo de impacto", www.impactmapping.org, 2012.

Adzic, Gojko, Declan Whelan y otros, diagrama de "Especificación por ejemplo", <https://docs.google.com/drawings/d/1cbfKq-KazcbMVCnRfih6zMMSDBdtf90KviV7I2oxGyWM/edit?hl=en>, taller Agile Alliance aa-fft, 2010.

Andrea, Jennitta, "Desarrollo basado en pruebas de aceptación: no es tan opcional como usted piensa", www.stickyminds.com/article/acceptance-test-driven-development-not-optional-you-think , StickyMinds, boletín informativo Iterations, 2010.

Gat, Israel y Jim Highsmith, número de Cutter IT Journal sobre "Deuda técnica", www.cutter.com/offers/technicaldebt.html, Cutter Consortium, 2009.

Hassa, Christian, "Story Maps in Practice", www.slideshare.net/chassa/2013-0509story-mapsagilemeetupbp , 2013.

Hendrickson, Elisabeth, "Impulsando el desarrollo con pruebas: ATDD y TDD", <http://testobsessed.com/wp-content/uploads/2011/04/atddexample.pdf>, 2008.

Hussman, David, "Cutting an Agile Groove: una serie de videos educativos gratuitos para principiantes y profesionales", <http://pragprog.com/screencasts/v-dhcag/cutting-an-agile-groove> , Pragmatic Bookshelf, 2011.

_____, "Story Maps, Slices, Customer Journeys y otras herramientas de diseño de productos", <http://agilepalooza.com/austin2013/slidedecks/Story-Maps-Slices-Customer-Journeys.pdf> , 2013.

Keogh, Liz, "Desarrollo impulsado por el comportamiento", www.slideshare.net/lunivore/behavior-driven-development-11754474 , 2012.

Matts, Chris y Gojko Adzic, "Inyección de funciones: tres pasos hacia el éxito", www.infoq.com/articles/feature-injection-success, InfoQ, 2011.

Morgan, Jeff "Cheezy", "Pruebas de interfaz de usuario: datos predeterminados", www.cheezyworld.com/2010/11/21/ui-tests-default-dat , 2010.

_____, joya de objeto de página, <https://github.com/cheezy/page-object>, 2014.

North, Dan, "Presentación de BDD", <http://dannorth.net/introduciendo-bdd/>, Better Software, 2006.

Patton, Jeff, "Todo depende de cómo lo divida: diseñe su proyecto en capas de trabajo para evitar versiones incrementales a medio hacer", www.agileproductdesign.com/writing/how_you_slice_it.pdf , Mejor software revista, 2005.

_____, "El nuevo backlog de historias de usuario es un mapa de historias", www.agileproductdesign.com/blog/the_new_backlog.html , 2008.

_____, "Personas pragmáticas", www.stickyminds.com/article/pragmatic-personas , StickyMinds/Techwell, 2010.

Rainsberger, JB, "La próxima década de desarrollo de software ágil", www.slideshare.net/jbrains/the-next-decade-of-agile-software-development , 2013.

Ries, Eric, "Estudio de caso: SlideShare se vuelve Freemium", www.startuplessonslearned.com/2010/08/case-study-slideshare-goes-freemium.html , 2010.

Parte V: Pruebas de investigación

Libros

Hendrickson, Elisabeth, ¡explóralo! Reduzca el riesgo y aumente la confianza con pruebas exploratorias, Programador pragmático, 2013.

Kaner, Cem, Sowmya Padmanabhan y Douglas Hoffman, The Domain Testing Workbook, Context Driven Press, 2013.

Krug, Steve, La cirugía con cohetes simplificada: la guía que puede hacer usted mismo para encontrar y solucionar problemas de usabilidad, New Riders, 2009.

_____, No me hagas pensar, revisado: un enfoque de sentido común para Usabilidad web, tercera edición, New Riders, 2014.

Charlas, Mike, El campo minado del software, LeanPub, 2012.

Whittaker, James, Pruebas exploratorias de software: consejos, trucos, recorridos y técnicas para guiar el diseño de pruebas, Addison-Wesley, 2009.

Artículos, publicaciones de blogs , presentaciones de diapositivas y sitios web

Bach, Jon, "Gestión de pruebas basada en sesiones", www.satisfice.com/articles/sbtm.pdf, Revista Software Testing and Quality Engineering , ahora revista Better Software , 2000.

_____, "Pruebas en sesión: cómo medir las pruebas exploratorias", www.sasqag.org/pastmeetings/ExploratoryTesting_SessionBasedTestManagement.pdf, Quardev Laboratories, 2004.

Bolton, Michael, "Blog: Of Testing Tours and Dashboards", www.developsense.com/blog/2009/04/of-testing-tours-and-dashboards/ , 2009.

Eliot, Seth, "Pruebas de la A a la Z en producción: metodologías , técnicas y ejemplos de TiP en STPCon 2012", www.setheliot.com/blog/a-to-z-testing-in-production-tip-methodologies-techniques-and-ejemplos-en-stpcon-2012, 2012.

_____, "Hazlo en producción", <http://blogs.msdn.com/b/seliot/archive/2013/05/01/do-it-in-production-testbash-video-now-available.aspx> , 2013.

Francino, Yvette, "Seis recorridos para realizar pruebas exploratorias en el distrito comercial de su aplicación", <http://searchsoftwarequality.techtarget.com/tip/Six-tours-for-exploratory-testing-the-business-district-of-your-application>, Calidad del software de búsqueda, 2009.

Gärtner, Markus, "Pomodoro Testing", www.shino.de/tag/pomodoro-testing , 2011.

Gilb, Tom y Kai Gilb, "Planguage", www.gilb.com/definitionPlanguage&structure=Glossary&page_ref_id=476 , 2013.

_____, "Requisitos: la base para una gestión exitosa de proyectos", www.gilb.com/Requirements, 2013.

Harty, Julian, "Trinity Testing", <http://julianharty-softwaretesting.blogspot.ca/2010/05/trinity-testing.html> , 2010.

Hendrickson, Elisabeth, "Las dos caras de las pruebas de software: verificación y exploración", www.agileconnection.com/article/two-sides-software-testing-check-and-exploring?page=0%2C1 , Agile Connection, 2009.

_____, "Hoja de referencia para pruebas de heurística", <http://testobsessed.com/wp-content/uploads/2011/04/testtheuristicscheatsheetv1.pdf> , 2011.

Hussman, David, "Agile Journeys", www.slideshare.net/agileeee/agile-journeys-by-david-hussman-2029404 , 2009.

_____, "Cortando un ritmo ágil: una serie de videos educativos gratuitos para principiantes y profesionales", <http://pragprog.com/screencasts/v-dhcag/cortando-un-ritmo-agil>, Pragmatic Bookshelf, 2011.

_____, "Story Maps, Slices, Customer Journeys y otras herramientas de diseño de productos", [http://agilepalooza.com/austin2013/slidedecks/ Story-Maps-Slices-Customer-Journeys.pdf](http://agilepalooza.com/austin2013/slidedecks/Story-Maps-Slices-Customer-Journeys.pdf), 2013.

Jamie (no se proporcionó el apellido), "Detrás de escena: Pruebas A/B del sitio de marketing de gran altura, parte 1", <http://signalvnoise.com/posts/2977-behind-the-scenes-highrise-marketing-site-ab-prueba-parte-1>, 2011.

Kaner, Cem, "Visitas de prueba: ¿investigación de mejores prácticas?", <http://kaner.com/?p=96> , 2011.

Kaner, Fiedler & Associates, "Diseño de pruebas: una encuesta sobre técnicas de pruebas de caja negra", www.testingeducation.org/BBST/testdesign, 2014.

Kelly, Michael D., "Touring Heuristic", <http://michaeldkelly.com/blog/2005/9/20/touring-heuristic.html> , 2005.

Knight, Adam P., "El hilo de una idea: adopción de un enfoque basado en hilos para las pruebas exploratorias", www.a-sisyphean-task.com/2011/11/Enfoque-basado-en-hilos-para-exploratoria.html, 2011.

_____, "Pruebas exploratorias fractales", www.a-sisyphean-task.com/2013/01/fractal-exploratory-testing.html , 2013.

Kohavi, Ron, Thomas Crook y Roger Longbotham, "Experimentación en línea en Microsoft", www.exp-platform.com/Documents/ExP_DMCaseStudies.pdf , 2009.

Lambert, Rob, "Gestión de pruebas exploratorias", <http://thesocialtester.co.uk/managing-exploratory-testing/> , 2013.

Lorang, Noah, "Detrás de escena: Pruebas A/B Parte 2 : Cómo probamos", <http://signalvnoise.com/posts/2983-behind-the-scenes-ab-testing-part-2-how-we-test>, 2011.

Lyndsay, James, "Aventuras en pruebas basadas en sesiones", www.workroom-productions.com/papers/AiSBTv1.2.pdf , 2003.

_____, "Por qué la exploración tiene un lugar en cualquier estrategia", www.workroom-productions.com/papers/Exploration%20and%20Strategy.pdf , 2006.

_____, "Pruebas en un entorno ágil", www.workroom-productions.com/papers/Testing%20in%20an%20agile%20environment.pdf , 2007.

_____, "Herramientas para pruebas exploratorias", <http://workroomprds.blogspot.ca/2008/06/tools-for-exploratory-testing.html> , 2008.

Margolis, Michael, "Obtenga mejores datos de los estudios de usuarios: 16 consejos para entrevistas", www.designstaff.org/articles/get-better-data-from-user-studies-interviewing-tips-2012-03-07.html , 2012 .

McKinney, Andrew, publicaciones y artículos de blog sobre pruebas de usabilidad e investigación de usuarios, <http://andrewmckinney.com>.

_____, "Colaboración remota de artistas", <http://andrewmckinney.com/proyectos/artista-colaboracion-disney/> , 2013.

McMillan, Darren, publicaciones de blog sobre pruebas de accesibilidad, www.bettertesting.co.uk/contenido/?s=accesibilidad, 2012.

Miller, Evan, "Cómo no ejecutar una prueba A/B", www.evanmiller.org/how-not-to-run-an-ab-test.html , 2010.

Morville, Peter, "Diseño de experiencia de usuario", <http://semanticstudios.com/publicaciones/semanatica/000029.php>, 2004.

- Nguyen, Buu, "Pruebas exploratorias con qTrace 2.0", www.qasymphony.com/exploratory-testing-with-qtrace-2-0.html , 2012.
- Nordstrom, "Laboratorio de innovación de Nordstrom: estudio de caso de la aplicación Sunglass para iPad", www.youtube.com/watch?v=szr0ezLyQHY, 2011.
- Patton, Jeff, "Pragmatic Personas", www.stickyminds.com/article/pragmatic-personas , StickyMinds/Techwell, 2010.
- Ries, Eric, "Comenzando con las pruebas divididas", www.startuplessonslearned.com/2008/12/getting-started-with-split-testing.html , 2008.
- _____, "La prueba de división de una línea, o cómo hacer A/B todo el tiempo", www.startuplessonslearned.com/2008/09/one-line-split-test-or-how-to-ab-all.html , 2008 .
- Shaulis, Carl, "Informe de experiencia en pruebas A/B", <http://kungfutesting.blogspot.com/2014/03/ab-testing-experience-report.html> , 2014.
- Software de prueba de software, "Todos los tipos de pruebas de software", www.softwaretestingsoftware.com/all-types-of-software-testing/, Software Test-ing Software, 2012.
- Traynor, Des, "Una entrevista con Andy Budd", <http://insideintercom.io/an-interview-with-andy-budd/> , Intercom, 2011.
- Whittaker, James, "ACC Explicado", <http://code.google.com/p/test-analytics/wiki/AccExplained> , 2011.
- _____, "Visitas de prueba exploratorias", http://msdn.microsoft.com/en-us/biblioteca/jj620911.aspx#bkmk_tours, Microsoft, 2012.
- Wikipedia, "Consulta contextual", http://en.wikipedia.org/wiki/Contextual_inquiry , 2014.
- _____, "Pruebas de software", http://en.wikipedia.org/wiki/Software_testing, 2014.

Parte VI: Automatización de pruebas

Libros

Adzic, Gojko, Especificación con el ejemplo: cómo los equipos exitosos ofrecen el software adecuado, Manning, 2011.

Chelimksy, David y otros, The RSpec Book: Behaviour-Driven Development with RSpec, Cucumber and Friends, Pragmatic Bookshelf, 2010.

Gärtner, Markus, ATDD con el ejemplo: una guía práctica para el desarrollo basado en pruebas de aceptación, Addison-Wesley, 2012.

Marick, Brian, Scripting diario con Ruby: para equipos, evaluadores y para usted, Pragmatic Bookshelf, 2007.

Morgan, Jeff "Cheezy", Pepino y queso: un taller de evaluadores, LeanPub, 2013.

Vance, Stephen, Código de calidad: principios, prácticas y patrones de pruebas de software, Addison-Wesley, 2013.

Wynne, Matt y Aslak Hellesøy, The Cucumber Book: Desarrollo impulsado por el comportamiento para evaluadores y desarrolladores, programadores pragmáticos, 2012.

Artículos, publicaciones de blog , cursos, videos, ejemplos de código

Adzic, Gojko, "Cómo implementar pruebas de interfaz de usuario sin dispararse en el pie", <http://gojko.net/2010/04/13/how-to-implement-ui-testing-without-shooting-yourself-in-the-pie-2/>, 2010.

Bell, Rob, "Una guía para principiantes sobre la notación O grande", <http://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/> , 2009.

Cunningham, Ward, "La metáfora de la deuda", www.youtube.com/watch?v=pqeJFYwnkjE, 2009.

Emery, Dale, "Redacción de pruebas de aceptación automatizadas mantenibles", dhemery.com/pdf/writing_maintainable_automated_acceptance_tests.pdf, 2009.

Gärtner, Markus, "La deuda técnica aplicada a las pruebas", www.shino.de/2009/01/08/technical-debt-applied-to-testing/ , 2009.

- Goucher, Adam, "Secuencias de comandos para ingenieros de pruebas", http://adam.goucher.ca/?page_id=305 , 2007.
- _____, "Objetos de página en Python", <http://pragprog.com/magazines/2010-08/page-objects-in-python> , Pragmatic Bookshelf, 2010.
- Kemerling, Ashton, "Generative Testing", www.pivotaltracker.com/community/tracker-blog/generative-testing , Pivotal, 2014.
- Keogh, Liz, "BDD antes de las herramientas", <http://lizkeogh.com/2013/10/24/bdd-before-the-tools/> , 2013.
- Kubasek, Stanley, patrón de reglas de los Boy Scouts, <http://pragmaticcraftsman.com/2011/03/the-boy-scout-rule/> , 2011.
- Marcano, Antony, "Un poco de UCD para BDD y ATDD: Metas → Tareas → Acciones", <http://antonymarcano.com/blog/2011/03/goals-tasks-action/> , 2011.
- Mendenhall, Connor, "Check Your Work", <http://blog.8thlight.com/connor-mendenhall/2013/10/31/check-your-work.html> , 8th Light, 2013.
- Morgan, Jeff "Cheezy", "Pruebas de interfaz de usuario: datos predeterminados", www.cheezyworld.com/2010/11/21/ui-tests-default-dat , 2010.
- Ottinger, Tim y Jeff Langr, "Arrange-Act-Assert", <http://agileinaflash.blogspot.com/2009/03/arrange-act-assert.html> , 2009.
- _____, "Refactor-rojo-verde", <http://agileinaflash.blogspot.co.uk/2009/02/rojo-verde-refactor.html> , 2009.
- Ramdeo, Anand, "El viaje de un probador de software desde lo manual a lo político", www.testinggeek.com/a-software-tester-s-journey-from-manual-to-political , 2012.
- _____, "Un paso a la vez", www.youtube.com/watch?v=dFPgzH_XP1I , 2012.
- _____, "Automatización de pruebas: cómo manejar componentes comunes con ¿Modelo de objetos de página?", www.testinggeek.com/test-automation-how-to-handle-common-components-with-page-object-model , 2013.
- Scott, Alister, "Especificación por ejemplo: una historia de amor", <http://watirmelon.files.wordpress.com/2011/05/specificationbyexamplealovelystory1.pdf> , 2011.

_____, “Otra pirámide de pruebas de software más”, <http://watirmelon.com/2011/06/10/ yet-another-software-testing-pyramid/>, 2011.

Sweets, Tony, “Ejemplo de objeto de página”, <https://github.com/tsweets/page-objects> , 2013.

Wikipedia, “Patrón de eliminación”, http://en.wikipedia.org/wiki/Dispose_pattern , 2013.

_____, “SÓLIDO: Diseño orientado a objetos”, <http://en.wikipedia.org/wiki/SOLID>, 2014.

Parte VII: ¿Cuál es su contexto?

Libros

Gruber, Gary, Mike Young y Pat Fulghum, Un enfoque práctico para el desarrollo ágil a gran escala: cómo HP transformó el firmware LaserJet FutureSmart, Addison-Wesley, 2012.

Hagar, Jon, Ataques de prueba de software para romper dispositivos móviles e integrados, Chapman y Hall/CRC, 2013.

Harty, Julian, Pruebas y automatización de pruebas para aplicaciones de teléfonos móviles, Chapman y Hall/CRC, 2015.

Hubbard, Douglas W., Cómo medir cualquier cosa: encontrar el valor de los intangibles en los negocios, Wiley, 2010.

Humble, Jez y David Farley, Entrega continua: lanzamientos de software confiables mediante la automatización de compilación, prueba e implementación, Addison-Wesley, 2010.

Hüttermann, Michael, DevOps para desarrolladores: integrar el desarrollo y las operaciones de forma ágil, Apress, 2012.

Kohl, Jonathan, Aproveche las pruebas móviles, LeanPub, 2013.

Larman, Craig y Bas Vodde, Scaling Lean & Agile: herramientas de pensamiento y organización para Scrum a gran escala, Addison-Wesley, 2009.

_____, Prácticas para escalar el desarrollo eficiente y ágil: grande, multisitio, y desarrollo de productos offshore con Scrum a gran escala, Addison-Wesley, 2010.

Larsen, Diana y Ainsley Nies, Liftoff: Lanzamiento de equipos y proyectos ágiles, Onyx Neon Press, 2011.

Artículos, publicaciones de blog , presentaciones de diapositivas , sitios web

Brodwell, Johannes, "Programación de pares remotos", www.slideshare.net/jhannes/2013-0807-agile-2013-remote-pair-programming, agosto de 2013.

Chaney, Clareice, Clyneice Chaney y XBOSoft, "Seminario web sobre la mejor manera de contratar un equipo de pruebas ágil subcontratado" , www.slideshare.net/xbosoft/best-way-to-contract-an-outsourced-agile-w-bnr-0718v3 , 2013.

Instituto CMMI, "Soluciones", <http://cmmiinstitute.com/cmmi-solutions/>, Carnegie Mellon, 2014.

CNET, ejemplo de informe de uso de dispositivos móviles, http://news.cnet.com/8301-1023_3-57605422-93/ie-usage-resurges-in-september-new-stats-report/ , octubre de 2013.

Cockburn, Alistair, "La piedra angular de Agile: por qué funciona, por qué duele", www.slideshare.net/itweekend/the-cornerstone-of-agile-why-it-works-why-it-hurts , 2012.

Crispin, Lisa, "Factores de éxito para equipos distribuidos", <http://searchsoftwarequality.techtarget.com/tip/Success-factors-for-distributed-teams> , Search Software Quality, 2010.

de Kok, Dirk, "Cómo aplicar Lean Startup a dispositivos móviles", <http://blog.mobtest.com/2012/11/how-to-apply-lean-startup-to-mobile/> , 2012.

Evans, David, "Visualising Quality", <http://prezi.com/yych7sndetph/visualising-quality-atd> , 2013.

Gregory, Janet, "Equipos distribuidos: mi experiencia", <http://janetgregory.ca/distributed-teams-my-experience/> , 2014.

Hagar, Jon, "Breaking Mobile and Embedded Software", <http://breakingembeddedsoftware.com/> .

Hagar, Jon y Mark Dornseif, "Agile Evolution of Launch Vehicle Space Software Systems", Conferencia AIAA Space 2004, septiembre de 2004, <http://arc.aiaa.org/doi/abs/10.2514/6.2004-5802>.

Hagar, Jon y Randall Smith, "Producing Embedded Flight Software with Agile Commercial and Government Practices", Conferencia de tecnología de software IEEE 2003, mayo de 2003, <https://sw.thecsiac.com/techs/abstract/347299>.

Humble, Jez, "No existe tal cosa como un 'equipo DevOps'", <http://continuedelivery.com/2012/10/theres-no-such-thing-as-a-devops-team/>, 2012.

Hummel, Geoff, "¿El desarrollo de software ágil se combina bien con las regulaciones de la FDA?" www.thetestking.com/2013/09/does-agile-software-development-mix-well-with-fda-regulations/, 2013.

Kelly, Allan, "Contratos ágiles", www.infoq.com/articles/agile-contracts, InfoQ, 2011.

Knight, Adam P., "Testing Big Data in an Agile Environment", [www.ministryoftesting.com/2013/06/testing-big-data-in-an-agile-environment/](http://www.ministryoftesting.com/2013/06/testing-big-data-in-an-agile-environment), Ministerio de Pruebas, 2013.

Kohl, Jonathan, "Pruebe aplicaciones móviles con I SLICED UP FUN!" www.kohl.ca/articles/ISLICEDUPFUN.pdf, 2010.

Lehrer, Jonah, "La tecnología por sí sola no es suficiente", www.newyorker.com/en línea/blogs/newsdesk/2011/10/steve-jobs-pixar.html, El neoyorquino, 7 de octubre de 2011.

McDonald, Kent J., "Crear un contrato de proveedor mientras se mantiene ágil", www.techwell.com/2013/05/create-vendor-contract-while-keeping-agile, Techwell, 2013.

McDonald, Mark, "¿Es su empresa una empresa? La respuesta importa", http://blogs.gartner.com/mark_mcdonald/2009/06/15/is-your-company-an-enterprise-the-answer-matters/, Gartner Blog Network, 2009.

McMahon, Chris, "Política de teletrabajo", <http://chriscmahonsblog.blogspot.com/2009/12/telecommuting-policy.html>, 2009.

Mobiletech, ejemplo de informe de uso de dispositivos móviles, <http://mobiletechglobal.com/estadísticas-moviles-enero-2013/>, 2013.

Morville, Peter, "Diseño de experiencia de usuario", <http://semanticstudios.com/publicaciones/semantica/000029.php>, 2004.

Ramdeo, Anand, "Automatización de pruebas: ¿Cómo manejar componentes comunes con el modelo de objetos de página?" www.testinggeek.com/test-automation-how-to-handle-common-components-with-page-object-model , 2013 .

Sanchez, Carlos, presentaciones sobre DevOps y pruebas de infraestructura, www.slideshare.net/carlosgg/presentations, 2013.

Shah, Shahid, "Escribir software crítico para la seguridad utilizando un enfoque ágil basado en riesgos", www.healthcareguy.com/2013/06/14/writing-safety-critical-software-using-an-agile-risk-based-approach -debería-ser-la-norma-en-el-desarrollo-de-dispositivos-médicos-modernos/, 2013.

Sherry, Rosie, mapa mental de pruebas móviles del Ministerio de Pruebas, www.flickr.com/photos/softwaretestingclub/7159412943/sizes/o/in/photostream/ , Ministerio de Pruebas, 2013.

Siener, Graham, "Inception: saber qué construir y por dónde empezar", http://pivotallabs.com/agile-inception_knowing-what-to-build-and-where-to-start/ , 2013.

Sweets, Tony, "Entornos virtuales de construcción continua de Hudson: fuera lo viejo", www.stickyminds.com/article/virtual-hudson-continuous-build-environments-out-old , StickyMinds, 2011.

TforTesting (sin otro nombre), "Casos de prueba para aplicaciones de juegos/Listas de verificación para aplicaciones de juegos ", <http://tfortesting.wordpress.com/2012/10/04/test-cases-for-games-apps-checklist-for-juegos-aplicaciones/>, 2012.

Trimble, Jay y Chris Webster, "Métodos de desarrollo ágiles para operaciones espaciales", http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20120013429_2012013093.pdf , 2012.

Waters, John K., "Un enfoque ágil de la ciencia espacial", <http://adtmag.com/articles/2004/10/06/an-agile-approach-to-rocket-science.aspx> , Revista Application Development Trends, 16 de octubre de 2004.

Webster, C., "Entrega de software al Centro de control de misión de la NASA mediante técnicas de desarrollo ágiles", <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6187329&icp=false&url=http%3A%2F%2Fieeexplore.ieee.org%2Fapplication%2F6187329&isip=false&isipid=1&isipname=Guest>

Wikipedia, "Imágenes de disco", http://en.wikipedia.org/wiki/Disk_imaging, 2014.

_____, "Sistema integrado: Historia", http://en.wikipedia.org/wiki/Sistema_integrado#Historia, 2014.

_____, "Software empresarial", http://en.wikipedia.org/wiki/Enterprise_software, 2014.

_____, "Pruebas de aceptación operativa", http://en.wikipedia.org/wiki/Pruebas_de_aceptación_operacional, 2014.

_____, "Esquema en estrella", http://en.wikipedia.org/wiki/Star_schema, 2014.

_____, "Starship Enterprise", http://en.wikipedia.org/wiki/Starship_Enterprise, Wikipedia, 2014.

Sitio web de Wikispeed, <http://wikispeed.org/>, 2014.

Parte VIII: Pruebas ágiles en la práctica

Gregory, Janet, "Rompecabezas y piezas pequeñas", <http://janetgregory.ca/rompecabezas-pequeños-trozos/>, 2009.

Kaner, Cem, "¿Qué son las pruebas basadas en el contexto?", <http://kaner.com/?p=49>, 2012.

Herramientas (ordenadas por nombre de herramienta)

Nota: Estos enlaces son válidos al momento de escribir este artículo, pero no hay garantía de que la herramienta mencionada o el enlace todavía estén en uso.

Clonezilla: Código abierto, Clonezilla, <http://sourceforge.net/projects/clonezilla/>, SourceForge.

Fit: Open Source, creado originalmente por Ward Cunningham, Fit: Framework for Integrated Test, <http://fit.c2.com/>, c2.com.

FitNesse: código abierto, <http://fitnesse.org/>.

Obtener Kanban: GetKanban.com, juego de mesa "Get Kanban", <http://getkanban.com/BoardGame.html>, GetKanban.com.

Git: Código abierto, Git, <http://git-scm.com>.

Jdefault: Sweets, Tony, "jdefault: biblioteca de datos predeterminada de Java", <https://github.com/tsweets/jdefault>, GitHub, 2013.

Jenkins CI: código abierto, Jenkins CI, <http://jenkins-ci.org/>.

Jenkins Job DSL/Plugin: Código abierto, Jenkins Job DSL/Plugin, <https://github.com/jenkinsci/job-dsl-plugin>, GitHub.

Gema de objeto de página: Morgan, Jeff "Cheezy", gema de objeto de página, <https://github.com/cheezy/page-object>, 2014.

Marioneta: código abierto, marioneta, <http://puppetlabs.com/>, Puppet Labs.

Puppet-Lint: código abierto, joya Puppet-Lint, <http://puppet-lint.com>, Rodjek.

Rastrillo: Weirich, Jim, rastrillo, <https://github.com/jimweirich/rake>, GitHub.

Rapid Reporter: Gershon, Shmuel, "Rapid Reporter, Exploratory Notetaking", <http://testing.gershon.info/reporter/>.

SonarQube: Código abierto, SonarQube, <http://sonarqube.org>, Sonar Source.

Vagrant: Código abierto, Vagrant, www.vagrantup.com, HashiCorp.

Índice

Números

"Cinco porqués", en visualización del proceso de pensamiento, 44

7 Dimensiones del producto (Gottesdiener y Gorman)

ejemplo de aplicación de 7 dimensiones del producto, 131–133

para identificar las necesidades de productos en todos los niveles de planificación, 129–131

A

AA-FTT (Herramientas de prueba funcionales de Agile Alliance), 4, 153

Pruebas A/B

en pruebas de hipótesis, 134 para

aplicaciones móviles, 328

descripción general de, 203–204

Diseñadores y probadores de UX que utilizan, 140

Matriz ACC (capacidad de componente de atributo), uso del modelo en pruebas, 113–114

Automatización de

pruebas de aceptación, 214,

259 definidas,

415 de alto nivel,

153 incluido el diseño de UX en las pruebas,

141 pruebas de aceptación operativa (OAT), 387–389, 419

Enfoques ágiles de desarrollo basado en pruebas de aceptación

(ATDD) utilizados con sistemas móviles e integrados,

329 prácticas de fomento de la

confianza, 394 herramientas de prueba

funcionales y 237 guías con ejemplos,

55, 148–152 reducción de la deuda defectuosa,

214–215 escalamiento "Discutir -Ciclo

Destilar-Desarrollar-Demo", 276

Rendición de cuentas, a partir de auditorías, 340

Departamento de contabilidad, gestión de dependencias internas, 292

Acciones, en 7 Dimensiones de Producto, 130

Adzic, Gojko, xxix, 112, 153, 155

sobre la sección de colaboradores, xxxv

ATDD y, 151 sobre

pruebas automatizadas como documentación viva, 339 sobre la

automatización de pruebas de aceptación, 214

desafío al modelo de Cuadrantes, 108–112 sobre el

fomento de una cultura de aprendizaje, 10, 13–14 sobre

mapeo de impacto, 123 sobre el

uso de modelos para visibilidad, 114 sobre

SBE, 56, 153 talleres

de especificación, 42 sobre pruebas a

través de la UI, 243

Diagramas de afinidad, 44

"Prueba de ácido ágil" (Hendrickson), 79

Herramientas de prueba funcional de Agile Alliance (AA-FTT), 4, 153

ALM ágil (Hüttermann), 104

Manifiesto Ágil, 15, 87

El ágil samurái (Rasmussen), xxi

Pruebas ágiles: una guía práctica para probadores y ágiles

Equipos (Crispin y Gregory), xxi, 3, 282 colaboración, 239 equipos de clientes

y desarrolladores, 27 sistemas de seguimiento

de defectos, 321 principios y patrones

de diseño, 240–241, 243 documentación, 141, 340 sistemas integrados, 326 final del juego, 285

uso de ejemplo, 145, 148, 155

personas, 171

SBTM, 176

siete factores clave de éxito, 393 ejemplos

de guiones gráficos, 386 creación de

tareas, 97 diez

principios para los evaluadores, 28, 386

automatización de pruebas, 209, 223, 237, 254, 262

función del administrador de pruebas, 19

- Pruebas ágiles: una guía práctica para probadores y ágiles
- Equipo (continuación)
 - prueba Cuadrantes, 65–66, 85, 108
 - selección de herramientas,
 - 264 herramientas para obtener ejemplos y
 - requisitos, 123 pruebas
 - de aceptación del usuario, 90, 202 visibilidad, 381, 391 enfoque de todo el equipo, xvii hilos de acero, 160, 313
- ALMA. Ver Gestión del ciclo de vida de la aplicación (ALM)
- Habilidades analíticas, requisitos para los probadores DW/BI, 348. Ver también Análisis empresarial (BA)
- Software de análisis, herramientas para dispositivos móviles, 327
- Andrea, Jennitta, 149 Nivel
- API (nivel de servicio) en la
- pirámide de automatización, 223–224 coherencia en la selección de herramientas y 265 pruebas, 241–243 arquitectura API (interfaces de programación de aplicaciones), 58 codificación habilidades y, 57–58 enfoque de equipo completo, 258–260 Appelo, Jurgen sobre desarrollo ágil, 16 entrenamiento "Feedback Wrap", 45
 - Ejemplo de Apple, sobre el valor del diseño de producto, 140 Gestión del ciclo de vida de la aplicación (ALM)
- Agile ALM (Hüttermann), 104 definidos, 415
- Arquitectura, para
- API, 58 Ariely, Dan, 50 Patrón
- Arrange-Act-Assert
- (Ottinger y Langr), 240 Artículos, recursos de aprendizaje, 75 ATDD por ejemplo (Gärtner), 56 Componente de atributos Matriz de capacidad (ACC), uso del modelo en pruebas, 113–114 Atributos
- en matriz ACC, 113 atributos de calidad. Ver Auditorías de atributos de calidad rendición de cuentas de, 340 auditores como partes interesadas, 342, 346
- Automatización, flexibilidad de herramientas de automatización, 256 de canalizaciones de compilación, 368 de pruebas de verificación de compilación, 369–371 de procesos de extracción, transformación y carga (ETL), 349–350
- de aprovisionamiento de estados base de configuración, 374–376
- de pruebas. Ver automatización de pruebas
- ## B
- Bach, Jon, 177
- Copia de seguridad/restauración, gestión de Big Data, 358
- "Bake-offs", en pruebas de herramientas, 261–262
- Barcomb, Matt
- sobre la sección de contribuyentes, xxxv sobre cómo convertirse en un especialista generalizador, 33–36
 - sobre el uso de la pirámide de automatización de pruebas, 228–229 BDD. Ver Desarrollo impulsado por el comportamiento (BDD)
- Beck, Kent, 9
- "Mente de principiante" (Hunt), 50 Desarrollo impulsado por el comportamiento (BDD)
- prácticas (básicas) de fomento de la confianza, 394 definidas, 415–416
 - evolución de las pruebas ágiles y 3 herramientas de prueba funcionales y, 237
 - Estilo dado, cuándo, entonces, xxii–xxiii, 147, 152, 156, guía del desarrollo con ejemplos, 55, 148, 152–153
 - uso con sistemas móviles e integrados, 329
- Evaluación comparativa en el establecimiento de objetivos, 277
- Desafíos de
- Big Data de, 357–359
 - transacciones de bases de datos y, 352
- Beneficios
- generales de los principios ágiles, 297
 - pruebas de funciones y, 391, 396–397 factores clave de éxito, 393 entrega de productos y, 91 uso del modelo de cuadrantes, 108 comenzar bien y, 239
 - visualizar, 127

- Bligh, Susan
sobre la sección de contribuyentes, xxxv–xxxvi sobre pruebas de aceptación de usuarios en empresas, 294–296
- BLM (construir-medir-aprender)
definido, 416
para pruebas o exploración tempranas, 134
- Boltón, Michael
evolución de las pruebas ágiles y, 3 sobre las pruebas como ciencia social, 25
- Lluvia de ideas para lograr consenso para soluciones de automatización, 262, 263 en proceso
de colaboración, 49, 52 por ejemplo fuentes, 155 facilitación, 42 mapeo de impacto, 123 herramientas de visualización, 44
- Cuestiones de ramificación e integración continua relacionadas con, 62–65
- Brodwell, Johannes, 312
- Navegadores
problemas de compatibilidad, 91 pruebas de aplicaciones basadas en navegador, 244 Budd, Andy, 140 errores. Consulte Defectos/errores Automatización de canalizaciones de compilación, 368 definidos, 416 DevOps y, 367–368 entornos de prueba y, 367 pruebas de verificación, 369–371
- Creación de lo correcto. Consulte Desarrollo, creación de lo correcto, definido por Construir-medir-aprender (BLM), 416 para pruebas o exploración tempranas, 134 Análisis de negocios (BA) combinado con pruebas, 129, 395, incluidos analistas de negocios en equipos ágiles, 27–28 proceso de incorporación para analistas de negocios, 37–38 requisitos de habilidades, 137–139 evaluadores y 139–140 capacidades comerciales. Ver características
- Pruebas de inteligencia empresarial (BI)
aplicar principios ágiles a, 351 desafíos de Big Data, 357, 359 datos en, 352–353 aprender a probar, 351–352 administrar datos de prueba, 355–356 para rendimiento y escala, 357–359 resolver problemas de datos de prueba incorrectos, 353–354 aspectos únicos de, 347–350
- Reglas del negocio
niveles para pruebas a través de la interfaz de usuario, 243 en la pirámide de automatización de pruebas, 233–234
- Valor de negocio
determinar los trabajos del probador. Consulte Probadores, que determinan las responsabilidades laborales y desarrollan lo correcto. Ver Desarrollo, construyendo lo correcto ejemplos. Ver ejemplos que guían el desarrollo de componentes clave de, 119–120, 122 pruebas en la entrega, 4–5 aceptación, falta de, 160–161
- C
Capacidades, en matriz ACC, 114 Utilización de la capacidad, Zheglov en, 10–12 Sistema de tarjeta, para seguimiento de deuda técnica, 230 Carvalho, Paul sobre la sección de colaboradores, xxxvi sobre pruebas de internacionalización y localización, 195–199
- Cambio
de adaptación a (Ruhland), 15 pruebas de un extremo a otro, 229–230 Cambio intrépido (Manns y Rising), 19 aprendizaje y, 9
- Carlos, Fiona, 25
- Cartas
creación, 168–171 generación de ideas para, 171 viajes en creación, 175–176 gestión, 176 personas en creación, 171–174 en SBTM, 176–178 historias como base de, 175 en TBMT, 178–183

- Tours charters (continuación)
en la creación, 174–175
- Gráficos
ejemplo de uso en la vida cotidiana, 146–147
visualizando lo que está probando, 100
- Codificador limpio (Martin), 401
- Entrenadores
retiros de coaching, 74
aprender de, 71
habilidades de, 48–49
- Cockburn, Alistair, 36 años
- Código/codificación
que verifica el código comercial con SonarQube, 366 creación de pruebas antes de codificar, 96
retroalimentación rápida y 354
integración con pruebas en TDD, 105–106
Guía de prueba del segundo trimestre, 103
refactorización, 213 habilidades técnicas, 56–58 control de versión (o código fuente), 60–65, 420
- Cohn, Mike, xxix–xxx, 115–116, 223
- Colaboración
en enfoque a la selección de herramientas, 264
herramientas colaborativas versus no colaborativas, 265 con clientes, 147, 393 DevOps y.
Consulte equipos distribuidos de DevOps y, 309–311 pruebas de DW/BI y, 350 aprender de, 73–74
escuchar y, 49 satisfacer necesidades
regulatorias, 344–346 procesos (Robson), 52–53
reducir la deuda técnica, 218–220
habilidades técnicas y, 56 a través de pruebas, 311–312 herramientas para equipos distribuidos, 319–321 usando herramientas de video chat para, 312, 319–320 valorando la colaboración del cliente sobre la negociación de contratos, 151–152
enfoque de equipo completo para las pruebas y, 239
- Línea de comando, habilidades técnicas generales, 59
- Comunicación
gestión de conflictos, 318 cuestiones culturales y lingüísticas, 303–304 equipos distribuidos y, 306–307, 309–311 de la importancia de las pruebas, 381–382
- problemas de zona horaria, 304–305
herramientas para equipos distribuidos, 319
uso de video chat para, 306
- Comunidad de práctica (CoP) de aprendizaje e intercambio, 76 evaluadores actuando como, 80
- Competencias. Ver Cumplimiento de Roles/competencias. Ver también Entornos regulatorios parte de Liberación realizada, 342 regulatorio, 340–341
- Pruebas de componentes, en la pirámide de automatización de pruebas, 224–225, 233–234
- Pruebas de simultaneidad, 194–195
- Conferencias, aprender de, 72–74
- Prácticas (básicas) de fomento de la confianza sensibilidad al contexto, 399–400
aprendizaje continuo, 397–398 uso de ejemplos, 394–395 pruebas exploratorias, 395–396 pruebas de características, 396–397 mantenerlo real, 401
- Estados base de configuración, automatización del aprovisionamiento de, 374–376
- Gestión de conflictos, equipos distribuidos y, 318
- Controles organizacionales de coherencia para lograr, 278 en la selección de herramientas, 265, 289
- Restricciones resultado mínimo aceptable, 277 planificación de pruebas del cuarto trimestre y, 107–108 limitaciones de tiempo y habilidades organizativas, 51
- Contexto sistemas de inteligencia empresarial. Ver Negocios La inteligencia (BI) prueba los almacenes de datos. Ver Almacenes de datos (DW) DevOps. Consulte Equipos distribuidos de DevOps. Ver sistemas integrados de equipos distribuidos. Consulte Pruebas de nivel empresarial de sistemas/software integrados. Consulte Descripción general de empresas (grandes organizaciones) de 271–273 entornos regulados. Ver entornos regulatorios
- Pruebas de aceptación de sensibilidad al contexto y, 150

- alternativas al modelo de cuadrante, 112-113
- Big Data y, 5 roles
- borrosos y, 142 en
- coordinación de múltiples equipos, 283 como práctica central, 399-400
- problemas culturales que enfrentan equipos distribuidos, 302-303 educación relativa a roles y responsabilidades, 17
- interdependencia del desarrollo de software, la infraestructura y las operaciones, 26 aprender de los picos antes de la planificación, 90 sistemas móviles e integrados y, 5, 333 resolución de problemas y, 43 pruebas del cuarto trimestre (pruebas orientadas a la tecnología) y, 65-66 en simplificación de enfoque, 157 comenzar bien en la automatización de pruebas, 239 TBTM y, 182 administradores de pruebas y, 19 pruebas basadas en el contexto, 399 aprendizaje continuo. Ver también Aprendizaje adquirir habilidades de automatización y codificación, 57 Habilidades de licenciatura y, 138 como práctica básica, 397-398 empoderar a los miembros del equipo, 13 contratar personas que quieran aprender, 37 tiempo para, 9 de personas de otros departamentos, 365 pequeños experimentos, 80 pruebas de software móvil, 336 Habilidades en forma de T, 28-29 valor del conocimiento del dominio, 47 Entrega continua, DevOps y 366 Entrega continua (Humble y Farley), 361 Mejora continua Kanban utilizado para, 386-389 resultados positivos de, 297 Habilidades en forma de T, 28 visualización para, 386, 390 Pruebas de regresión automatizadas de integración continua (CI) para, 289-290 crear y mantener entornos de prueba y, 60 prácticas de desarrollo centrales, 394 definidas, 416 determinar cuánta automatización es suficiente, 262 pruebas de larga duración y 229 sistemas móviles e integrados y 329 pruebas de regresión y 97 compilaciones fallidas y 263 sistemas, 62 a 65 problemas de zona horaria, 305 Habilidades en forma de T y 29 señales visuales de deuda técnica, 217 guiar al nuevo evaluador a través del proceso de CI, 38 Negociación de contratos, valoración de la colaboración del cliente, 151-152 Control en 7 dimensiones del producto, 130 que tratan de controles organizacionales, 278-283 Conversaciones obtener ejemplos, 239 como medio para entregar valor, 150-152 valor de, 147 Coordinación de múltiples equipos, 283-284 CoP (Comunidad de práctica) aprendiendo e compartiendo, 76 evaluadores actuando como, 80 prácticas principales. Consulte Prácticas (básicas) de fomento de la confianza Cursos, aprendizaje de, 74-75 Compañeros de trabajo, aprendizaje de, 77 Creatividad, aprendizaje y, 8 Pensamiento crítico, 49 Secuencias de comandos entre sitios, pruebas de seguridad y, 65-66 Pepino y queso (Morgan), 332 The Cucumber Book (Wynne y Hellesøy), 56 Herramienta Cucumber, 265 Cuestiones culturales, equipos distribuidos y, 302-303 Cunningham, Ward "ejemplos comprobados" y, 103 acuñar el término "deuda técnica", 211 evolución de las pruebas ágiles y, 3 Clientes en Agile Testing (Crispin y Gregory), 27 falta de aceptación, 160-161 capturan expectativas de, 152 colaboran con, 147, 393 pruebas orientadas al cliente que guían el desarrollo, 239 determinan el propósito de las nuevas funciones y 122 empresas involucran, 294-296 centrarse en, 276

-
- Ciencia (continuación)
- desempeñando el papel de "mal cliente" en las pruebas exploratorias, 166–167
 - equipos y 416
 - herramientas para participar. Ver *Herramientas, para el cliente*
 - Compromiso que
 - valora la colaboración del cliente sobre la negociación del contrato, 151–152.
- Tiempo de
- ciclo definido, 416
 - Acortamiento de DevOps, 361–362
 - reducción, 367
 - establecimiento de objetivos para, 387–388
- D**
- Panel de control, visibilidad de las pruebas y resultados de las pruebas, 392
- Datos
- en 7 dimensiones del producto, 130
 - inteligencia empresarial y, 352–353 desafíos de Big Data, 357, 359 gestión de datos de prueba, 249–250, 355–356 resolución de problemas de datos de prueba incorrectos, 353–354
- Pruebas de integridad de datos, requisitos de habilidades para probadores de DW/BI, 348
- Abstracción de
- modelado de datos de modelos lógicos, 351
 - para almacén de datos, 347
- Almacenes de datos (DW) que
- aplican principios ágiles, 351 desafíos de Big Data, 357, 359 entrada de datos, 352–353 aprender a realizar pruebas, 351–352 administrar datos de prueba, 355–356 resolver problemas de datos de prueba incorrectos, 353–354 tener éxito con pruebas ágiles en, 400 rendimiento y escala de las pruebas, 357–359 aspectos únicos de las pruebas en, 347–350
- Equipo de base de datos, gestión de dependencias internas, 292
- Bases de datos
- Habilidades de tester de DW/BI, 348 habilidades técnicas generales, 59 de Bono, Edward, 50
- Plazos
- generar confianza y, 17 presión poco realista, 7–8
- Sesiones informativas
- seguimiento de recorridos de prueba (Gärtner), 175
 - registro de resultados de pruebas exploratorias, 186 uso de SBTM para la capacitación de evaluadores, 176
 - depuración. Consulte *Defectos/errores*
 - Tablas de decisión, técnicas de diseño de pruebas, 67
 - Patrón de datos predeterminados (Morgan), 250
- Defectos/errores
- "búsqueda de errores", 183 captura/seguimiento, 392 errores de prueba de depuración, 250–251 implementación ágil en Dell y 281 realización de pruebas mantenimiento visible, 213 prevención, 36, 47, 102, 158, 216, 362 priorización de correcciones, 289 procesos para tratar, 287 reducción de la deuda de defectos (tolerancia cero), 213–216 enfoque visual (pizarra), 81–82
- Sistemas de seguimiento de defectos (DTS), 321
- Definiciones/supuestos, en proceso de colaboración, 52
- Entregables
- documentación formal de, 92–94 mapeo de impacto y, 124–125 Entrega. Ver también Lanzamiento del producto
 - equipo de entrega (desarrollo), 416–417 entrega de productos y, 296–297 ciclos de equipo único y múltiple, 91
- Ejemplo de Dell, de viaje ágil
- desafíos y soluciones, 279–280 evolución de, 282 implementación, 281 resultados, 296 ampliación de pruebas ágiles, 287–288
- Dependencias que
- coordinan entre equipos, 285 participación del cliente en empresas, 294–296 equipos distribuidos, 305 gestión, 292 eliminación, 293 asociaciones con terceros, 292–294
- Derby, Esther
- añadiendo bucles de retroalimentación y, 16
 - Curso de Liderazgo en Resolución de Problemas (PSL), 74
- Patrones y principios de diseño.
- Patrón de datos predeterminado (Morgan), 250

- Patrón de fábrica, 410–412
descripción general
de, 240 Patrón de objeto de página, 246–248, 407–
410 pruebas a través de la API, 241–243
pruebas a través de la interfaz de usuario,
243–246 Uso compartido de escritorio, herramientas
de colaboración, 321 Detalle, empantanarse en , 159–
160 Nivel de detalle en la planificación. Ver *Niveles de precisión*
(detalle), para la planificación
del Desarrollo. Consulte también *Prácticas*
principales de DevOps,
394 entorno para, 59–60
orientación con ejemplos, 55–56, 148–149
responsabilidad al separarse de TI u operaciones, 365 necesidad
de evaluadores, 26–27
planificación del desarrollo iterativo, 88
orientación de pruebas,
102 Equipos de desarrollo (entrega) que
adoptan valores ágiles, 7
vincularse, 345 colaborar
para satisfacer las necesidades regulatorias, 344–345 definidos,
416–417 Desarrollo,
construir lo correcto
7 Dimensiones del producto (Gottesdiener y
Gorman), 129–131
determinando el propósito de las nuevas características,
121–123
ejemplo de aplicación de 7 dimensiones del producto,
131–133
mapeo de impacto, 123–126
inversión y, 134–135
descripción general de, 119–
120 mapeo de historias, 126–
129 herramientas para la participación del
cliente, 123 herramientas para explorar tempranamente, 134
DevOps
agrega infraestructura al alcance de las pruebas,
365–367
en todos los cuadrantes, 363–364
automatizar las pruebas de verificación de compilación, 369–
371 automatizar el aprovisionamiento de la base de configuración
estados, 374–376
problemas de ramificación y, 63
construcción de canalizaciones y,
367–368
definidos, 417 evolución de las pruebas ágiles y, 5
interdependencia del desarrollo de software y la infraestructura
y operaciones, 26
monitoreo y registro y, 59 descripción
general de, 361–362 calidad
y, 363 evaluadores
que agregan valor de DevOps, 371–372
infraestructura de prueba, 372–373
DevOps para desarrolladores (Hüttermann), 361
Dinwiddie, George, xxix, 4
Descubrir para entregar (Gottesdiener y Gorman), 107, 191–192
Ciclo “Discutir-Destilar-Desarrollar-Demostrar”, en ATDD,
276 equipos
Dispersos. Ver también *Equipos distribuidos*
comunicación y colaboración y, 310 descripción general
de, 299–301 visualización
de pruebas en, 386
Equipos distribuidos
desafíos de, 302
colaboración y, 309–312, 400 herramientas
de colaboración, 319–321 comunicación
y, 309–311, 400 herramientas de comunicación,
319 estrategias de afrontamiento,
308 cuestiones culturales,
302–303 dependencias, 305
experiencia de trabajo,
306–307 experiencia trabajando en un equipo
de pruebas en alta mar, 315–317
facilitar la comunicación en línea, 306 integración,
308–309 problemas de
idioma, 303–304 problemas de
gestión, 317–318 pruebas en el
extranjero, 312–317 descripción
general de, 299–302
sesiones de planificación, 305, 308
razones para, 301–302
ciclos cortos de retroalimentación, 321–
322 problemas de zona horaria,
304–305 herramientas
para, 319–322 visualización
de pruebas, 386 Documentación. Véase también *Documentación*
viva de los entregables, 92–
94 excesiva, 341–342
mito de la “falta de documentación”, 339–340 vínculo
con los requisitos, 345

- Documentación (continuación) que
- registra los resultados de las pruebas exploratorias, 185
 - requisitos de habilidades, 141–142
 - capacitación de usuarios finales, 295
- Conocimiento del dominio, 46–47
- Lenguaje de dominio específico (DSL) que
- colabora con equipos distribuidos, 311 que describen el comportamiento de las funciones, 147
 - ejemplos de uso y 157–158 que guían el desarrollo con pruebas orientadas al cliente, 239 aceptación
 - del programador y 160 pruebas que utilizan, 56 en un enfoque de equipo completo, 259
- Hecho
- Característica terminada, 286 Producto terminado, 287 Lanzamiento terminado, 342
- Historia terminada, 286 Principio de no repetirse (DRY), 35, 240 Impulsando el desarrollo. Consulte Ejemplos que guían el desarrollo de herramientas
- de colaboración de Dropbox, 312 experiencia trabajando en un equipo de pruebas en el extranjero, 315–317
- Principio SECO (No te repitas), 35, 240 DSL. Ver Lenguaje específico de dominio (DSL)
- DTS (Sistemas de seguimiento de defectos), 321
- DW. Ver Almacenes de datos (DW)
- Y
- Elaboración, tipos de habilidades de pensamiento, 49.
 - Eliot, Set, 200–201
 - El correo electrónico, como herramienta de comunicación, 319
 - Pruebas exploratorias asistidas por automatización de software/sistemas integrados (Agar), 334–335 naturaleza crítica de las pruebas en, 328–329 definido, 325 aprendizaje de pruebas ágiles para software móvil (Harrison), 336–337 lecciones aprendidas en la aplicación de pruebas ágiles (Hagar), 330–332 descripción general de, 325–326
- similitudes y diferencias de las pruebas ágiles en, 326–328
- estrategias de automatización de pruebas (Morgan), 332–334 contexto de prueba y, xxvi tipos de enfoques ágiles utilizados con, 329
- Emery, Dale, 248
- Emociones, aprendizaje y, 70–71
- Empatía, dar/recibir feedback y, 45
- Actividades
- de finalización del juego, 285 pruebas adicionales durante, 90 abordan problemas de integración tempranamente, 291 definidos, 417 se lanzan al producto Listo, 287
- Pruebas de un extremo a otro
- Ruhland on, 193 susceptibilidad al cambio, 229–230 pirámide de automatización de pruebas, 225
- Grupo de soluciones empresariales (ESG), en Dell desafío de la matriz de compatibilidad de hardware, 369, 371 desafíos en la transición a ágil, 266–267, 279–280
- resultados de adoptar ágil, 296
- Empresas (grandes organizaciones) que alinean el establecimiento de objetivos para la automatización de pruebas en todo, 277–278
- Big Data como desafío para, 357 coordinar múltiples equipos, 283–284 coordinar herramientas para, 289–290 participación del cliente, 294–296 definir, 417 gestión de dependencia, 292 implementar automatización de pruebas en, 254–258 controles organizacionales, 278–283 descripción general de, 275 entrega de productos y 296–297 automatización de escalado para, 264–265, 267–268 pruebas de escalado para, 276–277 equipo y entorno de prueba del sistema y 284–289 cobertura de prueba, 291 asociaciones con terceros, 292–294 versión controlar, 290 qué son, 275–276
- Ambientes
- en 7 dimensiones del producto, 130

- en DevOps, 376
entornos de desarrollo, 59 a 60 pruebas de funciones realizadas por equipos de prueba de 286 sistemas y 284 a 289 entornos de prueba. Ver entornos de prueba ESG. Consulte Enterprise Solutions Group (ESG) en Dell ETL. Ver Extraer, transformar y cargar (ETL)
- Evangelisti, Augusto, xxix sobre la sección de contribuyentes, xxxvi sobre estrategia de ramificación, 63 sobre valor de conversación, 150 sobre gestión de evaluadores, 19 sobre gremios de calidad, 14 sobre reducción de deuda defectuosa, 213–216
- Evans, David sobre la sección de contribuyentes, xxxvi sobre automatización pruebas como documentación viva, 339 sobre cómo conocer a los miembros del equipo distribuido, 310 modelo de "Pilares de las pruebas", 402–405 sobre cómo dedicar el tiempo de manera efectiva, 9 Secuencias de comandos diarias con Ruby (Marick), 57 ejemplos que guían el desarrollo ATDD, 149–150 BDD, 152–153 beneficios de, 157–158 capturar ejemplos y convertirlos en pruebas automatizadas, 147 "ejemplos verificados" en las pruebas del primer y segundo trimestre, 103 colaborar con equipos distribuidos, 311 conversación como medio para entregar valor, 150–152 como práctica central, 394–395 en prevención de defectos, 158 evolución de las pruebas ágiles y, 3 descripción general de, 55–56, 145 dificultades de, 159– 162 planificación de pruebas Q2 y, 105 poder de uso, 145, 147–148 materiales de recursos para aprender mecánica de, 162 SBE, 153–154 resolución de problemas de datos de prueba incorrectos, 354 uso en la vida real, 146–147 varios enfoques, 148–149 dónde conseguirlos, 155–157
- Ejecutivos, educación, 17–19 Experimentos de aprendizaje y, xxviii pequeños experimentos de aprendizaje continuo, 80 Pruebas exploratorias en un contexto de pruebas ágiles, 188–190 asistidas por automatización, 334–335 colaboración con el equipo de desarrollo y, 345 como práctica central, 395–396 creación de cartas de prueba, 94, 168–171 definidas, 417 en DW/BI, 350 en evolución de las pruebas ágiles, 5 generación de ideas para cartas de prueba, 171 grupos en, 183–185 ayudando a otros y, 80 Hendrickson en adelante, 66 a 67 viajes en, 175 a 176 administrando vuelos de prueba, 176 equipos de prueba en el extranjero y 315 descripción general de, 165 a 167 personas en, 171 a 174 Pruebas del tercer trimestre (pruebas orientadas al negocio) y, 66, 103, 106–107 registrar resultados, 185–188 en SBTM, 176–178 desarrollo de habilidades para, 167 requisitos de habilidades en equipos de forma cuadrada, 30–32 historias en, 175 en TBMT, 178–183 en pirámides de automatización de pruebas 223, 227–228 recorridos en, 174–175 Explorarlo (Hendrickson), 50, 66, 106, 168 Extraer, transformar y cargar (ETL) definido, 417 aprender a probar BI y, 351 administrar datos de prueba y, 355–356 acelerar/automatizar, 349 probar reglas de datos, 353–354
- Fabricación extrema, 331 Programación extrema (XP) cuadrantes de pruebas ágiles y, 101 evolución de las pruebas ágiles y, 3–4 necesidades de pruebas y, 192

F

- Habilidades de facilitación, habilidades de pensamiento, 42–43
- Patrón de fábrica, patrones de diseño, 410–412
- Fallar rápido, aprender y, 12
- Farley, David, 361
- Comentarios rápidos, principios ágiles
- Pruebas de nivel API y, 223
 - que se aplican a DW/BI, 351–352
 - pruebas automatizadas y, 97, 153, 161, 227 niveles de automatización y, 115
 - Pruebas A/B y 204
 - procesos de construcción y 416
 - DevOps y, 362 en
 - equipos distribuidos, 301
 - aprendiendo de, 12, 87, 394 como objetivo de ejecución de pruebas, 256
 - priorizando información de, 347 para mejora y visibilidad del producto, 17 proporcionando información a desarrolladores, 331 ciclos de lanzamiento rápido y, 200 en regresión fallas, 234, 289 para reducción de riesgos, 291 entornos de prueba y, 60 en enfoque de prueba y codificación, 354
- "Trinity Testing" (Harty) y, 184 de usuarios, 327
- Fazal, Kareem
- sobre la sección de contribuyentes, xxxvi sobre la automatización de las pruebas de verificación de compilación, 369–371
- Cambio intrépido (Manns y Rising), 19
- Función terminada, 286
- Inyección de funciones, BDD y 153
- Presentar pruebas
- como práctica central, 396–397
 - en diferentes entornos, 286 pruebas iterativas y retroalimentación rápida y, 5 planificación de lanzamiento y, 92 recordar el panorama general, 396–397 mapas de historias en, 126
- Características
- pruebas exploratorias a nivel de característica, 189
 - niveles de precisión en la planificación, 92–94, 96
 - descripción general
 - de, 88 planificación de pruebas Q2 y, 105 planificación de pruebas Q3 y, 106–107
- priorización, 283 el "por qué" de las nuevas funciones, 121–123
- datos federados, 348, 417
- ejercicios de "envoltura de retroalimentación" (Appelo), 45 integración continua de bucles
- de retroalimentación/retroalimentación y 262 fallas rápidas y
 - 12 retroalimentación rápida. Consulte Retroalimentación rápida, mapeo del impacto de los principios ágiles y 125 factores clave de éxito, 393 sistemas móviles e integrados y 326 equipos de prueba externos y 316–317 cultura organizacional y 15–17 planificación para, 87 de creación de prototipos, 205–206 pruebas de regresión. y 161 ciclos cortos de retroalimentación en pruebas ágiles, 321–322 habilidades para dar/recibir, 45–46 pirámide de automatización de pruebas y 223 diagramas de espina de pescado. Consulte los diagramas de Ishikawa, 44 FIT (Marco para pruebas integradas), 3 herramienta FitNesse, 254–255, 257, 265 Beneficios de los diagramas de flujo de los sistemas basados en flujo, 383 uso de ejemplo en, 145 kanban y, 88 nivel de piso y, 96
- Grupos focales, 140
- Marco para pruebas integradas (FIT), 3
- Freeman, Steve, 126
- Frempong, Benjamin sobre la sección de contribuyentes, xxxvii sobre la automatización del aprovisionamiento de estados base de configuración, 374–376
- Descomposición funcional, tipos de habilidades de pensamiento, 49.
- Pruebas funcionales
- ATDD y, 150 en la pirámide de automatización de pruebas ampliada, 233–234 en la pirámide de automatización de pruebas, 115 herramientas para, 237 convertir ejemplos en pruebas automatizadas, 147
- Componentes
- clave de funcionalidad del valor comercial del software, 122
 - pruebas a través de la interfaz de usuario, 243
- Modelo FURPS, para pruebas de planificación, 113–114

G

g11n. Véase Globalización

Gärtner, Markus, xxix sobre

ATDD, 56 sobre

informes posteriores a las giras, 175 Pomodoro

Testing, 182–183 sobre cómo iniciar

correctamente el proceso de prueba, 239 sobre

deuda técnica, 211 Gawande,

Atul, 9 Especialistas en

generalización, 33–36 George, Chris sobre

la sección de

contribuyentes, xxxvii sobre la reducción de la deuda

técnica, 218–220 Gilb, Kai, 107 Gilb, Tom, 107,

277 Herramienta

Git, 365–366 Globalización

desafíos de los mercados globales, 195

definidos, 415

guiando el desarrollo con ejemplos, 55–56, 148–150

soporte de lenguaje y juego de caracteres, 198 pruebas de planificación del cuarto trimestre

y 107 aceptación del programador y 160

Objetivos

en proceso de colaboración, 52

métricas en configuración, 277

SMART, 49, 386 partes

interesadas en el proceso de establecimiento de objetivos, 124

Gorman, María, xxx

7 Dimensiones del producto, 129, 131–133 sobre

la sección de contribuyentes, xxxvii plantilla

dada_cuando_entonces para obtener ejemplos, 156

sobre Planguage,

107 sobre la

representación de atributos de calidad, 191

GoToMeeting, herramienta de colaboración, 312

Sierva de Dios, Elena, xxx

7 Dimensiones del producto, 129, 131–133 sobre

la sección de contribuyentes, xxxvii plantilla

dada_cuando_entonces para obtener ejemplos, 156

sobre Planguage,

107 sobre la

representación de atributos de calidad, 191

Gráficos para registrar los resultados de las pruebas, 187

Chats grupales, herramienta de colaboración, 321

Pruebas de

conurrencia de abrazos grupales y, 194–195

pruebas exploratorias y, 183–184 Grupos, en

pruebas exploratorias, 183–185 Invitado, David, 29

Desarrollo orientador.

Ver ejemplos que guían el desarrollo.

h

Hagar, Jon

sobre la sección de contribuyentes, xxxviii sobre

pruebas exploratorias asistidas por automatización, 334–335

definición de software integrado y aplicaciones móviles, 325 sobre

lecciones aprendidas aplicando pruebas ágiles a sistemas móviles e integrados, 330–332

Hagberg, Jan Petter, 303, 322

Matriz de compatibilidad de hardware, en Dell, 369

Hariprasad, Parimala sobre

la sección de contribuyentes, xxxviii sobre equipos

de pruebas en alta mar, 315–318

Harrison, JeanAnn, xxix sobre

la sección de contribuyentes, xxxviii sobre la colaboración para satisfacer las necesidades regulatorias, 344–346

sobre el aprendizaje de pruebas ágiles para software móvil, 336–337

Harty, Julián, 330

sobre la naturaleza crítica de las pruebas para aplicaciones móviles, 328 sobre “Trinity Testing”, 184

Hassa, cristiano, 276–277

Heinrich, Mike sobre

la sección de contribuyentes, xxxviii ejemplo de configuración de almacén de datos:

Figura 22-1, 348

sobre cómo aprender a probar BI, 351–352

Heinze, Jerez, xxix

acerca de la sección de colaboradores, xxxix sobre

ejemplos de uso en la vida cotidiana, 146–147

Hellesøy, Aslak, 56

Persona de la mesa de ayuda, 172–173

Hendrickson, Elisabeth, xvii, 109, 214

“Prueba ácida ágil”, 79

ATDD y 151 uso

desafiante de cuadrantes, 112–113

- Hendrickson, Elisabeth (continuación)
- Ciclo "Discutir-Destilar-Desarrollar-Demostrar", 276 evolución
 - de las pruebas ágiles y 3 sobre pruebas
 - exploratorias, 66–67 sobre la planificación
 - de pruebas del tercer trimestre, 106–107
 - sobre recursos para buenos estatutos, 168
 - "Hoja de referencia sobre heurística de prueba" como fuente de ideas en pruebas exploratorias, 167
 - sobre herramientas para el pensamiento estructurado y enfocado, 50
- Heurística
- definidos, 418 en
 - pruebas exploratorias, 166 modelos de planificación y, 114
- Heusser, Mateo
- acerca de la sección de contribuyentes, xxxix sobre estatutos y pruebas basadas en sesiones, 169–170 sobre la gestión de pruebas de regresión, 177–178
- Contratación y búsqueda de las personas
- adecuadas, 36–37 proceso de incorporación de evaluadores, 37–38
- Contratación de geeks que encajen (Rothman), 37
- Cómo prueba Google el software (Whittaker), 109 "¿Cómo?" preguntas
- en la creación de hojas de ruta, 123–124 Ejemplo de iPhone, 140
- Humilde, Jez, 361
- "Joroba de dolor", en automatización de pruebas, 237
- Caza, Andy, 49–50
- Hussman, David, 71 años
- sobre viajes como medio para crear cartas, 175–176
 - uso de persona por, 171
- Hüttermann, Michael, 372
- acerca de la sección de contribuyentes, xxxix
 - sobre cómo agregar infraestructura al alcance de las pruebas, 365–367
 - sobre DevOps, 361
 - sobre "colaboración sin barreras y de afuera hacia adentro" en Q pruebas, 103–104
- Hipótesis, pruebas A/B para comprobar, 134
- |
- i18n. Ver Internacionalización
- IaaS (Infraestructura como servicio), 418 IDE. Ver entornos de desarrollo integrados (IDE)
- Descripción general del mapeo de impacto de, 123–126 para la visión del producto, 89 herramientas para la visualización del proceso de pensamiento, 43–44
- Infraestructura
- agregar infraestructura al alcance de las pruebas, 365–367
 - "Niveles de servicio" proporcionados por probadores de infraestructura, 371–372
 - soporte para equipos distribuidos, 322 pruebas, 372–373 requisitos de prueba y, 192
- Infraestructura como servicio (IaaS), 418
- Iniciativa, probadores tomando, 142–143
- Colaboración en entornos de desarrollo integrados (IDE) y, 56, gestionar la automatización con, 262 experiencia del probador con herramientas IDE, 60
- Habilidades en forma de T y, 29
 - pruebas unitarias en, 4
- Integración que
- aborda los problemas de integración desde el principio, 291 de equipos distribuidos, 308–309 nivel de lanzamiento del producto y 90 equipo de integración de sistemas en pruebas en el extranjero, 313
- Pruebas de integración, pruebas en el extranjero y 312–314
- Comprobaciones de integridad, de datos y relacionales, 59
- Interfaces
- en 7 dimensiones de producto, 129 a 130
 - habilidades técnicas y 58
 - interfaces de usuario. Ver interfaz de usuario (UI)
- Internacionalización
- planificación de pruebas del cuarto trimestre y 107 necesidades de pruebas y 195–200 Interoperabilidad, planificación de pruebas y 91 pruebas de investigación
 - pruebas A/B. Ver pruebas A/B
 - pruebas de concurrencia, 194–195 pruebas exploratorias. Consulte Internacionalización y localización de pruebas exploratorias, descripción general de 195–200, 163 pruebas de regresión. Consulte Tipos de pruebas de regresión, 164 pruebas de aceptación del usuario. Ver aceptación del usuario Pruebas de experiencia de usuario (UAT). Ver Experiencia de usuario (UX)

- Ishikawa diagrama tipos
de habilidades de pensamiento, 49
para visualizar el proceso de pensamiento, 44
- Departamentos de TI, responsabilidad al separarse del desarrollo, 365
- Iteraciones
demonstraciones de iteración como oportunidad de aprendizaje, 77 gestión de Big Data, 358 planificación, 88, 93, 96–97, 130, 293, 310 reuniones de planificación, 173–174, 186 pruebas de regresión y 98 historias y 28 señales visuales de deuda técnica, 217
- Ivarsson, Anders, 297
- J JavaScript, pirámide de automatización de pruebas y, 226 JBehave, para automatizar pruebas de aceptación, 214 biblioteca Jdefault, 250 herramientas Jenkins en el ejemplo de DevOps, 366 infraestructura de prueba con, 373 responsabilidades laborales. Ver Roles/competencias; Probadores, determinación de las responsabilidades laborales Johnson, Karen N., 330 Jones, Griffin sobre la sección de contribuyentes, xxviii sobre la posibilidad de agilidad en entornos regulados medio ambiente, 343 Viajes, en la creación de cartas de prueba, 175–176 Pruebas JUnit, 265
- K**
- Kahneman, Daniel, 50 años Kämper, Stephan acerca de la sección de contribuyentes, xl sobre infraestructura de pruebas, 372–373 Kanban para utilización de capacidad (Zheglov), 10–12 como herramienta de comunicación, 319 para mejora continua, 386–389 curso sobre, 75 equipos definidos, 418 distribuidos y, 311 como método basado en flujo, 88 para reducir la deuda por defectos, 214 uso en pruebas (Rogalsky), 382–385 para visualizar pruebas, 382–385
- Kaner, Cem, 399 Karten, Naomi, 49 Mantenerlo real, mejora y adaptación continua, 6 como práctica central, 401 tomar atajos y, 220 conseguir aceptación, 160 mantener pruebas de automatización, 248 hacer que los requisitos regulatorios formen parte del trabajo, 344 gestionar la fuente código, 64 opciones para lidar con la incertidumbre, 18 planificación y, 87 pruebas de regresión y, 200 pruebas, habilidades y recursos y, 104–105 creación de confianza dentro de los equipos, 45 ritmo insostenible y, 8 valor de que todo el equipo comprenda el entorno operativo, 376 “que” versus “por qué” en el desarrollo de funciones, 122 trabajar con terceros y, 293 Kelln, Nancy, 25 Keogh, Liz, xxix, 18 en BDD, 152 en implementar las capacidades necesarias antes de realizar la prueba, 239 en tener también muchas incógnitas, 161–162 sobre la revisión del flujo de trabajo para simplificar, 385 sobre pruebas para monitorear el rendimiento del sistema, 107–108 sobre el valor de las conversaciones, 147
- Kerievsky, Josué, 3 Khoo, Trish sobre la sección de contribuyentes, xl sobre la necesidad de evaluadores en desarrollo, 26–27 Kniberg, Henrik sobre el uso ágil de Spotify, 16 usando principios ágiles sin ser demasiado riguroso, 297
- Knight, Adam, xxix sobre la sección de contribuyentes, xl sobre estrategia de ramificación, 64–65 sobre entornos de desarrollo, 59 Pruebas DW/BI y, 350

- Caballero, Adam (continuación)
- experiencia en el uso de TBTM, 182 en
 - gestión de probadores, 19 en
 - equipos de forma cuadrada, 30–32 en
 - rendimiento y escala de pruebas, 357–359 en herramientas para extender los arneses de prueba, 237–238
- Kohl, Jonathan, 326, 330
- I**
- L10n. Ver localización
- Lambert, Rob, 29 Langr,
- Jeff Patrón
- Arrange-Act-Assert, 240 pruebas de aceptación automatizadas, 214 mnemónicos para requisitos no funcionales: FURPS, 113 Grandes organizaciones.
- Ver Empresas (grandes organizaciones)
- Habilidades de liderazgo, 74 sesiones de Lean Coffee, 14 implementación de principios Lean, 276 justo a tiempo y 85 en manufactura, 10 startups, 108–110, 134, 203 Aprendizaje. Véase también Cambio de aprendizaje continuo y, 9 conferencias, cursos, reuniones, campamentos de entrenadores y colaboración, 72–75 conocimientos de dominio, 46–47 evolución de las pruebas ágiles y, 5 experimentos y, xxviii fomento de una cultura de aprendizaje, 13–15, 77 ayudar a los demás, 79–80 la importancia de una cultura de aprendizaje, 12–13 dedicar tiempo a, 8–12, 77–79 mentores, aprender de, 71 descripción general de, 69 publicaciones, podcasts y comunidades en línea, 75–77 recursos, 72–74 roles y competencias y. Ver Roles/competencias en redes sociales, 74 estilos, 69–72 aprendizaje sorpresa, 80–83
- habilidades técnicas. Ver Conciencia técnica (habilidades técnicas) para evaluar la inteligencia empresarial, 351–352 habilidades de pensamiento. Consulte Habilidades de pensamiento Sistemas heredados definidos, 418 requisitos de internacionalización, 198 reducción de la deuda técnica, 218 pruebas basadas en sesiones y, 169 estrategia de prueba para, 177 Niveles de precisión (detalle), para la planificación que se aplica al ejemplo empresarial, 284 puntos de vista diferentes y, 87–89 nivel de característica, 92–94, 96 descripción general de, 87 ciclo de entrega del producto, 91–92 nivel de lanzamiento del producto, 89–90 pruebas de regresión y, 97–98 nivel de historia, 96 nivel de tarea, 96–97 visualizar lo que está probando, 98–100 escuchar aprender, 69 habilidades de pensamiento, 48–49 documentación viva. Consulte también Continuidad de la documentación proporcionada por el lenguaje específico del dominio, 239 SBE y, 153–154, 420 TDD y, 56 automatización de pruebas y, 38, 209, 268, 339, 342 pruebas a través de la API y 241–243 pruebas a través de la interfaz de usuario y, 243 modo L (lineal y lento). habilidades de pensamiento y, 49 pruebas de carga aplicando pruebas Q4, 103 entornos de prueba, 61 localización planificación de pruebas del cuarto trimestre y, 107 necesidades de pruebas y, 195–200 Archivos de registro, lectura, 59 burbujas lógicas, habilidades de pensamiento y, 50 Lyndsay, James, xxix, 178 sobre la amplitud y especificidad de las cartas de prueba, 169 recursos para pruebas basadas en sesiones, 178 sobre pruebas escritas versus pruebas exploratorias, 166

- METRO
- Maassen, Olav, 123
- Mantenibilidad, en la pirámide ampliada de automatización de pruebas, 233–234
- Mantenimiento
- hacer visible el mantenimiento de las pruebas, 213 de las pruebas, 248–251 señales visuales de deuda técnica, 217 Maksymchuk, Cory sobre la sección de contribuyentes, xl sobre el uso de ejemplos para prevenir defectos, 158
- Management 3.0 (Appelo), 16
- Problemas de gestión, equipos distribuidos y, 317–318
- Manns, Mary Lynn, 19 Mapeo de impacto.
- Consulte Mapeo mental de mapas de impacto.
 - Consulte Mapeo mental, mapeo de relaciones, mapeo de 49 historias. Consulte Herramientas de mapeo de historias para un pensamiento estructurado y enfocado, 50 Marick, Brian, 103 sobre la automatización de pruebas de aceptación, 214 sobre la evolución de las pruebas ágiles y 3 sobre desarrollo basado en ejemplos, 145 sobre aprender a codificar, 57 sobre cuadrantes, 101 Martin, Robert C., (Bob) sobre cómo generar confianza, 17 sobre priorización de alto valor trabajo, 401 sobre reglas de código limpio, 35 Matrices agregar visibilidad a las pruebas y a los resultados de las pruebas, 390
- Matriz de capacidad de componente de atributo (ACC), 113–114
- monitoreo de riesgos y suposiciones, 92 matriz de prueba a nivel de versión, 99 Matts, Chris, 153 sobre opciones reales, 18 sobre la determinación de las opciones más valiosas, 123 McDonald, Mark P., 276, 292 McKee, Lynn, 25, 55
- McKinney, Drew acerca de la sección de contribuyentes, xl sobre investigación de usuarios, 205–206
- McMahon, Chris, 310
- Reuniones
- Herramienta de colaboración GoToMeeting, 312 para priorizar el trabajo, 364 para conseguir las personas adecuadas, 365 reuniones de pie, 15
- Melnick, Grigori, 214
- Métricas, en el establecimiento de objetivos, 277
- Meyer, Geoff, xxix, 287–288, 296–297 sobre la sección de contribuyentes, xli sobre el viaje ágil en Dell, 279–282 sobre el enfoque adecuado para la automatización de pruebas, 266–267
- Mapas mentales
- que agregan visibilidad a las pruebas y a sus resultados, 390–391 como herramienta de colaboración, 321 conversión de SBTM a TBTM, 180–181 para documentación de entregables, 94 mapeo de impacto desarrollado a partir de, 123 aplicaciones móviles y 327 para registrar resultados de pruebas exploratorias, 185 tipos de habilidades de pensamiento, 49 para visualización del proceso de pensamiento , 43 para visualizar lo que estás probando, 98–100
- Producto mínimo viable (MVP) definido, 418 visualizando, 126
- Ministerio de Pruebas, 327
- Declaración de misión, como fuente de estatutos, 169
- Pruebas exploratorias asistidas por automatización de aplicaciones/dispositivos móviles (Hagar), 334–335
- pruebas de concurrencia y, 194 naturaleza crítica de las pruebas en, 328–329 integración de hardware y sistemas operativos en, 344–345 aprendizaje de pruebas ágiles para (Harrison), 336–337 lecciones aprendidas en la aplicación de pruebas ágiles (Agar), 330–332 descripción general de, 325–326 similitudes y diferencias en las pruebas ágiles en, 326–328 automatización de pruebas, 244 estrategias de automatización de pruebas (Morgan), 332–334 planificación de pruebas y 91 tipos de enfoques ágiles utilizados con, 329

-
- Modelos, datos, 347
- Modelos, planificación de
- cuadrantes de pruebas ágiles y 101–105 alternativas
 - a los cuadrantes (Hendrickson), 112–113 desafíos a los cuadrantes, 108–111
 - Modelo FURPS y matriz ACC aplicados a, 113–114
- heurística aplicada a, 114 descripción general de, 101 pruebas
- Q1, 105 pruebas
- Q2, 105–106 pruebas Q3, 106–107 pruebas Q4, 107–108 automatización de pruebas y, 115–116 Monitoreo de habilidades
- técnicas generales, 59 gestión de Big Data, 359 memoria , CPU, registro y, 59 riesgos y suposiciones, 92 requisitos de prueba y, 192 Morgan, Jeff ("Cheezy") sobre la sección de contribuyentes, xii sobre la gestión de datos de prueba, 249–250, 355–356 sobre estrategias de automatización de pruebas para dispositivos móviles y sistemas integrados, 332–334 sobre pruebas de aplicaciones basadas en navegador, 244–245 Morville, Peter, 328 Moss, Claire sobre la sección de contribuyentes, xli sobre aprendizaje, 80–83 MVP.

Ver Producto mínimo viable (MVP)

norte

Convenciones de nomenclatura, estándares de prueba, 265

Lenguaje natural, uso en BDD, 152

North, Dan sobre

 - BDD, 152 evolución de las pruebas ágiles y, 3

Notas que registran los resultados de las pruebas exploratorias, 185–186

oh

Tiempo del ciclo OAT (prueba de aceptación operativa)

 - desde la demostración hasta, 387–389 definido, 419

Habilidades de observación, 48.

O'Dell, Chris sobre

 - la sección de contribuyentes, xlvi sobre cómo invertir una pirámide de automatización de pruebas invertida, 229–230 Pruebas en el extranjero. Ver también Equipos distribuidos

cuestiones culturales y, 303 experiencia trabajando en un equipo de prueba, 315–317 uso del idioma, 312 descripción general de, 301 pros y contras, 312–315 planes de prueba de una página, 94 tableros en línea, para visualizar pruebas para equipos distribuidos dispersos, 386 en línea comunidades, aprender de, 75–77 Cursos en línea, aprender de, 74 Juegos en línea, para la integración de equipos distribuidos, 309

Mapas mentales en línea, visualizar lo que está probando, 98–100 Herramientas de seguimiento en línea, administrar dependencias, 305 Proyectos de código abierto, como oportunidad de aprendizaje, 76 Tiempo del ciclo de pruebas de aceptación operativa (OAT) desde la demostración hasta, 387–389 definido, 419

Desarrollo de operaciones y. Consulte Responsabilidad de DevOps al separarse del desarrollo, 365

Tablero de opciones, para mostrar 7 dimensiones del producto, 132 OPV (opiniones de otras personas), habilidades de pensamiento y, 50 Oráculos, en pruebas exploratorias, 166 Organización de pruebas de regresión, 97 Cultura organizacional que educa a las partes interesadas, 17–19 fomentando una cultura de aprendizaje, 13 –15, 77 importancia de una cultura de aprendizaje, 12–13 invertir tiempo en el aprendizaje, 8–12 gestión de evaluadores, 19–20 descripción general de, 7–8 ciclos de transparencia y retroalimentación, 15–17 Organizaciones que se ocupan de controles organizacionales, 278–283 Nivel de Empresa. Ver Empresas (grandes organizaciones)

Puntos de vista de otras personas (OPV), habilidades de pensamiento y, 50

- Ottinger, Tim
- Patrón Organizar-Actuar-Afirmar, 240 sobre la automatización de pruebas de aceptación, 214 mnemónicos para requisitos no funcionales. FURPS, 113 Propiedad, equipo de prueba en alta mar y, 316
- PAQ
- Patrón de objeto de página que crea pruebas de UI mantenibles, 245 creación con la clase PageFactory , 410–412 ejemplo escrito con Selenium 2 (WebDriver), 407–410 implementación, 247–248 estrategias de automatización de pruebas para sistemas móviles e integrados, 332–334 pruebas a través de la interfaz de usuario y, 243–246 comprensión, 246–247 prototipos en papel. Consulte Paralelización de creación de prototipos, en la gestión de Big Data, 358–359 Patton, Jeff persona use by, 171 sobre mapeo de historias, 126–127 Pruebas de rendimiento DW/BI y, 357–359 en la pirámide de automatización de pruebas ampliada, 233–234 entornos de prueba, 61
- Personas creación de cartas de prueba, 171–174 desempeñando el papel de "mal cliente" en pruebas exploratorias, 166–167 en mapeo de historias, 128 diseñadores de experiencias de usuario creando, 396
- Modelo "Pilares de prueba" (Evans), 402–405
- Proyectos piloto, para solucionar problemas, 280
- Lenguaje (Gilb y Gilb), 107, 277
- Planificación aplicada al ejemplo empresarial, 284 equipos distribuidos y, 305, 308 ejemplo de plan de prueba ligero (BMI calculadora), 95–96 características, 92– 94 retroalimentación/bucles de retroalimentación, 87 identificación de necesidades de productos, 129–131 planificación de iteraciones, 93, 96– 97, 130, 293, 310 reuniones de planificación de iteraciones, 173–174, 186
- "Niveles de Servicio" proporcionados por probadores de infraestructura, 371 modelos. Consulte Modelos, puntos de vista de planificación en, 87–89 a nivel de producto, 286 autor de preguntas y, 25 pruebas de regresión y, 97– 98 planificación de lanzamiento, 89–92 preparación de la historia, planificación previa, 96, 293 nivel de tarea, 96– 97 enfoque visual (pizarra), 81–82 visualizar lo que está probando, 98–100
- Plataforma como servicio (PaaS), 419
- Juego, como técnica de aprendizaje, 73–74 Plunkett, Brian, 280 POC (prueba de concepto), 64 Podcasts, aprender de, 75–77 Puntos de vista, en la planificación, 87–89 Pruebas Pomodoro (Gärtner), 182–183 "El poder de tres" (Dinwiddie), 4 Práctica, aprendizaje y, 74 Pensamiento y aprendizaje pragmático (Hunt), 49 Priorización de trabajos pendientes, 173– 174 de trabajos pendientes de defectos, 81 de correcciones de errores/defectos, 289 de características, 46, 159, 283 de resultados, 17 de historias, 127–128, 293 de cartas de prueba, 396 como habilidad de pensamiento para realizar pruebas, 41, 49 Privacidad, cuestiones relacionadas con los datos, 353
- Curso de Liderazgo en Resolución de Problemas (PSL), 43, 74
- Habilidades para resolver problemas, habilidades de pensamiento, 43–44
- Producto hecho, lanzamiento a, 287
- Lanzamiento del producto definido, 419 empresas y, 296–297 integración de pruebas exploratorias en pruebas ágiles, 189 descripción general de, 88 planificación, 89–92 cumplimiento normativo como parte de la versión Hecho, 342 ciclos de entrega de productos de uno o varios equipos, 91– 92

-
- Productos
- hoja de ruta, 90
 - planificación de pruebas a nivel de producto, 286 Creación de perfiles, habilidades técnicas generales, 59
 - Aceptación de los programadores y, 160–161 en el equipo de pruebas del sistema, 284 Programación, 57.
 - Véase también Código/codificación
- Prueba de
- concepto (POC), 64
 - Creación de prototipos ejemplo de uso en, 145 uso en pruebas, 205–206 diseñadores evaluadores de UX que usan, 140 "Iniciadores de provocación" (Vaage), 71–72, 413–
 - 414 Pryce, Nat, 126 Pseudocódigo, automatización de pruebas y, 57 PSL (Liderazgo en resolución de problemas), 43, 74 Publicaciones, aprendizaje de, 75–77
 - Herramientas Puppet, en el ejemplo de DevOps, 365–367
 - Poniéndolo todo junto prácticas de fomento de la confianza, 394 sensibilidad al contexto, 399–400 aprendizaje continuo, 397–398 creación visión compartida, 402 uso de ejemplos, 394–395 pruebas exploratorias, 395–396 pruebas de características, 396–397 mantenerlo real, 401 modelo "Pilares de pruebas" (Evans), 402–405 siete factores clave de éxito, 393 Pirámide, automatización de pruebas versiones alternativas, 224–227 peligros de retrasar la automatización de las pruebas, 227–229 evolución de las pruebas ágiles en Dell, 282 ejemplo de pirámide expandida, 231–234 ejemplo de invertir una pirámide invertida, 229–231 versión original, 223–224 descripción general de, 223 planificación para la automatización de pruebas, 115–116 que muestra diferentes dimensiones, 231, 234
- Q**
- Herramienta qTrace, para grabar, 187–188
 - Cuadrantes, alternativa de prueba ágil a (Hendrickson), 112–113
- Pruebas A/B en el segundo trimestre. Consulte los desafíos de las pruebas A/B, 108–111 DevOps trabajando en todos, 363–364 evolución de las pruebas ágiles en Dell, 282 orden de aplicación, 103–105, 191, 239 descripción general de, 101–103 pruebas del primer trimestre, 105, 145 Pruebas del segundo trimestre, 105–106, 145 pruebas del tercer trimestre, 106–107 pruebas del cuarto trimestre, 107–108, 268 deuda técnica debido a omitir las pruebas del tercer y cuarto trimestre, 211 atributos de calidad de las pruebas, 65–67 UAT en el tercer trimestre, 201 Equilibrio de calidad en muchos aspectos de, 135 DevOps y, 363 gremios y, 14 atributos de calidad en 7 dimensiones del producto, 130 en pruebas de aceptación, 150 representación en operaciones y dimensiones de desarrollo, 191 pruebas, 65–67 lenguajes de consulta, requisitos de habilidades, 59 cuestionamientos (herramientas) herramientas para la visualización del proceso de pensamiento, 44 "¿Por qué?", "¿Quién?", "¿Cómo?" y "¿Qué?", 123–124 Teoría de colas, utilización de la capacidad y, 10–12
- R**
- Rainsberger, JB, 147 Rall, Aldo, xxix sobre la sección de contribuyentes, xli–xlii sobre cómo ayudar a otros, 79 Rasmussen, Jonathan, xxi Real Options (Matts y Maassen), 18 Verificaciones de la realidad. Consulte Mantenerlo real Grabar ideas de refactorización, en pizarras, 230 resultados de pruebas exploratorias, 185–188 Refactorización definida, 419 escalar la automatización para grandes organizaciones, 268 pizarra para registrar ideas, 230

- Reflexionar y adaptar (Hendrickson), xviii
- Desafíos de las pruebas
- de regresión, 200–201 integración
 - continua con pruebas automatizadas, 289 definidos, 419
- Creación/implementación de roles de DevOps, 376
- DW/BI y, 349 en
- pirámide ampliada de automatización de pruebas, 234 fracasos, 62 factores
- clave de éxito, 393 gestión con estatutos SBTM, 177–178 manual, 213, 223 planificación, 97–98 escalamiento para grandes organizaciones, 267–268 enfoque de equipo al mayor problema, 217–218 deuda técnica, 216–217 pruebas a través de la interfaz de usuario y, 243, 245 demasiadas, 161
- Entornos regulatorios uso ágil en entornos regulados, 343 colaboración para satisfacer las necesidades regulatorias, 344–346
- cumplimiento y, 340–341 documentación excesiva y, 341–342 incluir auditores en la solución, 342 mito de la “falta de documentación”, 339–340 descripción general de, 339
- Mapeo de relaciones, tipos de habilidades de pensamiento, 49
- Habilidades de relación, 74
- Liberar candidatos
- creando continuamente, 365 definidos, 419 niveles de lanzamiento de productos y, 189
- Liberación realizada, cumplimiento normativo como parte de, 342
- Sesiones de planificación de lanzamientos, 92
- Fiabilidad, en la pirámide ampliada de automatización de pruebas, 233–234
- Toma de requisitos, 159
- Recursos para conferencias de aprendizaje, cursos, reuniones y colaboración, entre 72 y 75 publicaciones, podcasts y comunidades en línea, 75–77
- Retrospectivas para abordar la deuda técnica, 217 herramientas de colaboración, 322 poderes de, 13 equipos autogestionados y, 24
- Retorno de la inversión (ROI) definido, 419 componentes clave del valor comercial del software, 122 cumplir con los desafíos de automatización y, 258 pirámide de automatización de pruebas y, 115 planificación de pruebas y, 91 pruebas a través de la interfaz de usuario y, 243
- Ries, Eric, 134
- Levantándose, linda
- Fearless Change (Manns and Rising), 19 sobre el poder de las retrospectivas, 13
- Pruebas basadas en riesgos, 51
- Riesgos, pruebas, 125
- Modo R (no lineal, rápido, “rico”), habilidades de pensamiento y, 49
- Hojas de ruta, preguntas en la creación de, 123–124
- Robson, Sharon, xxix sobre la sección de contribuyentes, xlvi agregar dimensiones a la pirámide de automatización de pruebas, 231–232
- sobre la aplicación de habilidades de pensamiento a las pruebas, 49–51
 - sobre la colaboración eficaz, 52–53
- Rogalsky, Steve, xxix sobre la sección de contribuyentes, xlvi sobre aprendizaje, 79 sobre podcast como recurso de aprendizaje, 75–76 sobre mapeo de historias y pruebas, 127–129 sobre el uso de kanban en pruebas, 382–385
- Retorno de la inversión. Ver Retorno de la inversión (ROI)
- Ventajas de roles
- competencias de algunos roles borrosos en los equipos, 142 competencias versus roles, 24–28 especialistas en generalización, 33–36 contratación de las personas adecuadas, 36–37 proceso de incorporación para evaluadores, 37–38 descripción general de, 23 equipo de forma cuadrada y 30–32 conjunto de habilidades en forma de T y 28–30 importancia de los títulos (Walen), 25

- Análisis de causa raíz, herramientas para la visualización del proceso de pensamiento, 44
- Rothman, Johanna, xix sobre ciclos de retroalimentación y transparencia, 16 sobre la contratación de personas adecuadas, 37
- Enfoque RSpec, para el desarrollo basado en pruebas, 259
- Rubin, Ken, xxix
- Biblioteca Ruby Faker, 250
- Ruhland, Bernice Niel, xxix, 77 sobre la sección de contribuyentes, xlvi sobre adaptación al cambio, 15 sobre el proceso de exploración grupal, 183 sobre el mantenimiento de una lista de tipos de pruebas para su equipo, 193 sobre planes de prueba de una página, 94 sobre el registro de resultados de pruebas exploratorias, 187 sobre el uso de SBTM para entrenar testers, 176 en equipos autogestionados, 23-24 en testers y analistas de negocios, 138
- S**
- SaaS (software como servicio) definido, 420 desarrollo de productos y, 202 Sandboxes, entornos de prueba, 60 Satir Global Network, 74 SBE. Ver Especificación por ejemplo (SBE)
- SBMT. Ver Gestión de pruebas basada en sesiones (SBTM)
- Escalar, en automatización de pruebas ágiles y, 358-359 consideraciones con respecto a, 275 ejemplo de Dell, 287-288 para empresas, 276-277 velocidad y, 357-358 escenarios creación de cartas de prueba, 174 uso de ejemplos y, 159
- Schoots, Huib sobre la sección de contribuyentes, xlvi sobre documentación, 94 experiencia trabajando en equipos distribuidos, 306-307
- Alcance agregando infraestructura al alcance de prueba, 365-367 derivando usando SBE, 153
- Scott, pirámide de automatización de pruebas alternativa de Alister, 227 sobre pruebas a través de la interfaz de usuario, 243
- Pruebas escritas, 166
- Equipos scrum/scrum Evolución ágil en Dell y, 282 en verificación automatizada de compilación, 370
- Extreme Manufacturing y, 331 primeras experiencias ágiles en comparación con, 382 pruebas ágiles de escala, 287-288 ciclos de sprint, 266-267 time-boxing, 281
- Problemas relacionados con los datos de seguridad, 353 personas preocupadas por la seguridad, 172 aplicando pruebas del cuarto trimestre, 103 atributos de calidad y, 65-66 requisitos de habilidades para los evaluadores de DW/BI, 348
- Selenio 2 (WebDriver) Ejemplo de objeto de página, 407-410 Clase PageFactory , 410-412
- Biblioteca de pruebas de selenio, 244, 254-255, 265
- Equipos autogestionados, Ruhland en adelante, 23-24
- Estudios Semánticos, 329
- Hojas de sesión, registro de resultados de pruebas exploratorias, 185
- Gestión de pruebas basada en sesiones (SBTM) que se convierte a TBIM, 179-181 gestión de pruebas de regresión con cartas SBTM, 177-178 gestionar los estatutos de prueba, 176, 178 los estatutos y las pruebas basadas en sesiones, 169-170 registrar los resultados de las pruebas exploratorias, 186
- Desarrollo basado en conjuntos, 245, 259, 420
- Shannon, Paul sobre la sección de colaboradores, xlvi sobre cómo invertir una pirámide de automatización de pruebas invertida, 229-230
- "Compartir el dolor", 213
- Orilla, James, 214
- Principio de simplicidad que se aplica a DW/BI, 351 que se aplica a la planificación, 88 pruebas automatizadas y 256 mapas mentales para la captura de detalles, 94 que revisa el flujo de trabajo para simplificar, 385

- Sinclair, Jennifer, xxix, xlivi
- Sinclair, Toby
- sobre la sección de contribuyentes, xlivi sobre pruebas A/B, 203–204
- Único, responsabilidad, abierto/cerrado, sustitución de Liskov, segregación de interfaz, Inversión de dependencia (SÓLIDO), 57
- Sjodahl, Lars, 304
- Habilidades
- habilidades de análisis empresarial, 137–139
 - actividades de DevOps y 363
 - habilidades de documentación, 141–142
 - inversión de tiempo en aprendizaje, 8
 - habilidades de pensamiento. Consulte Ejemplo de equipo en forma de cuadrado de habilidades de pensamiento, 30–32 formación de equipos y, 24, 27–28 habilidades técnicas, consulte Conciencia técnica (habilidades técnicas)
 - Conjunto de habilidades en forma de T, 28 a 30 habilidades de diseño de UX, 140 a 141 cortes
 - pruebas de funciones y, 396
 - pruebas, 127
- Principio de trozos pequeños
- aplicando a DW/BI, 351 build-measure-lean (BML) y, 134 estatutos, 170 equipos distribuidos y, 313
 - identificando fragmentos para pruebas, 127 para aprendizaje y adaptación, 87 para planificación, 87 historias y, 88, 103 para entrenamiento, 38
- INTELIGENTE (específico, medible, alcanzable, relevante y con un límite de tiempo)
- mejora continua y, 386 tipos de habilidades de pensamiento, 49 pruebas de humo, 263 cuestionamientos socráticos, tipos de habilidades de pensamiento, 49 habilidades blandas. Ver Habilidades de pensamiento Software como servicio (SaaS)
 - definido, 420
 - desarrollo de productos y, 202Ciclo de entrega de software, bucles de retroalimentación, 15 Ataques de prueba de software para romper dispositivos móviles y Dispositivos integrados (Agar), 330
- SÓLIDO: Único, Responsabilidad, Abierto/Cerrado, Sustitución de Liskov, segregación de interfaz, Inversión de dependencia (SÓLIDO), 57
- Resolver problemas, 43–44
- SonarQube, comprobando código comercial con 366 Sistema de control de código fuente, 62–65, 420 experiencia del evaluador, 60 control de versiones, 61 Cumplimiento SOX, entornos regulatorios, 340 Especialistas, especialistas generalizadores, 33–36 Especificación por ejemplo (Adzic), 56 Enfoques ágiles de especificación mediante ejemplo (SBE) utilizados con sistemas móviles e integrados, 329 prácticas de fomento de la confianza, 394 definidas, 420 herramientas de prueba funcionales, 237 desarrollo de guías con ejemplos, 55–56, 148, 153–154 uso para reducir la deuda por defectos, 214 en un enfoque de todo el equipo para enfrentar nuevos desafíos, 259
- Talleres de especificación, facilitación, 42 Spikes aplazar la planificación de pruebas hasta después de las soluciones de pico, 92 planificación y, 90 prueba de herramienta de automatización potencial con, 261 Prueba dividida. Consulte las pruebas A/B Spotify, 16, 297 Spott, Dave, 280 Ejemplo de uso de hojas de cálculo en, 145, 157 registros de resultados de pruebas exploratorias, 186 problemas con compilaciones de CI fallidas, 263 SQL (lenguaje de consulta estructurado), 59 Inyección de SQL, pruebas de seguridad, 65–66 Equipo de forma cuadrada, 30–32 Entornos de preparación, para pruebas, 60 Responsabilidad de las partes interesadas ante, 343 auditores como, 342 aceptación, 160–161 educar, 17–19 obtener ejemplos de, 147–148 dar/recibir retroalimentación, 46

- Talleres de especificación de partes interesadas (continuación) y 42 herramientas para la participación, 123–127
- Estándares, convenciones de nomenclatura, 265
- Reuniones stand-up, para comunicación cara a cara, 15
- Diagramas de transición de estados, 67.
- Hilos de acero, 160, 313
- Historias
- bloqueadas, 292
 - dividir funciones en, 92, 94 en entrenamiento, 48 para refactorización de código, 213 crear cartas de prueba a partir de, 175 determinar el propósito de, 121 pruebas de funciones, 94, 96 obtener ejemplos para, 155 guiar el desarrollo con ejemplos, 55 pruebas de iteración y, 28 en niveles de precisión, 89 niveles de precisión en la planificación, 96 descripción general de, 88–89 planificación de pruebas Q2 y, 105 planificación de pruebas Q3 y, 106–107 priorización, 127–128, 293 pruebas de regresión y, 97 – 98 enfoque en equipo para el mayor problema, 218
- Herramientas de comunicación de guiones gráficos, 319 visualización efectiva del proceso de prueba. 384–386
- conocer a los miembros del equipo distribuido, 311 Historia realizada frente a característica realizada, 286
- Mapeo de historias,
- obtención de ejemplos y, 155 mapeo de impacto desarrollado a partir de, 123 descripción general de, 126–127 pruebas y, 127–129 Pruebas de estrés
 - satisfacer las necesidades regulatorias y, 345 dispositivos móviles y, 345
- Lenguaje de consulta estructurado (SQL), 59
- Subversión, 64
- Éxito
- importancia de celebrar, 17 factores clave en, 393
- Sweets, Tony
- sobre la sección de colaboradores, xliii–xliv sobre el uso del patrón Page Object, 246–248
- Equipo de integración de sistemas, enfoques para pruebas en el extranjero, 313
- Equipo de prueba
- del sistema que coordina las dependencias entre equipos, 285–286
 - crear, 284–285 gestionar, 287
- Cambios y actualizaciones del sistema bajo prueba (SUT) y, 240, cuánta automatización es suficiente y, 263
- t**
- Tabaka, Jean, 71 años
- Tablas que registran los resultados de las pruebas exploratorias, 187.
- Charlas, Mike, xxix
- sobre la sección de contribuyentes, xliv sobre el aprendizaje constante, 78–79
 - sobre las desventajas de los tableros kanban, 384 ejemplo de la necesidad de un análisis empresarial en las pruebas, 138–139
 - sobre el aprendizaje del conocimiento del dominio, 46–47 sobre el proceso de incorporación para nuevos evaluadores, 38 Tareas que abordan las necesidades regulatorias y
 - del producto, 342 creación de tareas de prueba, 96–97 pruebas exploratorias con, 189 Taxonomía, Cuadrantes como, 104 TBMT. Ver Gestión de pruebas basada en subprocesos (TBMT)
- TDD. Ver Desarrollo basado en pruebas (TDD) equipos
- adoptar valores ágiles, 7
 - soluciones de automatización para equipos en transición, 253–254, 258 vincularse
 - con, 345 competencias
 - incluidas, 27 coordinar múltiples, 283–284 crear un equipo de prueba del sistema, 284–289 dependencias entre, 92 dispersos. Ver Equipos dispersos distribuidos. Ver equipos distribuidos

- encontrar/contratar a las personas adecuadas, 36–37 fomentar una cultura de aprendizaje, 13–15 especialistas en generalización y, 36 incluidos analistas de negocios, 129 multidisciplinarios, 256 ciclos de entrega de productos y 91 autogestionados (Ruhland), 23–24 cuadrados equipo formado, 30–32 habilidades técnicas y 56 comprensión de los problemas del equipo (Gregory), 18 enfoque de equipo completo. Ver todo el equipo Conciencia técnica (habilidades técnicas) habilidades de automatización y codificación, 56–58 automatización a través de la API, 240–243 automatización a través de la interfaz de usuario, 243 integración continua y control del código fuente sistemas, 62–65 entornos de desarrollo y, 59–60 pruebas DW/BI, 348 habilidades técnicas generales, 59 desarrollo guiado con ejemplos, 55–56 recursos de aprendizaje, 74 descripción general de, 55 pruebas de atributos de calidad, 65–67 técnicas de diseño de pruebas, 67 entornos de prueba, 60–62 Deuda técnica definida, 420 hacerla visible, 212–213, 216–217 descripción general de, 211–212 cuantificar el costo de, 17 reducir la deuda por defectos, 213–216 reducir mediante la colaboración, 218–220 enfoque de equipo a, 217–218, 220 volcán de automatización de pruebas y, 229 creación de plazos poco realistas, 7–8 Habilidades técnicas. Ver Conciencia técnica (técnica (habilidades)) Escritores técnicos, 141–142 Teletrabajo, creando políticas para, 310 Pruebas de aceptación de automatización de pruebas, 214 alinear el establecimiento de objetivos para la automatización de pruebas en toda la empresa, 277–278 pruebas exploratorias asistidas por automatización (Agar), 334–335 soluciones colaborativas en la selección de herramientas, 260–261, 264 integración continua con pruebas de regresión automatizadas, 289 gestión de datos, 249–250 patrones y principios de diseño, 240 entornos de desarrollo y, 59–60 ejemplos convertidos en pruebas automatizadas, 147 pruebas exploratorias utilizadas en conjunto con, 167 cuánta automatización es suficiente, 262–263 implementación en organizaciones grandes, 254–258 mantenimiento de pruebas, 248–251 para sistemas móviles e integrados, 332–334 descripción general de, 209–210 Patrón de objeto de página y, 246–248 Planificación , 115–116 enfoques piramidales. Ver Pirámide, automatización de pruebas Pruebas del cuarto trimestre (pruebas orientadas a la tecnología), 268 pruebas de regresión, 97 reglas y motivos, 241 escalamiento para grandes organizaciones, 264–265, 267–268 escalado para Big Data, 357–359 selección de soluciones de automatización de pruebas, 253 soluciones para equipos en transición, 253–254, 258 comenzando bien, 239–241 deuda técnica y. Consulte Habilidades técnicas de deuda técnica para, 56–58 entornos de prueba y, 61 pruebas a través de API (nivel de servicio), 241–243 pruebas a través de la interfaz de usuario, 243–246 asociaciones de terceros y, 293 herramientas y procesos en, 237–238 desafíos de transición y, 266–267 enfoque de equipo completo para, 238–239, 258–260 Cartas de prueba. Consulte Charters Cobertura de pruebas en empresas, 291 ejemplo de dispositivo médico, 344 Diseño de pruebas. Consulte también Técnicas de principios y patrones de diseño, 67 uso de diagramas para, 67 habilidades de diseño de UX, 140–141 diseñadores de UX que crean personas, 396

- Entornos de prueba. Consulte también Los entornos construyen canalizaciones y, 367 construcción y mantenimiento, 60–62 DevOps en el mantenimiento de, 362 haciendo visible, 291 descripción general de, 60–62 Depuración de fallas de prueba, 250–251 administración, 289–290 problemas de sincronización, 237 “Heurística de prueba Hoja de referencia” (Hendrickson), 114 Matriz de prueba. Ver Matrices Diseño basado en pruebas. Ver Desarrollo basado en pruebas (TDD)
- Desarrollo basado en pruebas (Beck), 9 Desarrollo basado en pruebas (TDD) BDD como respuesta a, 153 colaboración con equipos distribuidos, 312 prácticas de desarrollo centrales, 394 definidas, 421 ejemplos de DSL y 158 desarrollo basado en ejemplos en comparación con, 148 integración de codificación y pruebas, 105 en sistemas móviles e integrados, 326, 329 compra de programadores -en y, 160 reduciendo la deuda por defectos, 214 escalando, 276–277 a nivel de tarea, 96, 98 Probadores. Consulte también Roles/competencias que agregan valor a DevOps, 371–372 principios ágiles para, 28–29, 386 habilidades de análisis de negocios, 137–140 como comunidad de práctica (CoP), 80 habilidades de bases de datos, 348 determinación de responsabilidades laborales, 137 habilidades de documentación, 141–142 especialistas en generalización y 36 que dan/ reciben retroalimentación y 46 “Niveles de servicio” proporcionados por probadores de infraestructura, 371–372 gestión, 19–20 necesidad de desarrollo (Khoo), 26–27 proceso de incorporación para, 37–38 gremios de calidad (Evangelisti), 14 Uso de SBTM en la formación, 176 ejemplo de equipo de forma cuadrada, 30–32 en el equipo de prueba del sistema, 284 tomando iniciativa, 142–143 habilidades técnicas y colaboración, 56 Trío (desarrollador, probador, BA) en pruebas exploratorias, 184 habilidades de diseño de UX, 140–141 Pruebas de aceptación de pruebas . Ver Pruebas de aceptación Pruebas A/B Consulte Pruebas A/B Pruebas de BI. Consulte Análisis de negocios de pruebas de inteligencia empresarial (BI) que se superponen con, 137 pruebas de componentes, 224–225, 233–234 basadas en el contexto, 399 creación de tareas para, 96–97 creación de pruebas antes de la codificación, 96 tempranas, 134 pruebas de un extremo a otro . Consulte la evolución de las pruebas ágiles de un extremo a otro, 3–6, ejemplo de plan de prueba liviano (calculadora de IMC), 95–96 exploratorio. Consulte Funciones de pruebas exploratorias. Consulte Pruebas funcionales de pruebas de características. Ver pruebas funcionales integrando la codificación con, 105 investigativo. Consulte Aprendizaje de pruebas de investigación, 72–76 pruebas de carga, 61, 103 en el extranjero. Consulte Pruebas de rendimiento de pruebas en el extranjero. Consulte Pruebas de rendimiento Pruebas Pomodoro (Gärtner), creación de prototipos 182–183 en, cuadrantes 205–206. Consulte Cuadrantes, atributos de calidad de pruebas ágiles, 65–67 pruebas de regresión. Consulte los requisitos de las pruebas de regresión y el propósito de (Walen), 139–140 basadas en riesgos, 51 riesgos, 125 pruebas de escala, 276–277, 287–288 pruebas con guión, 166 basadas en sesiones, 169–170 cortes, 127 pruebas de humo, 263 como ciencias sociales (Walen), 25 entornos de puesta en escena para, 60 iniciar el proceso correctamente (Gärtner), 239

- mapeo de historias y, 127–129 pruebas de estrés, 345 pruebas como servicio, 314 habilidades de pensamiento aplicadas a (Robson), 49–51 a través de la API (nivel de servicio), 241–243 a través de la interfaz de usuario, 243–246 tipos de, 192–194 pruebas unitarias. Consulte Pruebas unitarias, pruebas de aceptación del usuario (UAT), 149–150: visualización de lo que está probando, 98–100 pruebas de flujo de trabajo. Ver Pruebas de flujo de trabajo Pruebas, en la empresa alinear el establecimiento de objetivos para la automatización de pruebas, 277–278 implementar la automatización de pruebas en, 254–258 escalar las pruebas, 276–277 equipo y entorno de pruebas del sistema y, 284–289 cobertura de pruebas, 291 Pruebas en producción (TiP) (Eliot), 200–201 Pruebas como servicio, 314 Entrenamiento de habilidades de pensamiento y escucha, 48–49 habilidades de colaboración, 52–53 diagramas para un pensamiento estructurado y enfocado (Robson), 50 conocimiento del dominio y, 46–47 pruebas exploratorias y, 165 facilitación, 42–43 dar/recibir retroalimentación, 45–46 recursos de aprendizaje, 75 habilidades organizativas, 51 descripción general de, 41–42 resolución de problemas, 43–44 pensar diferente, 49–51 Pensar rápido y lento (Kahneman), 50 Pensando para la acción (de Bono), 50 "El probador del pensamiento" (Hendrickson), 112 Problemas de compatibilidad con terceros, 178 bibliotecas de prueba, 244 Participación del cliente de proveedores externos en las empresas, 294–296 gestionar dependencias, 292–294 marcos de tiempo y expectativas de, 200 "por qué" del desarrollo de funciones y, 122 Gestión de pruebas basada en subprocessos (TBMT) que convierte de SBMT a, 179–181 representación fractal de, 182 descripción general de, 178, 181–183 "Tres amigos" (Dinwiddie), 4 Tucídides, por automatizar las pruebas de aceptación, 214 Limitaciones de tiempo, habilidades organizativas y, 51 Gestión del tiempo, dedicar tiempo al aprendizaje, 77–79 Zonas horarias comunicación y colaboración y, 309–310 estrategias de afrontamiento para equipos distribuidos, 309 problemas que enfrentan los equipos distribuidos, 304–305 Timeboxing en el proceso de colaboración, 53 pautas de Scrum y, 281 nivel de historia y, 96 planificación de pruebas y, 88 títulos, 25 herramientas . Véase también por tipos individuales Comité de herramientas de prueba funcionales de Agile Alliance (AA-FTT), 4, 153 herramientas analíticas, 327 "bake-offs" para pruebas, 261–262 herramientas de colaboración, 312, 322 enfoque colaborativo para la selección, 264 consistencia en la selección de, 265, 289 coordinación de herramientas, 289–290 para DevOps, 366, 373 para exploración temprano, 134 facilitando la comunicación en línea, 306–307 implementando Agile en Dell y 281 herramientas de grabación, 187–188 en automatización de pruebas, 237–238 experiencia en pruebas con herramientas IDE, 60 herramientas de pensamiento, 49–50 herramientas de videoconferencia, 315–317 para visualización del proceso de pensamiento, 43–44 Herramientas para la participación del cliente 7 Dimensiones del producto (Gottesdiener y Gorman), 129–133 mapeo de impacto, 123–126 descripción general de, 123 mapeo de historias, 126–129 Herramientas, para equipos distribuidos herramientas de colaboración, 319–321 herramientas de comunicación, 319

-
- Tours, creación de cartas de prueba, 174–175
- Transición, hacia soluciones ágiles de
automatización de desarrollo para equipos en transición, 253–254,
258 desafíos, 266–
267 Transparencia, en cultura
organizacional, 15–17. Véase también Visibilidad “Trinity Testing” (Harty),
184 Trust
- edificio, 17
- equipos autogestionados y, 24
- Colaboración de
habilidades en forma de T como medio para desarrollar, 344
desarrollar, 398
especialistas generalizadores y, 33 roles y
competencias y, 28–30
- Tung, Porcia, 73–74
- UAT . Ver Pruebas de aceptación de usuario (UAT)**
- UI. Ver interfaz de usuario (UI)**
- Incertidumbre, generación de confianza y 18
pruebas unitarias.
coherencia en la selección de herramientas y, 265
definidos, 421
DW/BI y, 349
Pruebas JUnit, 265
planificación anticipada y automatización, 105
en TDD, 98
pirámides de automatización de pruebas y 115, 224–225,
233–234
- Comandos de shell UNIX, requisitos de habilidades, 59
- Comentarios sobre
usabilidad en la pirámide de automatización de pruebas
ampliada, 233–234
de la creación de prototipos, 205–206
- Las pruebas de aceptación de las pruebas
de aceptación del usuario (UAT) contrastaron con 149–150 en
empresas (Bligh), 294–295 en la pirámide de
automatización de pruebas ampliada, 233–234 aprendiendo a probar
BI y 351–352 pruebas en el extranjero y 312–
314 descripción general de , 201–202 nivel
de lanzamiento del producto y,
90
- Pruebas del tercer
trimestre y, 107 valor de las conversaciones y, 151–152
- Experiencia de usuario (UX)
Pruebas A/B aplicadas a, 203
considerando el impacto de las partes interesadas en los objetivos
comerciales,
124 requisitos de habilidades de diseño, 140–141
dispositivos móviles y, 328–329 pruebas,
205–207
Diseñadores de UX creando personajes, 396
- Interfaz de usuario (UI)
que logra consenso para soluciones de automatización, 260–
261
colaboración para satisfacer las necesidades regulatorias, 344
coherencia en la selección de herramientas y, 265
ejemplo de beneficio de incluir el diseño de UX en las pruebas, 141
dispositivos
móviles y, 328 habilidades técnicas
y, 58 automatización de pruebas
y, 225–226, 287 en la pirámide de automatización
de pruebas, 223 –224 pruebas de proyectos móviles y
344 pruebas a través de la interfaz de usuario,
243–246
- Historias de usuarios. Ver Historias
Mapeo de historias de usuario (Patton), 127
usuarios
en 7 dimensiones del producto, 129 usuarios
finales de capacitación, 295
utilización, utilización de la capacidad, 10–12 UX. Ver
Experiencia de usuario (UX)
- V**
- Vaage, Carol, 71–72, 413–414 Herramienta
vagabunda, utilizada en el ejemplo de DevOps, 366 proveedores.
- Consulte Proveedores externos Control de
versiones en empresas,
290 Git y 365 sistemas de
control de código
fuente, 61–65, 420 Herramientas de colaboración de
videoconferencia, 319–320
trabajando en un equipo de prueba externo
y, 315–317 Máquinas virtuales, para ejecutar conjuntos de
pruebas, 262 Visibilidad que comunica la importancia de las
pruebas, 381–
382 diagramas para visualizar el proceso de pensamiento, 44 para la
mejora continua, 386–390

dar/recibir comentarios y, 46 kanban y, 382–385
 tableros en línea para equipos
 distribuidos o dispersos, 386 descripción general de, 381
 reducir la
 necesidad de control,
 16 guiones gráficos y, 384–386 de deuda
 técnica, 212–213, 216–217 de entornos
 de prueba, 291 de pruebas y resultados de pruebas,
 381, 390–392 trabajando en un equipo
 de prueba externo y, 316–317 Visión, creación
 compartida, 402 Ayudas visuales, 98–100 Aprendizaje visual, 69,
 81 Visualización. Consulte *Visibility*
 Volcán, automatización de
 pruebas, 228 Vuolii, Eveliina
 sobre la sección de contribuyentes, xlv
 sobre la introducción de la automatización
 en el sistema
 empresarial, 277–278

EN

Walen, Pete
 sobre la sección de contribuyentes, xlv sobre
 la importancia de los títulos, 25
 sobre los requisitos y el propósito de las pruebas,
 139–140
“Esqueleto andante” (Freeman y Pryce), 126 Walshe, Mary
 sobre la sección
 de contribuyentes, xliv–xlv sobre kanban para la
 mejora continua, 386–389 Navegadores web. Consulte
 la
 herramienta Browsers WebDriver,
 261 WebEx, trabajando
 en un equipo de prueba en el extranjero y, 315–317

Pruebas de fin de semana, 74
“¿Qué?” preguntas en la
 creación de hojas de ruta, 123–124 ejemplo de
 iPhone, 140 “qué” versus “por
 qué” en el desarrollo de funciones, 122 Whelan, Declan, 153
 Whittaker, James, 109
“¿Quién?” preguntas
 en la creación de mapas de impacto, 123–124 Ejemplo
 de mapa de impacto para iPhone, 140

Todo el equipo. Véase también
 DevOps logrando consenso para soluciones de
 automatización, 260–262
 enfoque para pruebas ágiles, 362 enfoque
 para la automatización de pruebas, 238–239 factores
 clave de éxito, 393 enfrentar los
 desafíos de la automatización, 228, 258–260 necesarios para las
 pruebas DW/BI, 348–349 comprender el entorno
 operativo, 376 “¿Por qué?” preguntas para construir lo correcto.
 Ver Desarrollo,

Construyendo el mapa de impacto
 de ejemplo de iPhone correcto, 140 preguntas
 sobre la creación de mapas de impacto, 123–124
 Wiedemann, Christin sobre
 la sección de contribuyentes, xlv sobre la
 conversión de SBTM a TBTM, 179–181

Wikis

que agregan visibilidad a las pruebas y sus resultados, 392
 soluciones de automatización en grandes
 organizaciones, 255
 herramientas de colaboración, 321
 herramientas de comunicación, 319
 integración de equipos distribuidos, 309 registros de
 resultados de pruebas exploratorias, 185 wikis de equipo, 36

Winterboer, Lynn sobre la
 sección de contribuyentes, xlv sobre cómo resolver
 el problema de los datos de prueba incorrectos, 353–354
 Ancho de banda WIP (trabajo en
 progreso) para pruebas y 401 kanban y 382

Pruebas de flujo de
 trabajo que automatizan el aprovisionamiento de la base de configuración
 estados, 374–376
 en pirámide de automatización de pruebas ampliada, 233–234 niveles
 para pruebas a través de la interfaz de usuario, 243 en
 pirámide de automatización de pruebas, 115–116
 Trabajo en progreso Ver WIP
 Talleres
 obteniendo participación en talleres de especificación,
 155 como recurso de
 aprendizaje, 74
 Wortel, Cirilo, 265 sobre la
 sección de colaboradores, xlv

- Wortel, Cirilo (continued)
sobre soluciones de automatización en grandes organizaciones,
254–258
- Wynne, comida, 56
- X
- XP. Ver Programación Extrema (XP)
- Zheglov, Alexéi
sobre la sección de colaboradores, xlvi
sobre la utilización de la capacidad, 10

Esta página se dejó en blanco intencionalmente.

informIT.com LA FUENTE DE APRENDIZAJE DE TECNOLOGÍA DE CONFIANZA



InformIT es una marca de Pearson y la presencia en línea de los principales editores de tecnología del mundo. Es su fuente de contenido y conocimientos confiables y calificados, brindando acceso a las principales marcas, autores y colaboradores de la comunidad tecnológica.

▲ Addison-Wesley

Cisco Press

EXAM/**CRAM**

IBM
Press.

QUE

PRENTICE HALL

SAMS

Safari
Books Online

Aprenda TI en InformIT

¿Busca un libro, un libro electrónico o un vídeo de formación sobre una nueva tecnología? ¿Busca información y tutoriales oportunos y relevantes? ¿Busca opiniones, consejos y sugerencias de expertos? InformIT tiene la solución.

- Obtenga información sobre nuevos lanzamientos y promociones especiales suscribiéndose a una amplia variedad de boletines.
Visite informit.com/newsletters.
- Acceda a podcasts GRATUITOS de expertos en informit.com/podcasts.
- Lea los últimos artículos de los autores y capítulos de muestra en informit.com/articles.
- Accede a miles de libros y videos en Safari Books
Biblioteca digital en línea en safari.informit.com.
- Obtenga consejos de blogs de expertos en informit.com/blogs.

Visite informit.com/learn para descubrir todas las formas en que puede acceder al contenido tecnológico más actual.

¿Es usted parte de la multitud de TI ?

¡Conéctese con los autores y editores de Pearson a través de canales RSS, Facebook, Twitter, YouTube y más! Visita informit.com/socialconnect.



informIT.com LA FUENTE DE APRENDIZAJE DE TECNOLOGÍA DE CONFIANZA



▲ Addison-Wesley

Cisco Press

EXAM/**CRAM**

IBM
Press.

QUE

PRENTICE HALL

SAMS

Safari
Books Online



Addison
Wesley

REGISTER



ESTE PRODUCTO

informit.com/registro

Registre los productos Addison-Wesley, Exam Cram, Prentice Hall, Que y Sams que posee para desbloquear grandes beneficios.

Para comenzar el proceso de registro, simplemente ingrese a informit.com/register para iniciar sesión o crear una cuenta.

Luego se le pedirá que ingrese el ISBN de 10 o 13 dígitos que aparece en la contraportada de su producto.

Registrar sus productos puede desbloquear los siguientes beneficios:

- Acceso a contenido complementario, incluidos capítulos adicionales, código fuente o archivos de proyecto.
- Un cupón para usar en tu próxima compra.

Los beneficios de registro varían según el producto. Los beneficios aparecerán en la página de su Cuenta en Productos registrados.

Acerca de InformIT : LA FUENTE DE APRENDIZAJE DE TECNOLOGÍA DE CONFIANZA

INFORMIT ES EL HOGAR DE LAS IMPRESIONES EDITORIALES DE TECNOLOGÍA LÍDER Addison-Wesley Professional, Cisco Press, Exam Cram, IBM Press, Prentice Hall Professional, Que y Sams. Aquí obtendrá acceso a contenido y recursos confiables y de calidad de autores, creadores, innovadores y líderes de la tecnología. Ya sea que esté buscando un libro sobre una nueva tecnología, un artículo útil, boletines informativos oportunos o acceso a la biblioteca digital Safari Books Online, InformIT tiene una solución para usted.

informarIT.com

LA FUENTE DE APRENDIZAJE DE TECNOLOGÍA DE CONFIANZA

Addison-Wesley | Prensa de Cisco | Examen intensivo
Prensa IBM | Que | Salón Prentice | Sams

LIBROS DE SAFARI EN LÍNEA