

The Addison-Wesley Signature Series

AGILE TESTING

A PRACTICAL GUIDE FOR
TESTERS AND AGILE TEAMS

LISA CRISPIN
JANET GREGORY



Forewords by Mike Cohn and Brian Marick

MIKE COHN
Mike Cohn
Signature
Book

Elogios por las pruebas ágiles "A"

medida que los métodos ágiles se han ido generalizando, hemos aprendido mucho sobre cómo encaja la disciplina de pruebas en los proyectos ágiles. Lisa y Janet nos dan una visión sólida de qué hacer y qué evitar en las pruebas ágiles".

—Ron Jeffries, www.XProgramming.com

"¡Una excelente introducción a la metodología ágil y cómo afecta a la comunidad de pruebas de software!"

—Gerard Meszaros, líder de práctica ágil y estratega jefe de pruebas de Solution Frameworks, Inc., una consultoría de desarrollo de software ágil y de coaching ágil

"En los deportes y la música, la gente sabe la importancia de practicar la técnica hasta que se convierta en parte de su forma de hacer las cosas. Este libro trata sobre algunas de las técnicas más fundamentales en el desarrollo de software (cómo incorporar calidad al código), técnicas que deberían convertirse en algo natural para todo equipo de desarrollo. El libro proporciona una cobertura amplia y profunda sobre cómo llevar las pruebas al frente del proceso de desarrollo, junto con una serie de ejemplos de la vida real que dan vida al libro".

—Mary Poppendieck, autora de Desarrollo e implementación de software Lean
Desarrollo de software

"Refrescantemente pragmático. Lleno de sabiduría. Ausente de dogma. Este libro cambia las reglas del juego. Todo profesional del software debería leerlo".

—Tío Bob Martin, Object Mentor, Inc.

"Con Agile Testing, Lisa y Janet han utilizado su sensibilidad holística de las pruebas para describir un cambio cultural para los evaluadores y los equipos dispuestos a elevar la efectividad de sus pruebas. La combinación de experiencias de proyectos de la vida real y técnicas específicas proporciona una excelente manera de aprender y adaptarse a las necesidades del proyecto en continuo cambio".

—Adam Geras, M.Sc. Desarrollador-Probador, Ideaca Knowledge Services

"En los proyectos ágiles, todo el mundo parece preguntarse: 'Pero, ¿qué pasa con las pruebas?' ¿Es responsabilidad exclusiva del equipo de desarrollo, del equipo de pruebas o de un esfuerzo de colaboración entre desarrolladores y evaluadores? O, "¿Cuántas pruebas deberíamos automatizar?" ¡Lisa y Janet han escrito un libro que finalmente responde a este tipo de preguntas y más! Ya sea que sea evaluador, desarrollador o gerente, aprenderá muchos ejemplos e historias excelentes de las experiencias laborales del mundo real que han compartido en este excelente libro".

—Paul Duvall, CTO de Stelligent y coautor de Integración continua: mejora de la calidad del software y reducción del riesgo

"Finalmente un libro para evaluadores de equipos ágiles que reconoce que no existe una sola manera correcta! Agile Testing proporciona una cobertura integral de los problemas que enfrentan los evaluadores cuando pasan a Agile: desde herramientas y métricas hasta roles y procesos. Ilustrado con numerosas historias y ejemplos de muchos colaboradores, ofrece una imagen clara de lo que los probadores Agile exitosos están haciendo hoy en día".

—Bret Pettichord, director técnico de WatirCraft y desarrollador principal de Watir

Esta página se dejó en blanco intencionalmente.

PRUEBAS ÁGILES

Esta página se dejó en blanco intencionalmente.

PRUEBAS ÁGILES

UNA GUÍA PRÁCTICA PARA PROBADORES
Y EQUIPOS ÁGILES

Lisa Crispín
Janet Gregorio

 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.

El editor ofrece excelentes descuentos en este libro cuando se solicita en cantidad para compras al por mayor o ventas especiales, que pueden incluir versiones electrónicas y/o portadas personalizadas y contenido específico para su negocio, objetivos de capacitación, enfoque de marketing e intereses de marca. Para obtener más información, póngase en contacto:

Ventas corporativas y gubernamentales de EE. UU.
(800) 382-3419
corpsales@pearsontechgroup.com

Para ventas fuera de los Estados Unidos, comuníquese con:

Ventas internacionales
internacional@pearson.com

Visítenos en la Web: informat.com/aw

Datos de catalogación en publicación de la Biblioteca del Congreso:

Crispín, Lisa.
Pruebas ágiles: una guía práctica para evaluadores y equipos ágiles / Lisa Crispin, Janet Gregory. — 1^a ed.
pag. cm.
Incluye referencias bibliográficas e índice.
ISBN-13: 978-0-321-53446-0 (pbk.: papel alcalino)
ISBN-10: 0-321-53446-8 (pbk.: papel alcalino) 1. Software de computadora—
Pruebas. 2. Desarrollo ágil de software. I. Gregorio, Janet. II. Título.

QA76.76.T48C75 2009
005.1—dc22

2008042444

Copyright © 2009 Pearson Education, Inc.

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 información sobre permisos escribir a:

Educación Pearson, Inc.
Departamento de Derechos y Contratos
501 Calle Boylston, Suite 900
Boston, MA 02116
Fax (617) 671-3447

ISBN-13: 978-0-321-53446-0
ISBN-10: 0-321-53446-8

Texto impreso en Estados Unidos en papel reciclado en RR Donnelley en Crawfordsville, Indiana.
Primera impresión, diciembre de 2008.

Para mi esposo, Bob Downing: ¡eres la rodilla de la abeja!
—Lisa

A Jack, Dana y Susan, y a todos los escritores de mi familia.
—Janet

Y a todos nuestros burros y dragones favoritos.
—Lisa y Janet

Esta página se dejó en blanco intencionalmente.

CONTENIDO

Prólogo de Mike Cohn	xxiii
Prólogo de Brian Marick	xxvi
Prefacio	xxvii
Expresiones de gratitud	xxxvii
Sobre los autores	xli

Parte I	Introducción	1
Capítulo 1 ¿Qué son las pruebas ágiles?		
Valores ágiles	3	
¿Qué queremos decir con “pruebas ágiles”?	4	
Un poco de contexto para los roles y actividades en un equipo ágil	7	
Equipo del cliente	7	
Equipo de desarrollador	7	
Interacción entre los equipos de cliente y desarrollador	8	
¿En qué se diferencian las pruebas ágiles?	9	
Trabajar en equipos tradicionales	9	
Trabajar en equipos ágiles	10	
Pruebas tradicionales frente a	12	
pruebas ágiles	15	
Resumen del enfoque de todo el equipo	17	
Capítulo 2 Diez principios para los probadores ágiles		
¿Qué es un probador ágil?	19	
La mentalidad de prueba ágil	20	

Aplicar principios y valores ágiles	21
Proporcionar retroalimentación continua	22
Entregar valor al cliente	22
Habilite la comunicación cara a cara	23
Ten coraje	25
Mantenlo simple	26
Practica la mejora continua	27
Responder al cambio	28
Autoorganizarse	29
Centrarse en las personas	30
Disfrutar	31
Valor agregado	31
Resumen	33

Parte II	Desafíos organizacionales	35
-----------------	----------------------------------	-----------

Capítulo 3 Desafíos culturales	37
Cultura organizacional	37
Filosofía de Calidad	38
Marcha sostenible	40
Relaciones del cliente	41
Tamaño de la organización	42
Potencia a tu equipo	44
Barreras para la adopción ágil exitosa por parte de los equipos de prueba y control de calidad	44
Pérdida de identidad	44
Roles adicionales	45
Falta de entrenamiento	45
No comprender los conceptos ágiles	45
Experiencia/actitud pasada	48
Diferencias culturales entre roles	48
Introduciendo el cambio	49
hablar de miedos	49
Dar propiedad al equipo	50
Celebre el éxito	50
Expectativas de gestión	52
Cambios culturales para los gerentes	52
Hablar el idioma del gerente	55
El cambio no es fácil	56
Ser paciente	56
Déjalos sentir dolor	56

Construya su credibilidad	57
Trabaja en tu propio desarrollo profesional	57
Cuidado con la mentalidad policial de calidad	57
Vota con tus pies	57
Resumen	58
Capítulo 4 Logística del equipo	59
Estructura de equipo	59
Equipos de control de calidad	60
independientes Integración de probadores en un	61
proyecto ágil Equipos de	64
proyectos ágiles	
Recursos	66
físicos de logística Proporción	66
probador-desarrollador	67
Contratación de un	69
probador ágil Creación de	69
un equipo Equipo	69
autoorganizado que involucra a otros equipos	70
Todos los miembros del equipo	70
tienen el mismo valor Rendimiento y recompensas ¿Qué puedes hacer?	71
Resumen	71
Capítulo 5 Transición de procesos típicos	73
Buscando procesos ligeros	73
Métrica	74
Medidas ajustadas Por	74
qué necesitamos métricas	75
Qué no hacer con las métricas	77
Comunicación de métricas ROI	77
de las métricas	78
Seguimiento de	79
defectos ¿Por qué deberíamos utilizar un sistema de seguimiento de defectos (DTS)?	80
¿Por qué no deberíamos utilizar una EDE?	82
Las herramientas de seguimiento de	83
defectos mantienen su	85
enfoque Planificación	86
de pruebas Estrategia de pruebas frente a trazabilidad	86
de la planificación de pruebas	88

Procesos y modelos existentes	88
Auditorías	89
Marcos, modelos y estándares	90
Resumen	93

Parte III Los cuadrantes de pruebas ágiles 95

Capítulo 6 El propósito de las pruebas	97
Los cuadrantes de pruebas ágiles	97
Pruebas que apoyan al equipo	98
Pruebas que critican el producto	101
Saber cuándo una historia está terminada	104
Responsabilidad compartida	105
Gestión de la deuda técnica	106
Pruebas en contexto	106
Resumen	108
Capítulo 7 Pruebas orientadas a la tecnología que respaldan al equipo Una base de pruebas ágiles El propósito de las	109
pruebas del cuadrante 1 que respaldan la	109
infraestructura ¿Por qué escribir	110
y ejecutar estas pruebas?	111
Nos permite ir más rápido y hacer más	112
Facilitar el trabajo de los evaluadores	114
Diseñar teniendo en cuenta las pruebas	115
Comentarios oportunos	118
¿Dónde terminan las pruebas tecnológicas?	119
¿Qué pasa si el equipo no hace estas pruebas?	121
¿Qué pueden hacer los evaluadores?	121
¿Qué pueden hacer los gerentes?	122
Es un problema de equipo Kit de	123
herramientas	123
IDE de control de código fuente	123
Herramientas de	124
construcción Herramientas de automatización	126
de construcción Resumen	126
de herramientas de prueba unitaria	127

Capítulo 8 Pruebas orientadas al negocio que apoyan al equipo	129
Impulsando el desarrollo con pruebas orientadas al negocio El dilema de los requisitos	129
Lenguaje común	134
Obtención de requisitos	135
Claridad anticipada	140
Condiciones de satisfacción	142
Efectos dominó	143
Porciones finas, porciones pequeñas ¿Cómo sabemos que hemos terminado?	144
Las pruebas mitigan el riesgo Resumen de capacidad de prueba y automatización	147
149	149
150	150
Capítulo 9 Conjunto de herramientas para pruebas orientadas a las empresas que Apoya al equipo	153
Estrategia de herramienta de prueba orientada al negocio	153
Herramientas para obtener ejemplos y requisitos	155
Listas de verificación	156
Mapas mentales	156
Hojas de cálculo	159
maquetas	160
Diagramas de flujo	160
Herramientas basadas en software	163
Herramientas para automatizar pruebas basadas en ejemplos	164
Herramientas para realizar pruebas por debajo del nivel de GUI y API Herramientas para realizar pruebas a través de la GUI Estrategias para	165
escribir pruebas Construir pruebas	170
de forma incremental Mantenga	177
las pruebas aprobadas Utilice patrones de diseño de pruebas apropiados Pruebas basadas en palabras clave	178
y datos Capacidad de prueba Diseño de código y diseño de pruebas Pruebas automatizadas versus manuales del cuadrante	179
2 Resumen de gestión de pruebas	182
183	183
184	184
185	185
186	186
187	187
188	188
189	189
Introducción al Cuadrante 3	190
Manifestaciones	191

Pruéba de escenario	192
Pruéba exploratoria	195
Pruébas basadas en sesiones	200
Automatización y pruebas exploratorias	201
Un probador exploratorio	201
Pruébas de usabilidad	202
Necesidades del usuario y pruebas de persona	202
Navegación	204
Mira la competencia	204
Detrás de la GUI	204
Pruébas API	205
Servicios web	207
Documentos y documentación de prueba	207
Documentación del usuario	207
Informes	208
Herramientas para ayudar con las pruebas exploratorias	210
Configuración de prueba	211
Generación de datos de prueba	212
Herramientas de monitoreo	212
Simuladores	213
Emuladores	213
Resumen	214
 Capítulo 11 Criticar el producto mediante pruebas tecnológicas	
Introducción al cuadrante 4	217
¿Quién lo hace?	217
¿Cuando lo haces? Pruebas de “ilidad”	220
Seguridad	222
Mantenibilidad	227
Interoperabilidad	228
Compatibilidad	229
Confiabilidad	230
Instalabilidad	231
Resumen de “ilidad”	232
Pruebas de rendimiento, carga, estrés y escalabilidad	233
Escalabilidad	233
Pruebas de rendimiento y carga	234
Herramientas de prueba de rendimiento y carga	234
Línea base	235

Entornos de prueba	237
Gestión de la memoria	237
Resumen	238
Capítulo 12 Resumen de los cuadrantes de prueba	241
Revisión de los cuadrantes de prueba	241
Un ejemplo de prueba del sistema	242
La aplicación	242
El equipo y el proceso	243
Pruebas que impulsan el desarrollo	244
Pruebas unitarias	244
Prueba de aceptación	245
Automatización	245
La estructura de prueba funcional automatizada	245
Servicios web	247
Pruebas integradas	248
Criticar el producto con pruebas orientadas al negocio	248
Prueba exploratoria	248
Prueba de fuentes de datos	249
Las pruebas de un extremo a otro	249
Pruebas de aceptación del usuario	250
Fidabilidad	250
Documentación	251
Documentar el código de prueba	251
Informar los resultados de la prueba	251
Uso de los cuadrantes de pruebas ágiles	252
Resumen	253

Parte IV Automatización 255

Capítulo 13 Por qué queremos automatizar las pruebas y qué Nos frena	257
¿Por qué automatizar?	258
Las pruebas manuales toman demasiado tiempo Los procesos manuales son propensos a	258
errores La automatización libera a las personas para hacer su mejor trabajo Las pruebas de regresión automatizadas proporcionan una red de seguridad Las pruebas automatizadas brindan retroalimentación temprana y frecuente Pruebas y ejemplos de que la codificación de impulsos puede hacer más	259
	261
	262
	262

Las pruebas son una excelente documentación	263
Retorno de la inversión y	264
recuperación de la inversión Barreras para la automatización: cosas que se	264
interponen en	264
el camino La	265
lista de Bret Nuestra lista Actitud de los programadores: "¿Por qué automatizar?"	265
La "joroba del dolor" (la curva de aprendizaje)	266
Código de inversión	267
inicial que siempre está en	269
constante cambio	269
269	269
Código	270
heredado Miedo Viejos hábitos ¿Podemos superar estas barreras?	270
Resumen	271
 Capítulo 14 Una estrategia ágil de automatización de pruebas	273
Un enfoque ágil para la automatización de pruebas	274
Categorías de pruebas de automatización	274
Pirámide de automatización de pruebas	276
¿Qué podemos automatizar?	279
Integración continua, compilaciones e implementaciones	280
Pruebas de unidades y componentes	282
Pruebas de API o servicios web	282
Pruebas detrás de la GUI	282
Pruebas de la GUI	282
Pruebas de	283
carga	283
Comparaciones	284
Tareas repetitivas Creación o	284
configuración de datos ¿Qué no deberíamos automatizar?	285
Pruebas de usabilidad	285
Pruebas exploratorias	286
Pruebas que nunca fallarán	286
Pruebas únicas	286
¿Qué podría ser difícil de automatizar?	287
Desarrollo de una estrategia de automatización: ¿por dónde empezamos?	288
¿Dónde duele más?	289
Enfoque de múltiples capas	290
Piense en el diseño y el mantenimiento de las pruebas	292
y elija las herramientas adecuadas	294

Aplicación de principios ágiles a la automatización de pruebas	298
Manténlo simple	298
Comentarios iterativos	299
Enfoque de equipo completo	300
Tomarse el tiempo para hacerlo bien	301
Aprender haciendo	303
Aplicar prácticas de codificación ágil a las pruebas	303
Suministro de datos para pruebas	304
Herramientas de generación de datos	304
Evite el acceso a la base de datos	306
Cuando el acceso a la base de datos es inevitable o incluso deseable	307
Comprenda sus necesidades	310
Evaluación de herramientas de automatización	311
Identificación de requisitos para su herramienta de automatización	311
Una herramienta a la vez	312
Elegir herramientas	313
Herramientas ágiles	316
Implementación de automatización	316
Gestión de pruebas automatizadas	319
Organizar pruebas	319
Organización de los resultados de las pruebas	322
Empezar	324
Resumen	324

Parte V Una iteración en la vida de un probador

327

Capítulo 15 Actividades del probador en la planificación de lanzamientos o temas	329
El propósito del dimensionamiento de la planificación	330
del	332
lanzamiento Cómo dimensionar	332
las historias El papel del evaluador en el	333
dimensionamiento de las historias Un ejemplo	334
de	338
dimensionamiento de las historias	338
Priorización ¿Por qué priorizamos las historias? Consideraciones	339
de prueba mientras se prioriza ¿Qué está dentro del alcance?	340
Los plazos y los cronogramas se	340
centran en el valor	341

Impacto en todo el sistema	342
Participación de terceros Planificación	342
de pruebas ¿Por	345
dónde empezar?	345
¿Por qué escribir un plan de pruebas?	345
Tipos de entornos	346
de prueba de	346
infraestructura de pruebas	347
Datos de prueba	348
Resultados de la prueba	349
Alternativas al plan de prueba	350
Planes de prueba ligeros	350
Usando una matriz de prueba	350
Hoja de cálculo de prueba	353
una pizarra	353
Lista de pruebas automatizadas	354
Preparándose para la visibilidad	354
Seguimiento de tareas y estado de prueba	354
Comunicación de los resultados de las pruebas	357
Métricas de lanzamiento	358
Resumen	366
 Capítulo 16 Empieza a correr	369
Sea proactivo	369
Beneficios	370
¿Realmente necesita esto?	372
Posibles desventajas de la preparación anticipada	373
Claridad anticipada	373
Los clientes hablan con una sola voz	373
Tamaño de	375
la historia Equipos geográficamente	376
dispersos	378
Ejemplos	380
Estrategias de prueba Priorizar los defectos	381
Recursos	381
Resumen	382
 Capítulo 17 Inicio de la iteración	383
Planificación de iteraciones	383
Aprendiendo los detalles	384
Considerando todos los puntos de vista	385

Escribir tarjetas de tareas	389
Decidir la carga de trabajo	393
Historias comprobables	393
Colaborar con los clientes	396
Pruebas y ejemplos de alto nivel	397
Revisar con los clientes	400
Revisando con programadores	400
Casos de prueba como documentación	402
Resumen	403
 Capítulo 18 Codificación y pruebas	405
Impulsando el desarrollo	406
Comience simple	406
Aregar complejidad	407
Evaluar el riesgo	407
Codificación y pruebas progresan juntas	409
Identificar variaciones	410
poder de tres	411
Centrarse en una historia	411
Pruebas que critican el producto	412
Colaborar con programadores	413
Prueba de pares	413
"Muéstrame"	413
Hablar a los clientes	414
Mostrar clientes	414
Entender el negocio	415
Completar tareas de prueba Cómo	415
solucionar errores ¿Es	416
un defecto o una característica?	417
Tolerancia cero a	418
errores de deuda técnica	418
Todo es cuestión de opciones	419
Decidir qué errores registrar	420
Elija cuándo corregir sus errores	421
Elija los medios que debe utilizar para registrar un error	423
Alternativas y sugerencias para lidiar con errores	424
Comience simple	428
Facilita la comunicación	429
Los probadores facilitan la comunicación	429
Equipos distribuidos	431

Pruebas de regresión	432
Mantenga la construcción "verde"	433
Mantenga la construcción rápida	433
Construyendo una suite de regresión	434
Comprobando el "panorama general"	434
Recursos	434
Métricas de iteración	435
Medir el progreso	435
Métricas de defectos	437
Resumen	440
 Capítulo 19 Concluye la iteración	443
Demostración de iteración	443
Retrospectivas	444
Iniciar, detener y continuar	445
Ideas para mejoras	447
Celebre los éxitos	449
Resumen	451
 Capítulo 20 Entrega exitosa ¿Qué caracteriza a un producto?	453
Planificar tiempo suficiente para probar El final del juego	455
Probar la prueba de lanzamiento candidato en un entorno de prueba	458
Pruebas finales no funcionales	458
Integración con aplicaciones externas Conversión de datos y actualizaciones de bases de datos	459
Instalación Pruebas	461
Comunicación ¿Qué pasa si no está listo?	463
Pruebas de clientes	464
Pruebas UAT Alfa/Beta	466
Ciclos de pruebas posteriores al desarrollo	467
Entregables Lanzamiento del producto Criterios de aceptación de lanzamiento	470
Gestión de lanzamientos Empaqueado	474

Expectativas del cliente	475
Soporte de producción	475
Comprender el impacto en las empresas	475
Resumen	476
Resumen de la Parte VI	479
Capítulo 21 Factores clave de éxito	481
Factor de éxito 1: utilice el enfoque de todo el equipo	482
Factor de éxito 2: adoptar una mentalidad de prueba ágil	482
Factor de éxito 3: automatizar las pruebas de regresión	484
Factor de éxito 4: proporcionar y obtener comentarios	484
Factor de éxito 5: construir una base de prácticas básicas	486
Integración continua	486
Entornos de prueba	487
Gestionar la deuda técnica	487
Trabajando incrementalmente	488
La codificación y las pruebas son parte de un proceso	488
Sinergia entre prácticas	489
Factor de éxito 6: colaborar con los clientes	489
Factor de éxito 7: mire el panorama general	490
Resumen	491
Glosario	493
Bibliografía	501
Índice	509

Esta página se dejó en blanco intencionalmente.

PREFACIO

Por Mike Cohn

“La calidad está incorporada”, me decían los programadores. Como parte de una adquisición propuesta, mi jefe me había pedido que realizara una debida diligencia final sobre el equipo de desarrollo y su producto. Ya habíamos establecido que el producto lanzado recientemente por la compañía estaba obteniendo buenos resultados en el mercado, pero debía asegurarme de que no íbamos a comprar más problemas que beneficios. Entonces dediqué mi tiempo al equipo de desarrollo. Estaba buscando problemas que pudieran surgir por haber apresurado el lanzamiento del producto. Me pregunté: “¿Estaba limpio el código? ¿Había módulos en los que solo podía trabajar un desarrollador? ¿Había cientos o miles de defectos esperando a ser descubiertos? Y cuando pregunté sobre el enfoque del equipo respecto de las pruebas, la respuesta que obtuve fue “La calidad está incorporada”.

Debido a que este coloquialismo bastante inusual podría haber significado casi cualquier cosa, presioné más. Lo que descubrí fue que ésta era la manera abreviada del fundador de la empresa para expresar uno de los famosos catorce puntos del pionero de la calidad W. Edwards Deming: incorporar calidad en el producto en lugar de intentar probarlo más tarde.

La idea de incorporar calidad a sus productos es la base de cómo trabajan los equipos ágiles. Los equipos ágiles trabajan en iteraciones cortas en parte para garantizar que la aplicación se mantenga en un estado de calidad conocido. Los equipos ágiles son altamente multifuncionales, con programadores, evaluadores y otros trabajando codo a codo durante cada iteración para que la calidad pueda incorporarse a los productos a través de técnicas como el desarrollo impulsado por pruebas de aceptación, un fuerte énfasis en las pruebas automatizadas y la integración integral. pensamiento en equipo. Los buenos equipos ágiles aportan calidad al desarrollar sus productos continuamente, integrando nuevos trabajos a los pocos minutos de su finalización. Los equipos ágiles utilizan técnicas como la refactorización y una preferencia por la simplicidad para evitar que se acumule deuda técnica.

Aprender cómo hacer estas cosas es difícil, y especialmente para los evaluadores, cuyo papel ha recibido escasa atención en libros anteriores. Afortunadamente, el libro que ahora tiene en sus manos responde a preguntas que todo evaluador comienza a trabajar en un proyecto ágil, como por ejemplo:

- ¿Cuáles son mis roles y responsabilidades?
- ¿Cómo trabajo más estrechamente con los programadores?
- ¿Cuánto automatizamos y cómo empezamos a automatizar?

La experiencia de Lisa y Janet brilla en cada página del libro.

Sin embargo, este libro no es sólo su historia. En este libro, incorporan docenas de historias de evaluadores ágiles del mundo real. Estas historias forman el corazón del libro y son lo que lo hace tan único. Una cosa es gritar desde la torre de marfil: "Aquí se explica cómo realizar pruebas ágiles". Otra es contar las historias de los equipos que han luchado y luego han salido ágiles y victoriosos de desafíos como las pruebas de usabilidad, el código heredado que resiste la automatización, los evaluadores en transición acostumbrados al desarrollo tradicional de fase, las pruebas que "se mantienen al día" con iteraciones cortas y saber cuándo una función está "terminada".

Lisa y Janet estuvieron allí desde el principio, aprendiendo cómo realizar pruebas ágiles cuando la idea predominante era que los equipos ágiles no necesitaban probadores y que los programadores podían lograr la calidad por sí mismos. A lo largo de los años y a través de artículos, presentaciones en conferencias y el trabajo con sus clientes y equipos, Lisa y Janet nos han ayudado a ver el importante papel que deben desempeñar los evaluadores en proyectos ágiles. En este libro, Lisa y Janet utilizan una pirámide de automatización de pruebas, los cuadrantes de pruebas ágiles de Brian Marick (él mismo, otro evaluador ágil de clase mundial) y otras técnicas para mostrar cuánto faltaba en la mentalidad de que dichas pruebas son necesarias, pero los probadores no lo son.

Si desea aprender cómo incorporar calidad a sus productos o es un aspirante a evaluador ágil que busca comprender su función, no se me ocurren mejores guías que Lisa y Janet.

PREFACIO

Por Brian Marick

Imagínese hojeando un paisaje hace miles de años, mirando a las personas de abajo. A duras penas se ganan la vida en un territorio hostil, cazando, pescando y plantando un poco. A lo lejos, ves el brillo de un glaciar. Acercándote, ves que se está derritiendo rápidamente y que apenas está represando un enorme lago. Mientras miras, el lago se abre paso, barriendo el lecho de un río, haciéndolo más profundo, chapoteando contra los acantilados en el otro lado del paisaje, algunos de los cuales colapsan.

Mientras observas, los aturdidos habitantes comienzan a explorar la abertura. Sobre el Al otro lado, hay un paisaje exuberante, combinado con animales más grandes que los que jamás hayan visto. jamás visto antes, algunos pastando en hierba con enormes cabezas de semillas, otros peleándose sobre montículos de fruta caída.

La gente se muda. Casi de inmediato, comienzan a vivir mejor. Pero como el Los años pasan volando, los ves adaptarse. Comienzan a utilizar redes para pescar en los rápidos arroyos. Aprenden el trabajo en equipo necesario para derribar a los más grandes. animales, aunque no sin algunas muertes en el camino. Encuentran formas cada vez mejores de cultivar esta nueva hierba que han llegado a llamar "trigo".

Mientras observa, el loco estallido de innovación da paso a una solución estable, una una buena manera de vivir en esta nueva tierra, una manera que se enseña a cada nueva generación. Aunque justo por allí, espías a alguien inventando la rueda. . .

En los primeros años de este siglo, la adopción de métodos ágiles a veces Parecía como si se rompiera una enorme represa, abriendo el camino a una forma mejor (más productiva y más alegre) de desarrollar software. Muchos de los primeros usuarios vieron beneficios de inmediato, aunque apenas sabían lo que estaban haciendo.

A algunos les resultó más fácil que a otros. Los programadores eran como los cazadores de la fábula anterior. Sí, tenían que aprender nuevas habilidades para poder cazar bisontes, pero sabían cazar conejos, más o menos, y había muchos conejos por ahí. Los evaluadores se parecían más a pescadores submarinos en una tierra donde la pesca submarina no funcionaba. Pasar de la pesca submarina a la pesca con red es un salto conceptual mucho mayor que pasar del conejo al bisonte. Y, si bien algunas de las habilidades (limpiar pescado, por ejemplo) eran las mismas en la nueva tierra, los evaluadores tuvieron que inventar nuevas habilidades para tejer redes antes de poder realmente hacer todo lo posible.

Así que las pruebas se quedaron atrás. Afortunadamente, tuvimos a los primeros en adoptarlos, como Lisa y Janet, personas que se lanzaron al lado de los programadores, evaluadores que no estaban celosos de su rol ni de su independencia, personas francamente agradables que pudieron descubrir el mayor cambio de todos en las pruebas ágiles: la capacidad del evaluador. nuevo rol social.

Como resultado, tenemos este libro. Es la solución estable, la buena manera para que los evaluadores vivan en esta nueva tierra ágil nuestra. No es la última palabra (nos vendría bien la rueda, y yo mismo estoy ansioso por que alguien invente los antibióticos), pero lo que se enseña aquí será de gran utilidad hasta que alguien, tal vez Lisa y Janet, produzcan el próximo gran cambio.

PREFACIO

Fuimos los primeros en adoptar la Programación Extrema (XP), realizando pruebas en equipos XP que no estaban del todo seguros de dónde encajaban los evaluadores o su tipo de pruebas. En ese momento, no había mucho en lo ágil (que no se llamaba ágil todavía) literatura sobre las pruebas de aceptación, o cómo los evaluadores profesionales podrían contribuir. Aprendimos no sólo de nuestras propias experiencias sino también de otros miembros de la pequeña comunidad ágil. En 2002, Lisa coescribió Testing Extreme Programming con Tip House, con mucha ayuda de Janet. Desde entonces, el desarrollo ágil ha evolucionado y la comunidad de pruebas ágiles ha florecido. Con tanta gente aportando ideas, hemos aprendido mucho más sobre las pruebas ágiles.

Individualmente y juntos, ayudamos a los equipos a hacer la transición a la metodología ágil, ayudamos a los evaluadores a aprender cómo contribuir en equipos ágiles y trabajamos con otros miembros de la comunidad ágil para explorar formas en que los equipos ágiles pueden tener más éxito en las pruebas. Nuestras experiencias difieren. Lisa ha pasado la mayor parte de su tiempo como probadora ágil en equipos estables trabajando durante años en aplicaciones web en las industrias minorista, telefónica y financiera. Janet ha trabajado con organizaciones de software desarrollando sistemas empresariales en una variedad de industrias. Estos proyectos ágiles han incluido el desarrollo de un sistema de manejo de mensajes, un sistema de seguimiento ambiental, un sistema de gestión remota de datos (incluida una aplicación integrada, con una red de comunicación además de la aplicación), una aplicación de contabilidad de producción de petróleo y gas, y aplicaciones en la industria del transporte aéreo. Ha desempeñado diferentes roles (a veces como evaluadora, a veces como entrenadora), pero siempre ha trabajado para integrar mejor a los evaluadores con el resto del equipo. Ha estado en equipos desde tan solo seis meses hasta un año y medio.

Con estos diferentes puntos de vista, hemos aprendido a trabajar juntos y complementar las habilidades de cada uno, y hemos realizado muchas presentaciones y tutoriales juntos.

POR QUÉ ESCRIBIMOS ESTE LIBRO

Varios libros excelentes orientados al desarrollo ágil sobre pruebas
Se han publicado patrones de prueba (consulte nuestra bibliografía). Estos libros son
generalmente enfocado en ayudar al desarrollador. Decidimos escribir un libro.
destinado a ayudar a los equipos ágiles a tener más éxito en la entrega de valor empresarial
utilizando pruebas que la empresa pueda comprender. Queremos ayudar a los probadores y
Profesionales de aseguramiento de la calidad (QA) que han trabajado en sectores más tradicionales.
Las metodologías de desarrollo hacen la transición al desarrollo ágil.

Hemos descubierto cómo aplicar, en un nivel práctico y cotidiano, los frutos
de nuestra propia experiencia trabajando con equipos de todos los tamaños y una variedad de ideas
de otros practicantes ágiles. Hemos reunido todo esto en este libro para
ayudar a los evaluadores, gerentes de control de calidad, desarrolladores, gerentes de desarrollo, propietarios
de productos y cualquier otra persona interesada en realizar pruebas efectivas en sistemas ágiles.
proyectos para entregar el software que sus clientes necesitan. Sin embargo, nos hemos centrado en el
papel del evaluador, un papel que puede ser adoptado por una variedad de
profesionales.

Las prácticas de pruebas ágiles no se limitan a los miembros de equipos ágiles. Ellos pueden ser
También se utiliza para mejorar las pruebas en proyectos que utilizan metodologías de desarrollo tradicionales.
Este libro también está destinado a ayudar a los evaluadores que trabajan en proyectos.
utilizando cualquier tipo de metodología de desarrollo.

El desarrollo ágil no es la única forma de entregar software con éxito. Sin embargo, todos los equipos exitosos
en los que hemos estado, ágiles o en cascada, han tenido
varios puntos en común críticos. Los programadores escriben y automatizan la unidad.
y pruebas de integración que proporcionen una buena cobertura de código. Son disciplinados
en el uso del control del código fuente y la integración del código. Los evaluadores capacitados participan
desde el inicio del ciclo de desarrollo y reciben tiempo y recursos para realizar un trabajo adecuado en todas
las formas necesarias de prueba. Un automatizado
Se ejecuta el paquete de regresión que cubre la funcionalidad del sistema en un nivel superior.
y revisado periódicamente. El equipo de desarrollo comprende las necesidades de los clientes.
puestos de trabajo y sus necesidades, y trabaja en estrecha colaboración con los expertos en negocios.

Las personas, no las metodologías ni las herramientas, hacen que los proyectos tengan éxito. Disfrutamos ágil
desarrollo porque sus valores, principios y prácticas fundamentales permiten a las personas
para hacer su mejor trabajo, y las pruebas y la calidad son fundamentales para un desarrollo ágil. En este
libro, explicamos cómo aplicar valores y principios ágiles a
su situación de prueba única y permita que sus equipos tengan éxito. Tenemos
Más información sobre esto en el Capítulo 1, “¿Qué son las pruebas ágiles, de todos modos?” y en
Capítulo 2, “Diez principios para los evaluadores ágiles”.

CÓMO ESCRIBIMOS ESTE LIBRO

Habiendo experimentado los beneficios del desarrollo ágil, utilizamos prácticas ágiles. para producir este libro. Cuando comenzamos a trabajar en el libro, hablamos con Agile probadores y equipos de todo el mundo para descubrir qué problemas encontraron y cómo los abordaron. Planeamos cómo cubriríamos estas áreas en el libro.

Hicimos un plan de lanzamiento basado en iteraciones de dos semanas. Cada dos semanas, nosotros Entregué dos capítulos borradores en el sitio web de nuestro libro. porque no lo somos Ubicados en el mismo lugar, encontramos herramientas para comunicarnos, proporcionar "control de código fuente" para nuestros capítulos, entregar el producto a nuestros clientes y obtener sus comentario. No pudimos "emparejar" mucho en tiempo real, pero intercambiamos capítulos y adelante para su revisión y revisión, y tenía "stand-ups" informales diariamente a través de mensajes instantáneos.

Nuestros "clientes" fueron las personas generosas de la comunidad ágil que se ofrecieron como voluntarios para revisar los borradores de los capítulos. Proporcionaron comentarios por correo electrónico o (si fuéramos afortunado) en persona. Usamos los comentarios para guiarnos mientras continuamos escribiendo. y revisando. Después de terminar todos los borradores, elaboramos un nuevo plan para completar las revisiones, incorporando todas las ideas útiles de nuestros "clientes".

Nuestra herramienta más importante fueron los mapas mentales. Empezamos creando una mente mapa de cómo imaginamos todo el libro. Luego creamos mapas mentales para cada sección del libro. Antes de escribir cada capítulo, hicimos una lluvia de ideas con un mapa mental. Mientras revisábamos, revisamos los mapas mentales, lo que nos ayudó Piense en ideas que quizás nos hayamos perdido.

Como creemos que los mapas mentales agregaron mucho valor, hemos incluido el mapa mental como parte de la apertura de cada capítulo. Esperamos que te ayuden obtenga una descripción general de toda la información incluida en el capítulo e inspire Intente utilizar mapas mentales usted mismo.

NUESTRA AUDIENCIA

Este libro le ayudará si alguna vez ha preguntado alguna de las siguientes excelentes preguntas que hemos escuchado muchas veces:

Si los desarrolladores escriben pruebas, ¿qué hacen los evaluadores?
Soy gerente de control de calidad y nuestra empresa está implementando un desarrollo ágil (Scrum, XP, DSDM, nombre su versión). ¿Cuál es mi papel ahora?

He trabajado como probador en un equipo tradicional en cascada y estoy muy entusiasmado con lo que he leído sobre ágil. ¿Qué necesito saber para trabajar en un equipo ágil?

¿Qué es un "probador ágil"?

Soy desarrollador en un equipo ágil. Primero estamos escribiendo código de prueba, pero nuestros clientes aún no están satisfechos con lo que ofrecemos. ¿Qué nos falta?

Soy desarrollador en un equipo ágil. Primero estamos escribiendo nuestro código de prueba. Nos aseguramos de tener pruebas para todo nuestro código. ¿Por qué necesitamos probadores?

Entreno a un equipo de desarrollo ágil. Nuestro equipo de control de calidad no puede seguirnos el ritmo y las pruebas siempre van por detrás. ¿Deberíamos simplemente planear probar una iteración detrás del desarrollo?

Soy gerente de desarrollo de software. Recientemente hicimos la transición a ágil, pero todos nuestros evaluadores abandonaron. ¿Por qué?

Soy un probador en un equipo que se está volviendo ágil. No tengo habilidades de programación o automatización. ¿Hay algún lugar para mí en un equipo ágil?

¿Cómo pueden las pruebas mantenerse al día con iteraciones de dos semanas?

¿Qué pasa con las pruebas de carga, las pruebas de rendimiento, las pruebas de usabilidad y todas las demás "ilidades"? ¿Dónde encajan estos?

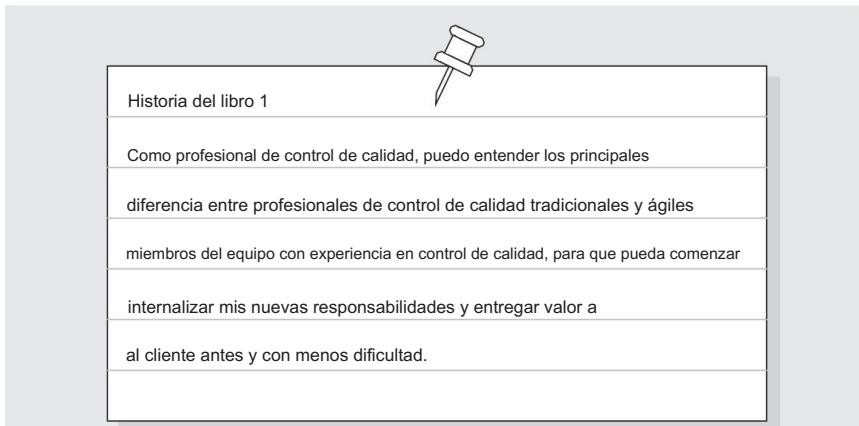
Tenemos requisitos de auditoría. ¿Cómo abordan estos problemas el desarrollo y las pruebas ágiles?

Si tiene preguntas similares y busca consejos prácticos sobre cómo los evaluadores contribuyen a los equipos ágiles y cómo los equipos ágiles pueden hacer un trabajo efectivo trabajo de pruebas, has elegido el libro correcto.

Hay muchos "sabores" de desarrollo ágil, pero todos tienen mucho en común. Apoyamos el Manifiesto Ágil, que explicamos en el Capítulo 1, "¿Qué son las pruebas ágiles, de todos modos?" Ya sea que estés practicando Scrum, Extreme Programación, Crystal, DSDM o su propia variación de desarrollo ágil, Aquí encontrará información que le ayudará con sus esfuerzos de prueba.

Una historia de usuario para un libro de pruebas ágiles

Cuando Robin Dymond, un consultor administrativo y formador que ha ayudado a muchos equipos a adoptar lean y agile, se enteró de que estábamos escribiendo este libro, nos envió la historia de usuario que le gustaría haber cumplido. Encapsula muchos de los requisitos que planeamos cumplir.



Condiciones de aceptación:

- Mis preocupaciones y temores acerca de perder el control de las pruebas son dirigido.
- Se abordan mis inquietudes y temores sobre tener que escribir código (nunca lo he hecho).
- Como evaluador, entiendo mi nuevo valor para el equipo.
- Como evaluador nuevo en Agile, puedo leer fácilmente sobre las cosas más importante para mi nuevo rol.
- Como evaluador nuevo en Agile, puedo ignorar fácilmente cosas que son menos importantes. importante para mi nuevo rol.
- Como evaluador nuevo en Agile, puedo obtener fácilmente más detalles sobre Agile. pruebas que son importantes para MI contexto.

Si tuviera que sugerir una solución a este problema, pensaría en Scrum versus XP. Con Scrum obtienes una vista simple que permite a las personas adoptar Agile rápidamente. Sin embargo, Scrum es la punta del iceberg para los equipos ágiles exitosos. Para los evaluadores nuevos, me encantaría ver ideas de pruebas ágiles expresadas en capas de detalle. ¿Qué necesito saber hoy, qué debo saber mañana y qué aspectos contextuales debo considerar para la mejora continua?

Hemos intentado proporcionar estos niveles de detalle en este libro. Abordaremos las pruebas ágiles desde diferentes perspectivas: transición al desarrollo ágil, utilizando una matriz de pruebas ágiles para guiar los esfuerzos de prueba y explicando todas las diferentes actividades de prueba que tienen lugar a lo largo del ciclo de desarrollo ágil.

COMO USAR ESTE LIBRO

Si no está seguro de por dónde empezar en este libro, o simplemente desea una descripción general rápida, le sugerimos que lea el último capítulo, el Capítulo 22, "Factores clave del éxito". y sigue a donde te lleve.

Parte I: Introducción

Si desea respuestas rápidas a preguntas como "¿Son las pruebas ágiles diferentes a las ¿Pruebas en proyectos en cascada? o "¿Cuál es la diferencia entre un probador en un Equipo tradicional y un probador ágil?", comience con la Parte I, que incluye los siguientes capítulos:

Capítulo 1: ¿Qué son las pruebas ágiles?

Capítulo 2: Diez principios para los probadores ágiles

Estos capítulos son la "punta del iceberg" que Robin pidió en su usuario historia. Incluyen una descripción general de en qué se diferencia la metodología ágil de la tradicional por fases. acercarse y explorar el enfoque de "todo el equipo" para la calidad y las pruebas.

En esta parte del libro definimos la "mentalidad de prueba ágil" y lo que hace probadores exitosos en equipos ágiles. Explicamos cómo los testers aplican valores ágiles y principios para aportar su experiencia particular.

Parte II: Desafíos organizacionales

Si es evaluador o gerente de un equipo de control de calidad tradicional, o está entrenando a un equipo que está pasando a ser ágil, la Parte II lo ayudará con los desafíos organizacionales que enfrentan los equipos en transición. La actitud de "todo el equipo" representa mucho de cambios culturales a los miembros del equipo, pero ayuda a superar los probadores de miedo tienen cuando se preguntan cuánto control tendrán o si serán

Se espera que escriba código.

Algunas de las preguntas respondidas en la Parte II son:

¿Cómo podemos involucrar al equipo de control de calidad?

¿Qué pasa con las expectativas de la gerencia?

¿Cómo deberíamos estructurar nuestro equipo ágil y dónde encajan los evaluadores?

¿Qué buscamos al contratar un tester ágil?

¿Cómo nos enfrentamos a un equipo distribuido por todo el mundo?

La Parte II también presenta algunos temas de los que no siempre nos gusta hablar. Nosotros explorar ideas sobre cómo hacer la transición de procesos y modelos, como auditorías o Cumplimiento SOX, que son comunes en entornos tradicionales.

Las métricas y cómo se aplican pueden ser un tema controvertido, pero hay formas positivas de utilizarlos en beneficio del equipo. El seguimiento de defectos se vuelve fácilmente un punto de discordia para los equipos, con preguntas como "¿Utilizamos un sistema de seguimiento de defectos?" o "¿Cuándo registramos errores?"

Dos preguntas comunes sobre las pruebas ágiles de personas con pruebas tradicionales experiencia del equipo son "¿Qué pasa con los planes de prueba?" y "¿Es cierto que no existe documentación sobre proyectos ágiles?" La segunda parte aclara estos misterios.

Los capítulos de la Parte II son los siguientes:

Capítulo 3: Desafíos culturales

Capítulo 4: Logística del equipo

Capítulo 5: Transición de procesos típicos

Parte III: Los cuadrantes de pruebas ágiles

¿Quieres más detalles sobre qué tipos de pruebas se realizan en proyectos ágiles?

¿Se pregunta quién hace qué pruebas? ¿Cómo sabes si

¿Has hecho todas las pruebas necesarias? ¿Cómo decides qué prácticas,

¿Las técnicas y herramientas se ajustan a su situación particular? Si estas son tus preocupaciones, consulte la Parte III.

Usamos los cuadrantes de pruebas ágiles de Brian Marick para explicar el propósito de pruebas. Los cuadrantes le ayudan a definir las diferentes áreas de sus pruebas. debe abordar, desde las pruebas a nivel unitario hasta la confiabilidad y otras "habilidades", y todo lo demás. Aquí es donde profundizamos en el meollo de la cuestión de cómo para entregar un producto de alta calidad. Te explicamos técnicas que pueden ayudarte a comunicarse bien con sus clientes y comprenda mejor sus necesidades. Esta parte del libro muestra cómo las pruebas impulsan el desarrollo en múltiples niveles. También proporciona herramientas para su kit de herramientas que pueden ayudarle a definir, diseñar y ejecutar pruebas que apoyen al equipo y critiquen el producto. Los capítulos incluyen lo siguiente:

Capítulo 6: El propósito de las pruebas

Capítulo 7: Pruebas tecnológicas que respaldan al equipo

- Capítulo 8: Pruebas empresariales que respaldan al equipo
- Capítulo 9: Conjunto de herramientas para pruebas empresariales que respalden al equipo
- Capítulo 10: Pruebas empresariales que critican el producto
- Capítulo 11: Criticar el producto mediante pruebas tecnológicas
- Capítulo 12: Resumen de los cuadrantes de prueba

Parte IV: Automatización

La automatización de pruebas es un foco central de los equipos ágiles exitosos, y da miedo

Tema para mucha gente (lo sabemos, ¡porque nos ha asustado antes!).

¿Cómo se puede comprimir la automatización de pruebas en iteraciones cortas y seguir obteniendo todas las ventajas? historias completadas?

La Parte IV entra en detalles sobre cuándo y por qué automatizar, cómo superar barreras para la automatización de pruebas y cómo desarrollar e implementar una estrategia de automatización de pruebas que funcione para su equipo. Porque las herramientas de automatización de pruebas cambian y evolucionan tan rápidamente, nuestro objetivo no es explicar cómo utilizar herramientas, sino para ayudarle a seleccionar y utilizar las herramientas adecuadas para su situación. Nuestro Los consejos ágiles de automatización de pruebas lo ayudarán con desafíos difíciles, como probar código heredado.

Los capítulos son los siguientes:

Capítulo 13: Por qué queremos automatizar las pruebas y qué nos frena

Capítulo 14: Una estrategia ágil de automatización de pruebas

Parte V: Una iteración en la vida de un probador

Si simplemente desea tener una idea de lo que hacen los evaluadores a lo largo del ciclo de desarrollo ágil, o necesita ayuda para reunir toda la información de este libro, vaya a la Parte V. Aquí relatamos una iteración, y más, en la vida de un ágil ensayador. Los evaluadores aportan un valor enorme a lo largo de los ciclos ágiles de desarrollo de software. En la Parte V, explicamos las actividades que los evaluadores realizan a diario. Comenzamos planificando lanzamientos e iteraciones para que cada iteración llegue a un buen comienzo y avance a través de la iteración: colaborar con el cliente y equipos de desarrollo, pruebas y escritura de código. Finalizamos la iteración ofreciendo nuevas funciones y encontrando formas para que el equipo mejore el proceso.

Los capítulos se dividen de esta manera:

Capítulo 15: Actividades del probador en la planificación del lanzamiento o del tema

Capítulo 16: Empiece a trabajar

- Capítulo 17: Inicio de la iteración
- Capítulo 18: Codificación y pruebas
- Capítulo 19: Concluye la iteración
- Capítulo 20: Entrega exitosa

Parte VI: Resumen En el

Capítulo 21, "Factores clave de éxito", presentamos siete factores clave que los equipos ágiles pueden utilizar para realizar pruebas exitosas. Si tiene problemas para decidir por dónde empezar con las pruebas ágiles o cómo trabajar para mejorar lo que está haciendo ahora, estos factores de éxito le darán alguna dirección.

Otros elementos

También hemos incluido un glosario que esperamos que le resulte útil, así como referencias a libros, artículos, sitios web y blogs en la bibliografía.

¡COMIENCE A HACERLO HOY!

El desarrollo ágil se trata de hacer tu mejor trabajo. Cada equipo tiene desafíos únicos. Hemos intentado presentar toda la información que creemos que puede ayudar a los evaluadores ágiles, sus equipos, gerentes y clientes. Aplica las técnicas que creas apropiadas para tu situación. Experimente constantemente, evalúe los resultados y vuelva a leer este libro para ver qué podría ayudarle a mejorar.

Nuestro objetivo es ayudar a los evaluadores y a los equipos ágiles a disfrutar ofreciendo el mejor y más valioso producto posible.

Cuando le preguntamos a Dierk König, fundador y director de proyectos de Canoo Web-Test, cuál pensaba que era el factor de éxito número uno para las pruebas ágiles, respondió: "¡Empiece a hacerlo hoy mismo!". Puedes dar un pequeño paso para mejorar las pruebas de tu equipo ahora mismo. ¡Empieza!

Esta página se dejó en blanco intencionalmente.

EXPRESIONES DE GRATITUD

Tanta gente nos ha ayudado con este libro que es difícil saber a quién gracias primero. Chris Guzikowski nos dio la oportunidad de escribir este libro y siguió animándonos a lo largo del camino. Cuando estábamos decidiendo si tomar En una tarea tan gigantesca, Mike Cohn nos dio el sabio consejo de que los mejores La razón para escribir un libro es que tienes algo que decir. Seguro que tenemos mucho que decir sobre las pruebas ágiles. Afortunadamente, también lo están muchas otras personas que estuvieron dispuestas a echarnos una mano.

Muchas gracias a Brian Marick y Mike Cohn por escribir tan amables prólogos. Nos sentimos honrados de que Mike haya seleccionado nuestro libro para su serie exclusiva. Estamos agradecidos por las muchas ideas y observaciones tuyas que se incluyen en este libro.

La “Matriz de pruebas ágiles” de Brian Marick nos ha guiado a ambos en nuestros proyectos ágiles. durante varios años y constituye el núcleo de la Parte III. Gracias Brian por pensar en los cuadrantes (y tantas otras contribuciones a las pruebas ágiles) y dejarnos usarlos aquí.

Hicimos uso constante del valor ágil de la retroalimentación. Muchas gracias a nuestros revisores oficiales: Jennitta Andrea, Gerard Meszaros, Ron Jeffries y Paul Du-vall. Cada uno tuvo comentarios únicos y reveladores que nos ayudaron mucho. mejorar el libro. Gerard también nos ayudó a ser más consistentes y correctos en nuestra terminología de pruebas y contribuimos con algunas historias de éxito de pruebas ágiles.

Un agradecimiento especial a dos revisores y evaluadores ágiles de primer nivel que leyeron cada Word escribimos y pasamos horas discutiendo los borradores de los capítulos con nosotros en persona: Pierre Veragen y Paul Rogers. Muchas de las buenas ideas contenidas en este libro son suyo.

xxxviii AGRADECIMIENTOS

Entrevistamos a varios equipos para saber qué consejos darían a las nuevas empresas ágiles. equipos y evaluadores, y solicitó historias de éxito y "lecciones aprendidas" de colegas de la comunidad de pruebas ágiles. Nuestro más sincero agradecimiento a nuestros numerosos colaboradores con comentarios laterales y citas, así como a quienes nos brindaron comentarios útiles, incluidos (sin ningún orden en particular) Robin Dymond, Bret Pettichord, Tae Chang, Bob Galen, Erika Boyer, Grig Gheorghiu, Erik Bos, Mark Benander, Jonathan Rasmusson, Andy Pols, Dierk König, Rafael Santos, Jason Holzer, Christophe Louvion, David Reed, John Voris, Chris McMahon, Declan Whelan, Michael Bolton, Elisabeth Hendrickson, Joe Yakich, Andrew Glover, Alessandro Col-lino, Coni Tartaglia, Markus Gärtner, Megan Sumrell, Nathan Silberman, Mike Thomas, Mike Busse, Steve Perkins, Joseph King, Jakub Oleszkiewicz, Pierre Veragen (nuevamente), Paul Rogers (nuevamente), Jon Hagar, Antony Marcano, Patrick Wilson-Welsh, Patrick Fleisch, Apurva Chandra, Ken De Souza y Carol Vaage.

Muchas gracias también al resto de nuestra comunidad de revisores no oficiales que leyó capítulos, brindó comentarios e ideas, y permitan intercambiar ideas a partir de ellos, incluidos Tom Poppendieck, Jun Bueno, Kevin Lawrence, Hannu Kokko, Titus Brown, Wim van de Goor, Lucas Campos, Kay Johansen, Adrian Howard, Henrik Kniberg, Shelly Park, Robert Small, Senaka Suriyaachchi y Erik Petersen. Y si no lo hemos incluido aquí, no es que valoremos su contribución menor, ¡es solo que no tomamos notas lo suficientemente buenas! Nosotros Espero que veas cómo tu tiempo y esfuerzo dieron sus frutos en el libro terminado.

Apreciamos el trabajo preliminar sentado por los pioneros ágiles que nos han ayudado y nuestros equipos triunfan con agilidad. Encontrarás algunos de sus trabajos en la bibliografía. Estamos agradecidos por los equipos ágiles que nos han brindado tantas oportunidades herramientas de prueba de origen que ayudan a todos nuestros equipos a ofrecer tanto valor. Algunos de esas herramientas también se enumeran en la bibliografía.

Gracias a Mike Thomas por tomar muchas de las fotografías de acción de un ágil equipo que aparece en este libro. Esperamos que estas fotos muestren a aquellos de ustedes nuevos a pruebas y desarrollo ágiles que no hay gran misterio: simplemente es bueno personas que se reúnen para discutir, hacer demostraciones y hacer dibujos.

Muchas gracias a nuestro equipo editorial y de producción de Addison-Wesley que Respondió pacientemente muchas preguntas y convirtió esto en el libro de aspecto profesional que ve aquí, incluidos Raina Chrobak, Chris Zahn, John Fuller, Sally Gregg, Bonnie Granat, Diane Freed, Jack Lewis y Kim Arney.

La historia de Lisa

Estoy eternamente agradecido a Janet por aceptar escribir este libro conmigo. Nos mantuvieron organizados y encaminados para que pudiéramos compatibilizar la escritura de libros con nuestros trabajos de tiempo completo y nuestra vida personal. Tengo la suerte de tener un compañero de escritura cuya experiencia es complementaria a la mía. Como cualquier proyecto ágil exitoso, este es un verdadero esfuerzo de equipo. Ha sido un trabajo duro, pero gracias a Janet también ha sido muy divertido.

Me gustaría agradecer a los miembros de mi equipo actual en ePlan Services Inc. (anteriormente conocido como Fast401k), al que me uní (gracias a Mike Cohn, nuestro primer líder) en 2003. Todos aprendimos tanto trabajando para Mike que primer año, y es un testimonio de su liderazgo que continuamos mejorando y ayudando al negocio a crecer.

Gracias a mis increíbles compañeros de equipo que me ayudaron a convertirme en un mejor evaluador y un miembro más ágil del equipo, y todos fueron buenos deportistas mientras Mike Thomas nos tomaba fotos en acción: Nanda Lankapalli, Tony Sweets, Jeff Untis, Lisa Owens, Mike Thomas, Vince Palumbo, Mike Busse, Nehru Kaja, Trevor Sterritt, Steve Kives y los ex pero aún queridos miembros del equipo Joe Yakich, Jason Kay, Jennifer Riefenberg, Matt Tierney y Charles LeRose. También he tenido la suerte de trabajar con el mejor equipo de atención al cliente del mundo. Son demasiados para mencionarlos aquí, pero muchas gracias a ellos, y en particular a Steve Perkins, Anne Olguin y Zachary Shannon, quienes nos ayudan a concentrarnos en generar valor. Gracias también a Mark y Dan Gutrich, fundadores y líderes de ePlan Services, por brindarnos la oportunidad de tener éxito con el desarrollo ágil.

Gracias a Kay y Zhon Johansen por enseñarme sobre mapas mentales en Agile 2006. Espero que hayamos aprovechado esta habilidad al crear este libro.

Mucha gratitud a todos mis amigos y familiares, a quienes descuidé terriblemente durante los muchos meses que pasé escribiendo este libro y que, sin embargo, me apoyaron constantemente. Hay demasiados para mencionarlos, pero debo agradecer especialmente a Anna Blake por su constante comprensión y provisión de terapia con burros. Chester y Ernest, los burros de mi corazón, han seguido tirando de mí. Dodger no hizo todo el viaje de escribir libros en este mundo, pero su memoria continúa animándome.

Mi pequeño caniche y musa Tango estuvo a mi lado cada minuto que trabajé en este libro en casa, acompañado ocasionalmente por Bruno, Bubba, Olive, Squiggy, Starsky, Bobcat y Patty. Gracias a mis padres por estar orgullosos de mí y no quejarse de mi negligencia hacia ellos durante este tiempo de escritura de libros.

Sé que mi esposo, Bob Downing, respiró hondo cuando exclamó: "Tengo la oportunidad de escribir otro libro sobre pruebas ágiles", pero aun así me animó constantemente y me hizo posible encontrar tiempo para escribir.

Mantuvo en funcionamiento el "refugio para no matar", mantuvo nuestras vidas en marcha, mantuvo mi ánimo en alto y me sostuvo con muchas comidas fabulosas. Él es la luz de mi vida.

—Lisa

Parte I

INTRODUCCIÓN

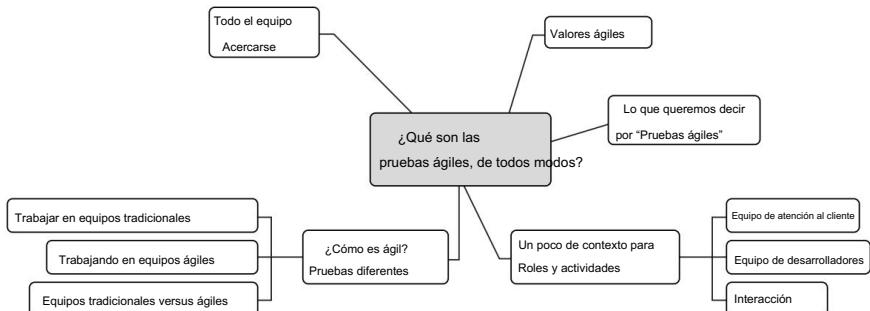
En los dos primeros capítulos, brindamos una descripción general de las pruebas ágiles, destacando en qué se diferencian de las pruebas en un enfoque tradicional en fases o en "cascada". Exploramos el enfoque de "todo el equipo" para la calidad y las pruebas.

Esta página se dejó en blanco intencionalmente.

Capítulo 1

QUE ES ÁGIL

¿ PRUEBAS DE TODOS MODOS?



Como mucha terminología, "desarrollo ágil" y "pruebas ágiles" significan cosas diferentes para diferentes personas. En este capítulo, explicamos nuestra visión de lo ágil, que refleja el Manifiesto Ágil y los principios y valores generales compartidos por los diferentes métodos ágiles. Queremos compartir un lenguaje común con usted, el lector, por lo que repasaremos parte de nuestro vocabulario. Comparamos y contrastamos el desarrollo y las pruebas ágiles con el enfoque por fases más tradicional. El enfoque de "todo el equipo" promovido por el desarrollo ágil es fundamental para nuestra actitud hacia la calidad y las pruebas, por lo que también hablamos de eso aquí.

VALORES ÁGILES

"Ágil" es una palabra de moda que probablemente dejará de usarse algún día y hará Este libro parece obsoleto. Está cargado de diferentes significados que se aplican en diferentes circunstancias. Una forma de definir el "desarrollo ágil" es observar la Manifiesto Ágil (ver Figura 1-1).

Utilizando los valores del Manifiesto como guía, nos esforzamos por ofrecer pequeños grandes cantidades de valor empresarial en ciclos de lanzamiento extremadamente cortos.

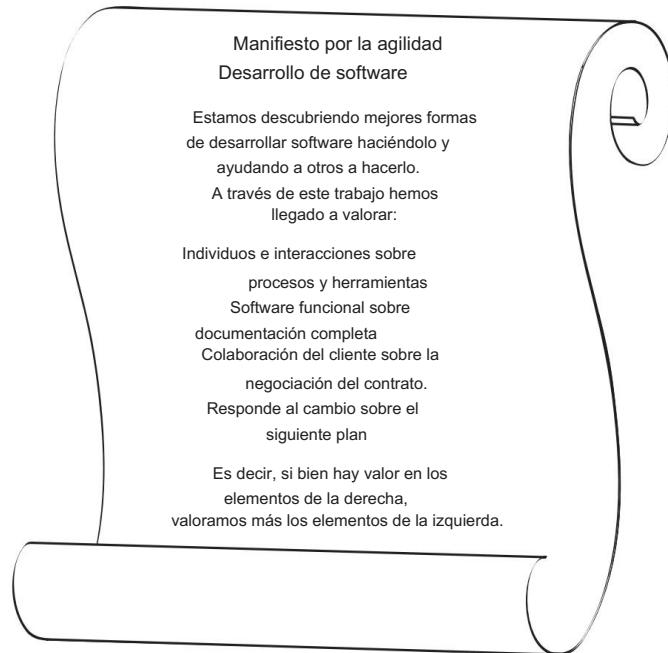


Figura 1-1 Manifiesto Ágil

Capítulo 21, "Clave Factores de éxito," enumera los factores clave de éxito para las pruebas ágiles.

Usamos la palabra "ágil" en este libro en un sentido amplio. Si tu equipo es practicar un método ágil particular, como Scrum, XP, Crystal, DSDM o FDD, por nombrar algunos, o simplemente adoptar cualquier principio y práctica que haga Si tiene sentido para su situación, debería poder aplicar las ideas de este libro. Si está entregando valor al negocio de manera oportuna con alta calidad software y su equipo se esfuerza continuamente por mejorar, aquí encontrará información útil. Al mismo tiempo, hay prácticas ágiles particulares que consideramos son cruciales para el éxito de cualquier equipo. Hablaremos de estos a lo largo del libro.

¿ QUÉ ENTENDEMOS POR “ PRUEBAS ÁGILES”?

Es posible que hayas notado que usamos el término "probador" para describir a una persona, cuyas principales actividades giran en torno a las pruebas y el aseguramiento de la calidad. usted También vemos que a menudo usamos la palabra "programador" para describir a una persona cuya Las actividades principales giran en torno a la escritura de código de producción. No pretendemos eso Estos términos suenan limitados o insignificantes. Los programadores hacen más que girar una especificación en un programa. No los llamamos "desarrolladores", porque todos

Todos los involucrados en la entrega de software son desarrolladores. Los probadores hacen más que realizar "tareas de prueba". Cada miembro del equipo ágil se centra en ofrecer un Producto de alta calidad que proporciona valor comercial. Los evaluadores ágiles trabajan para garantizar que su equipo brinde la calidad que sus clientes necesitan. Usamos el términos "programador" y "probador" por conveniencia.

Varias prácticas básicas utilizadas por los equipos ágiles se relacionan con las pruebas. Los programadores ágiles utilizan el desarrollo basado en pruebas (TDD), también llamado diseño basado en pruebas, para escribir código de producción de calidad. Con TDD, el programador escribe una prueba para un poquito de funcionalidad, lo ve fallar, escribe el código que lo hace pasar y luego pasa a la siguiente pequeña funcionalidad. Los programadores también escriben pruebas de integración de código para asegurarse de que las pequeñas unidades de código funcionen juntas como destinado. Esta práctica esencial ha sido adoptada por muchos equipos, incluso aquellos que no se llaman a sí mismos "ágiles", porque es simplemente una forma inteligente de pensar a través del diseño de su software y prevenir defectos. La Figura 1-2 muestra una muestra Resultado de la prueba unitaria que un programador podría ver.

Este libro no trata sobre pruebas a nivel de unidad o de componente, pero estos tipos de pruebas son fundamentales para un proyecto exitoso. Brian Marick [2003] describe este tipo de pruebas como "apoyo al equipo", ayudando a los programadores a saber qué código escribir a continuación. Brian también acuñó el término "pruebas orientadas a la tecnología". pruebas que caen dentro del dominio del programador y se describen usando términos y jerga del programador. En la Parte II, presentamos los cuadrantes de pruebas ágiles y examinamos las diferentes categorías de pruebas ágiles. Si quieres obtener más información sobre cómo escribir pruebas unitarias y de componentes, y TDD, la bibliografía lo orientará hacia algunos buenos recursos.

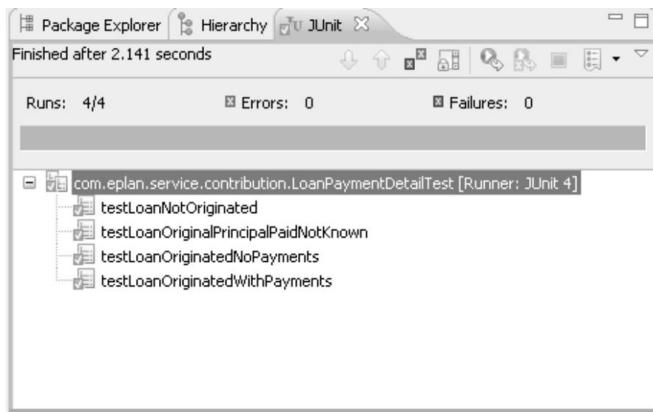


Figura 1-2 Salida de prueba unitaria de muestra

Si desea saber cómo los valores, principios y prácticas ágiles aplicados a las pruebas pueden ayudarlo, como evaluador, a hacer su mejor trabajo y ayudar a su equipo a cumplir. Para obtener más valor comercial, siga leyendo. Si te has molestado en recoger este libro, probablemente sea el tipo de profesional que se esfuerza continuamente por crecer y aprender. Es probable que tenga la mentalidad que un buen equipo ágil necesita para tener éxito. Este libro le mostrará formas de mejorar el producto de su organización, proporcionar el mayor valor posible a su equipo y disfrutar de su trabajo.

La historia de Lisa

Durante un descanso del trabajo en este capítulo, hablé con un amigo que trabaja en control de calidad para una gran empresa. Era una época del año muy ocupada y la dirección esperaba que todos trabajaran horas extra. Dijo: "Si pensara que trabajar 100 horas extra resolvería nuestros problemas, trabajaría hasta las 7 todas las noches hasta terminarlo. Pero la verdad es que podrían ser necesarias 4.000 horas adicionales para resolver nuestros problemas, por lo que trabajar más no tiene sentido". ¿Te suena esto familiar?

—Lisa

Si ha trabajado en la industria del software durante mucho tiempo, probablemente haya tenido la oportunidad de sentirse amigo de Lisa. Trabajar más duro y por más tiempo no ayuda cuando tu tarea es imposible de lograr. El desarrollo ágil reconoce la realidad de que solo tenemos un número limitado de horas productivas buenas en un día o semana y que no podemos ignorar la inevitabilidad del cambio.

El desarrollo ágil nos anima a resolver nuestros problemas en equipo. Negocio Las personas, programadores, evaluadores, analistas (todos los involucrados en el desarrollo de software) deciden juntos cuál es la mejor manera de mejorar su producto. Lo mejor de todo, como probadores, estamos trabajando junto con un equipo de personas que se sienten responsables por ofrecer la mejor calidad posible y que están todos centrados en las pruebas. Nosotros Me encanta hacer este trabajo y a ti también lo harás.

Cuando decimos "pruebas ágiles" en este libro, generalmente nos referimos a pruebas orientadas al negocio, pruebas que definen las características y funcionalidades deseadas por los expertos en negocios. Consideramos que "orientación al cliente" es sinónimo de "orientación empresarial". Las "pruebas" en este libro también incluyen pruebas que critican el producto y se enfocan en descubrir lo que podría faltar en el producto terminado para que podamos mejorarlo. Incluye casi todo más allá de la unidad y el componente. Pruebas de nivel: funcional, sistema, carga, rendimiento, seguridad, estrés, usabilidad, exploratorio, de extremo a extremo y de aceptación del usuario. Todos estos tipos de pruebas podrían ser apropiado para cualquier proyecto determinado, ya sea un proyecto ágil o uno que utilice metodologías más tradicionales.

Las pruebas ágiles no significan sólo probar en un proyecto ágil. Algunos enfoques de prueba, como las pruebas exploratorias, son inherentemente ágiles, ya sea que se realicen un proyecto ágil o no. Probar una aplicación con un plan para aprender sobre ella como usted va, y dejar que esa información guíe sus pruebas, está en consonancia con la valoración software que funcione y responda al cambio. Los capítulos posteriores analizan la agilidad. formas de prueba, así como prácticas de "pruebas ágiles".

UN PEQUEÑO CONTEXTO PARA ROLES Y ACTIVIDADES EN UN EQUIPO ÁGIL

Hablaremos mucho en este libro sobre el "equipo de clientes" y el "equipo de desarrolladores". equipo." La diferencia entre ellos son las habilidades que aportan para ofrecer un producto.

Equipo de atención al cliente

El equipo del cliente incluye expertos en negocios, propietarios de productos, expertos en dominios, gerentes de productos, analistas de negocios, expertos en la materia, todos ellos en el lado "comercial" de un proyecto. El equipo del cliente escribe las historias. o conjuntos de funciones que ofrece el equipo de desarrolladores. Proporcionan los ejemplos que impulsará la codificación en forma de pruebas empresariales. Se comunican y colaboran con el equipo de desarrolladores a lo largo de cada iteración, respondiendo preguntas, dibujando ejemplos en la pizarra y revisando historias terminadas o partes de historias.

Los evaluadores son miembros integrales del equipo del cliente, ayudan a obtener requisitos y ejemplos y ayudan a los clientes a expresar sus requisitos como pruebas.

Equipo de desarrolladores

Todos los involucrados en la entrega del código son desarrolladores y forman parte del equipo de desarrolladores. Los principios ágiles alientan a los miembros del equipo a asumir múltiples actividades; cualquier miembro del equipo puede asumir cualquier tipo de tarea. Muchos profesionales ágiles desaconsejan los roles especializados en los equipos y alientan a todos los miembros del equipo a transferir sus habilidades a otros tanto como sea posible. Sin embargo, cada equipo necesita decidir qué experiencia requieren sus proyectos. programadores, administradores de sistemas, arquitectos, administradores de bases de datos, redactores técnicos, especialistas en seguridad y personas que usan más de uno de estos sombreros podrían Ser parte del equipo, física o virtualmente.

Los evaluadores también forman parte del equipo de desarrolladores, porque las pruebas son un componente central del desarrollo ágil de software. Los evaluadores abogan por la calidad en nombre de el cliente y ayudar al equipo de desarrollo a ofrecer el máximo Valor de negocio.

Interacción entre los equipos de clientes y desarrolladores

Los equipos de clientes y desarrolladores trabajan en estrecha colaboración en todo momento. Idealmente, son sólo un equipo con un objetivo común. Ese objetivo es entregar valor a la organización. Los proyectos ágiles progresan en iteraciones, que son pequeños ciclos de desarrollo que normalmente duran de una a cuatro semanas. El equipo de atención al cliente, Con el aporte de los desarrolladores, priorizará las historias a desarrollar y el equipo de desarrolladores determinará cuánto trabajo pueden realizar. ellos trabajar juntos para definir requisitos con pruebas y ejemplos, y escribir el código que hace pasar las pruebas. Los evaluadores tienen un pie en cada mundo, entendiendo el punto de vista del cliente así como las complejidades de la implementación técnica (ver Figura 1-3).

Algunos equipos ágiles no tienen miembros que se definan a sí mismos como "probadores". Sin embargo, todos necesitan que alguien ayude al equipo del cliente a escribir pruebas comerciales para las historias de la iteración, asegurarse de que las pruebas pasen y asegurarse de que que se automaticen pruebas de regresión adecuadas. Incluso si un equipo tiene probadores, Todo el equipo ágil es responsable de estas tareas de prueba. Nuestra experiencia con equipos ágiles ha demostrado que las habilidades y la experiencia en pruebas son vitales para éxito del proyecto y que los evaluadores agregan valor a los equipos ágiles.

Interacción de roles

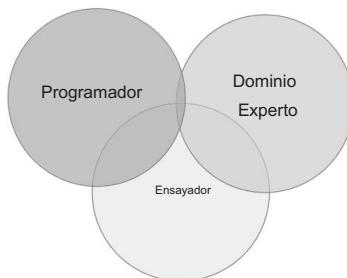


Figura 1-3 Interacción de roles

¿EN QUÉ SE DIFERENCIAN LAS PRUEBAS ÁGILES ?

Ambos empezamos a trabajar en equipos ágiles con el cambio de milenio. Como muchos evaluadores que son nuevos en Agile, al principio no sabíamos qué esperar. Junto con nuestros respectivos equipos ágiles, hemos trabajado y hemos aprendido mucho sobre pruebas en proyectos ágiles. También hemos implementado ideas y prácticas sugeridas por otros equipos y evaluadores ágiles. A lo largo de los años, también hemos compartido nuestras experiencias con otros evaluadores ágiles. Hemos facilitado talleres y dirigido tutoriales en conferencias ágiles y de pruebas, hemos hablado con grupos de usuarios locales y nos hemos unido a innumerables debates sobre listas de correo de pruebas ágiles. A través de estas experiencias, hemos identificado diferencias entre las pruebas en equipos ágiles y las pruebas en proyectos tradicionales de desarrollo en cascada. El desarrollo ágil ha transformado la profesión de las pruebas de muchas maneras.

Trabajar en equipos tradicionales Ni trabajar

estrechamente con programadores ni involucrarnos en un proyecto desde las primeras fases era nuevo para nosotros. Sin embargo, estábamos acostumbrados a aplicar estrictamente fases cerradas de un ciclo de vida de desarrollo de software estrechamente definido, comenzando con la planificación del lanzamiento y la definición de requisitos y generalmente terminando con una fase de prueba apresurada y un lanzamiento retrasado. De hecho, a menudo nos vimos obligados a asumir el papel de guardianes, diciéndoles a los gerentes de negocios: "Lo siento, los requisitos están congelados; Podemos agregar esa característica en la próxima v

Como líderes de equipos de control de calidad, a menudo también se esperaba que actuáramos como guardianes de la calidad. No podíamos controlar cómo se escribía el código, ni siquiera si algún programador probaba su código, salvo mediante nuestros esfuerzos personales de colaboración. Se esperaba que nuestras fases de prueba posteriores al desarrollo aumentaran la calidad una vez completado el código. Teníamos la ilusión de control. Por lo general, teníamos las claves de la producción y, a veces, teníamos el poder de posponer los lanzamientos o impedir que siguieran adelante. Lisa incluso tenía el título de "Jefa de Calidad", cuando en realidad era simplemente la gerente del equipo de control de calidad.

Nuestros ciclos de desarrollo fueron generalmente largos. Los proyectos en una empresa que produce software de bases de datos pueden durar un año. Los ciclos de lanzamiento de seis meses que experimentó Lisa en una nueva empresa de Internet parecieron cortos en ese momento, aunque todavía era mucho tiempo para tener los requisitos congelados. A pesar de mucho proceso y disciplina, de completar diligentemente una fase antes de pasar a la siguiente, había mucho tiempo para que la competencia saliera adelante y las aplicaciones no siempre eran las que esperaban los clientes.

Los equipos tradicionales se centran en garantizar que se cumplan todos los requisitos especificados. se entregan en el producto final. Si no todo está listo para la fecha de lanzamiento prevista original, el lanzamiento generalmente se pospone. Los equipos de desarrollo Por lo general, no tengo información sobre qué funciones se incluyen en la versión o cómo Debería trabajar. Los programadores individuales tienden a especializarse en un área particular. del código. Los evaluadores estudian los documentos de requisitos para redactar su prueba. planes y luego esperan que les entreguen el trabajo para probarlo.

Trabajando en equipos ágiles

La transición a las iteraciones cortas de un proyecto ágil podría producir resultados iniciales. conmoción y asombro. ¿Cómo podemos definir los requisitos y luego probarlos y ¿Entregar código listo para producción en una, dos, tres o cuatro semanas? Esto es particularmente difícil para organizaciones más grandes con equipos separados para diferentes funciones y aún más difícil para equipos que están dispersos geográficamente. Donde hacer ¿Todos estos diversos programadores, evaluadores, analistas, gerentes de proyectos e innumerables especialidades encajan en un nuevo proyecto ágil? ¿Cómo podemos codificar y probar de esa manera? ¿rápidamente? ¿Dónde encontraríamos tiempo para esfuerzos difíciles como la automatización? pruebas? ¿Qué control tenemos sobre el código incorrecto que se entrega a producción?

Compartiremos las historias de nuestras primeras experiencias ágiles para mostrarle que todo el mundo tiene que empezar por algún lado.

La historia de Lisa

Mi primer equipo ágil adoptó la Programación Extrema (XP), no sin algunas "experiencias de aprendizaje". Ser el único evaluador profesional en un equipo de ocho programadores que no habían aprendido a automatizar pruebas unitarias fue desalentador. La primera iteración de dos semanas fue como saltar por un precipicio.

Afortunadamente, teníamos un buen entrenador, una formación excelente, una comunidad solidaria de profesionales ágiles con ideas para compartir y tiempo para aprender. Juntos descubrimos algunos pormenores de cómo integrar las pruebas en un proyecto ágil; de hecho, cómo impulsar el proyecto con pruebas. Aprendí cómo podía utilizar mis habilidades y experiencia en pruebas para agregar valor real a un equipo ágil.

Lo más difícil de aprender para mí (el ex Jefe de Calidad) fue que los clientes, no yo, decidían los criterios de calidad para el producto. Me horroricé después de la primera iteración para descubrir que el código fallaba fácilmente cuando dos usuarios iniciaban sesión simultáneamente. Mi entrenador me explicó pacientemente, a pesar de mis estridentes objeciones, que nuestro cliente, una empresa nueva, quería poder mostrar funciones a clientes potenciales. La fiabilidad y la robustez aún no eran el problema.

Aprendí que mi trabajo era ayudar a los clientes a decirnos qué era valioso para ellos durante cada iteración y escribir pruebas para garantizar que eso fuera lo que obtenían.

—Lisa

La historia de Janet

Mi primera incursión en el mundo ágil también fue un compromiso con la Programación Extrema (XP). Acababa de llegar de una organización que practicaba la cascada con algunas prácticas extremadamente malas, incluida la de darle al equipo de prueba aproximadamente un día para probar seis meses de código. En mi siguiente trabajo como gerente de control de calidad, el gerente de desarrollo y yo estábamos aprendiendo lo que realmente significaba XP. Creamos con éxito un equipo que trabajó bien en conjunto y logramos automatizar la mayoría de las pruebas de funcionalidad. Cuando la organización se redujo durante la crisis de las puntocom, me encontré en una nueva posición en otra organización como el único evaluador con unos diez desarrolladores en un proyecto XP.

En mi primer día del proyecto, Jonathan Rasmusson, uno de los desarrolladores, se me acercó y me preguntó por qué estaba allí. El equipo estaba practicando XP y los programadores practicaban primero las pruebas y automatizaban todas sus propias pruebas. Participar en eso fue un desafío al que no pude resistirme. El equipo no sabía qué valor podía agregar, pero yo sabía que tenía habilidades únicas que podrían ayudar al equipo. Esa experiencia cambió mi vida para siempre, porque comprendí los matices de un proyecto ágil y entonces determiné que el trabajo de mi vida era hacer que el rol de tester fuera más satisfactorio.

—Janet

Lea la historia de Jonathan

Jonathan Rasmusson, ahora entrenador ágil en Rasmusson Software Consulting, pero compañero de trabajo de Janet en su segundo equipo ágil, explica cómo aprendió cómo los evaluadores ágiles agregan valor.

Así que ahí estaba yo, un joven desarrollador J2EE entusiasmado y entusiasmado por desarrollar software de la forma en que debería desarrollarse: usando XP. Hasta que un día, llega un nuevo miembro del equipo: un evaluador. Parece que la gerencia pensó que sería bueno tener un recurso de control de calidad en el equipo.

Está bien. Entonces se me ocurrió que este pobre evaluador no tendría nada que hacer. Quiero decir, como desarrollador de un proyecto XP, estaba escribiendo las pruebas. Por lo que pude ver, el control de calidad no tenía ningún papel aquí.

Entonces, por supuesto, me acerqué, me presenté y le pregunté claramente qué iba a hacer en el proyecto, porque los desarrolladores estaban escribiendo todas las pruebas. Si bien no recuerdo exactamente cómo respondió Janet, los siguientes seis meses dejaron muy claro lo que los evaluadores pueden hacer en proyectos ágiles.

Con la automatización de los tediosos casos de prueba de condiciones límite de bajo nivel, Janet, como evaluadora, ahora tenía libertad para centrarse en áreas de valor agregado mucho mayor, como pruebas exploratorias, usabilidad y probar la aplicación en formas que los desarrolladores no habían anticipado originalmente. Ella trabajó con el

cliente para ayudar a escribir casos de prueba que definan el éxito de las próximas historias. Se asoció con desarrolladores que buscaban lagunas en las pruebas.

Pero quizás lo más importante es que ayudó a reforzar un espíritu de calidad y cultura, entregando calcamanías con caras felices a aquellos desarrolladores que habían hecho un trabajo excepcional (estas se convirtieron en insignias de honor muy solicitadas que se exhiben de manera destacada en las computadoras portátiles).

Trabajar con Janet me enseñó mucho sobre el papel que desempeñan los evaluadores en proyectos ágiles y su importancia para el equipo.

Los equipos ágiles trabajan estrechamente con la empresa y tienen un conocimiento detallado de los requisitos. Se centran en el valor que pueden ofrecer y podría tener una gran aportación a la hora de priorizar funciones. Los probadores no se sientan y esperar el trabajo; se levantan y buscan maneras de contribuir a lo largo del ciclo de desarrollo y más allá.

Si probar en un proyecto ágil era como probar en un proyecto tradicional, No sentiría la necesidad de escribir un libro. Comparemos y contrastemos estos métodos de prueba.

Pruebas tradicionales versus ágiles

Es útil comenzar observando las similitudes entre las pruebas ágiles y las pruebas en Desarrollo de software tradicional. Considere la Figura 1-4.

En el diagrama del enfoque por fases, está claro que las pruebas ocurren al final, justo antes del lanzamiento. El diagrama es idealista, porque da la impresión hay tanto tiempo para probar como para codificar. En muchos proyectos, esto no es el caso. Las pruebas se "aplastan" porque la codificación lleva más tiempo que esperado, y porque los equipos entran en un ciclo de codificación y corrección al final.

Agile es iterativo e incremental. Esto significa que los evaluadores prueban cada incremento de codificación tan pronto como finaliza. Una iteración puede ser tan corta como una semana o hasta un mes. El equipo construye y prueba un poco de código, asegurándose de que funciona correctamente y luego pasa a la siguiente pieza que necesita ser construido. Los programadores nunca se adelantan a los evaluadores, porque una historia no es "hecho" hasta que haya sido probado. Hablaremos mucho más sobre esto a lo largo del libro.

Existe una enorme variedad en los enfoques de los proyectos que adoptan los equipos ágiles. Un equipo puede estar dedicado a un solo proyecto o puede ser parte de otro.

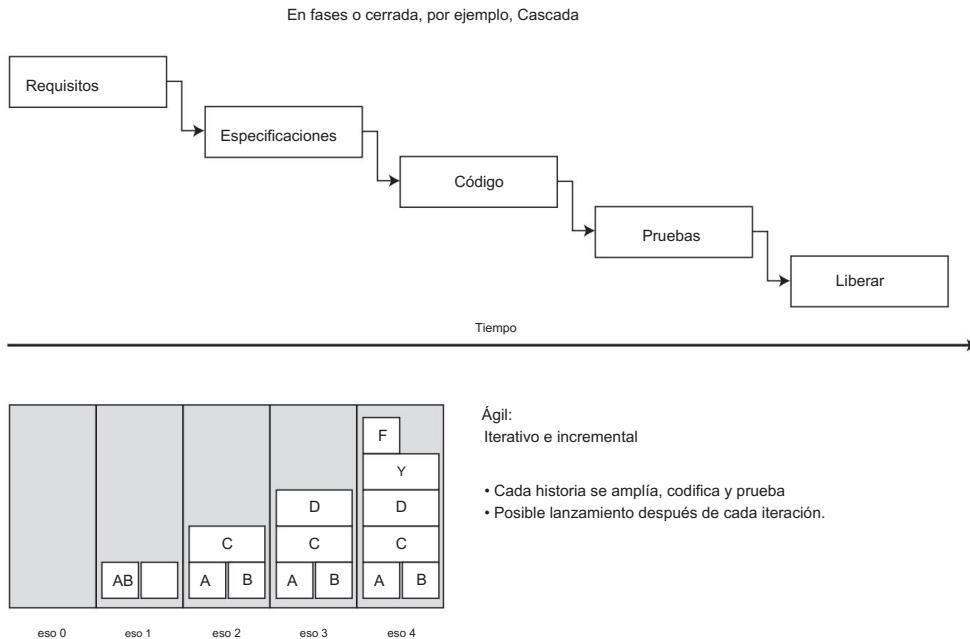


Figura 1-4 Pruebas tradicionales versus pruebas ágiles

proyecto más grande. No importa cuán grande sea tu proyecto, todavía tienes que comenzar por algún lado. Su equipo podría abordar una epopeya o una característica, un conjunto de historias relacionadas en un reunión de estimación, o podría reunirse para planificar el lanzamiento. Independientemente de cómo Cuando se inicia un proyecto o un subconjunto de un proyecto, necesitará obtener una comprensión de alto nivel del mismo. Podría idear un plan o estrategia para realizar pruebas a medida que Prepárese para un lanzamiento, pero probablemente se verá bastante diferente de cualquier prueba. plan que has hecho antes.

Cada proyecto, cada equipo y, a veces, cada iteración es diferente. Cómo su equipo resuelva los problemas debe depender del problema, las personas y las herramientas que tienes disponibles. Como miembro ágil del equipo, necesitarás ser adaptable a las necesidades del equipo.

En lugar de crear pruebas a partir de un documento de requisitos creado por analistas de negocios antes de que alguien pensara en escribir una línea de código, alguien necesitará escribir pruebas que ilustren los requisitos para cada día de historia. u horas antes de que comience la codificación. Esto suele ser un esfuerzo de colaboración entre un

experto en negocios o dominio y un evaluador, analista o algún otro desarrollador miembro del equipo. Los casos de prueba funcionales detallados, idealmente basados en ejemplos proporcionados por expertos en negocios, desarrollan los requisitos. Los probadores realizarán Pruebas exploratorias manuales para encontrar errores importantes que definieron los casos de prueba. podría perder. Los evaluadores pueden asociarse con otros desarrolladores para automatizar y ejecutar casos de prueba a medida que avanza la codificación de cada historia. Pruebas funcionales automatizadas se agregan al conjunto de pruebas de regresión. Cuando las pruebas demuestren un mínimo La funcionalidad está completa, el equipo puede considerar la historia terminada.

Si asistió a conferencias y seminarios ágiles a principios de esta década, escuchó mucho sobre TDD y las pruebas de aceptación, pero no tanto. sobre otros tipos críticos de pruebas, como carga, rendimiento, seguridad, usabilidad y otras pruebas de "ilidad". Como evaluadores, pensamos que era un poco extraño, porque todos estos tipos de pruebas son tan vitales en proyectos ágiles como lo son en proyectos que utilicen cualquier otra metodología de desarrollo. La verdadera diferencia es que nos gusta hacer estas pruebas lo más temprano posible en el proceso de desarrollo, que también pueden impulsar el diseño y la codificación.

Si el equipo realmente lanza cada iteración, como lo hace el equipo de Lisa, el último día o dos de cada iteración son el "final del juego", el momento en el que pueden ocurrir las pruebas de aceptación del usuario, la capacitación, la corrección de errores y las implementaciones en entornos de prueba. Otros equipos, como el de Janet, lanzan cada pocas iteraciones e incluso podrían Tener una iteración completa de actividades de "final del juego" para verificar el lanzamiento. preparación. La diferencia aquí es que todas las pruebas no se dejan para el final.

Como evaluador en un equipo ágil, usted es un actor clave en la liberación del código para producción, tal como lo habría sido en un entorno más tradicional. Tú podría ejecutar scripts o realizar pruebas manuales para verificar todos los elementos de una versión, como como scripts de actualización de bases de datos. Todos los miembros del equipo participan en retrospectivas u otras actividades de mejora de procesos que pueden ocurrir en cada iteración o cada versión. Todo el equipo piensa en formas de resolver problemas y mejorar procesos y prácticas.

Los proyectos ágiles tienen una variedad de sabores. ¿Tu equipo está comenzando con una limpieza? pizarra, en un (nuevo) proyecto de desarrollo totalmente nuevo? Si es así, es posible que tenga menos desafíos que un equipo que se enfrenta a reescribir o construir sobre un sistema heredado que no tiene un conjunto de regresión automatizada. Trabajar con un tercero trae desafíos de prueba adicionales para cualquier equipo.

Cualquiera que sea el tipo de desarrollo que esté utilizando, es necesario que sucedan prácticamente los mismos elementos del ciclo de vida del desarrollo de software. La diferencia

ágil es que los plazos se acortan enormemente y las actividades suceden al mismo tiempo. Los participantes, las pruebas y las herramientas deben ser adaptables.

La diferencia más importante para los evaluadores en un proyecto ágil es la rápida retroalimentación de las pruebas. Impulsa el proyecto hacia adelante y no hay guardianes.

listo para bloquear el progreso del proyecto si no se cumplen ciertos hitos.

Nos hemos encontrado con evaluadores que se resisten a la transición al desarrollo ágil, por temor a que el "desarrollo ágil" equivalga a caos, falta de disciplina, falta de documentación y un entorno hostil para los evaluadores. Mientras algunos

Los equipos parecen usar la palabra de moda "ágil" para justificar simplemente hacer lo que sea. quieren, los verdaderos equipos ágiles tienen que ver con calidad repetible además de eficiencia. Según nuestra experiencia, un equipo ágil es un lugar maravilloso para ser evaluador.

ENFOQUE DE TODO EL EQUIPO

Una de las mayores diferencias entre el desarrollo ágil y el desarrollo tradicional es el enfoque ágil de "todo el equipo". Con agilidad, no son solo los evaluadores

o un equipo de control de calidad que se sienta responsable de la calidad. no pensamos de "departamentos", solo pensamos en las habilidades y recursos que necesitamos para ofrecer el mejor producto posible. El objetivo del desarrollo ágil es producir software de alta calidad en un plazo que maximice su valor para el negocio. Este es el trabajo de todo el equipo, no solo de los evaluadores o del control de calidad designado profesionales. Todos los miembros de un equipo ágil quedan "infectados por las pruebas". Pruebas, desde el nivel de unidad hacia arriba, impulse la codificación, ayude al equipo a aprender cómo funciona la aplicación debería funcionar y avísenos cuando hayamos "terminado" con una tarea o historia.

Un equipo ágil debe poseer todas las habilidades necesarias para producir código de calidad que ofrece las características requeridas por la organización. Si bien esto podría significar incluir especialistas en el equipo, como evaluadores expertos, no limita tareas particulares a miembros concretos del equipo. Cualquier tarea puede ser completada por cualquier miembro del equipo, o un par de miembros del equipo. Esto significa que el equipo asume la responsabilidad de todo tipo de tareas de prueba, como la automatización de pruebas y las pruebas exploratorias manuales. También significa que todo el equipo piensa constantemente sobre el diseño de código para la capacidad de prueba.

El enfoque de todo el equipo implica una colaboración constante. Los evaluadores colaboran con los programadores, el equipo del cliente y otros especialistas del equipo, y no sólo para tareas de prueba, sino también para otras tareas relacionadas con las pruebas, como la construcción infraestructura y diseño para la capacidad de prueba. La Figura 1-5 muestra a un desarrollador revisando informes con dos clientes y un evaluador (no se muestra en la imagen).



Figura 1-5 Un desarrollador analiza un problema con los clientes

El enfoque de equipo completo significa que todos asumen la responsabilidad de las pruebas. tareas. Significa que los miembros del equipo tienen una variedad de habilidades y experiencia para emplear en desafíos como el diseño para la capacidad de prueba, convirtiendo ejemplos en pruebas y en código para hacer que esas pruebas pasen. Estos diversos puntos de vista sólo pueden significar mejores pruebas y cobertura de pruebas.

Lo más importante es que en un equipo ágil cualquiera puede solicitar y recibir ayuda. El equipo se compromete a proporcionar el mayor valor comercial posible como equipo, el equipo hace todo lo necesario para lograrlo. Algunas personas que son nuevas en Agile lo perciben como una cuestión de velocidad. El hecho es que todo es cuestión de calidad, y si es así Si no, nos preguntamos si realmente se trata de un equipo "ágil".

Tu situación es única. Por eso es necesario ser consciente del potencial Probar los obstáculos que su equipo podría enfrentar y cómo puede aplicar valores ágiles. y principios para superarlos.

RESUMEN

Comprender las actividades que realizan los evaluadores en equipos ágiles le ayudará Muestre a su propio equipo el valor que los evaluadores pueden agregar. Aprender las prácticas básicas de las pruebas ágiles ayudará a su equipo a ofrecer software que deleite a sus clientes.

En este capítulo, hemos explicado a qué nos referimos cuando usamos el término "ágil pruebas.

Mostramos cómo se relaciona el Manifiesto Ágil con las pruebas, con su énfasis en los individuos y las interacciones, el software en funcionamiento, la colaboración con el cliente y la respuesta al cambio.

Proporcionamos algo de contexto para este libro, incluidos algunos otros términos que utilizamos, como "probador", "programador", "cliente" y términos relacionados para que podamos hablar un lenguaje común.

Explicamos en qué se diferencian las pruebas ágiles, que se centran en el valor empresarial y en ofrecer la calidad que los clientes requieren, de las pruebas tradicionales, que se centran en el cumplimiento de los requisitos.

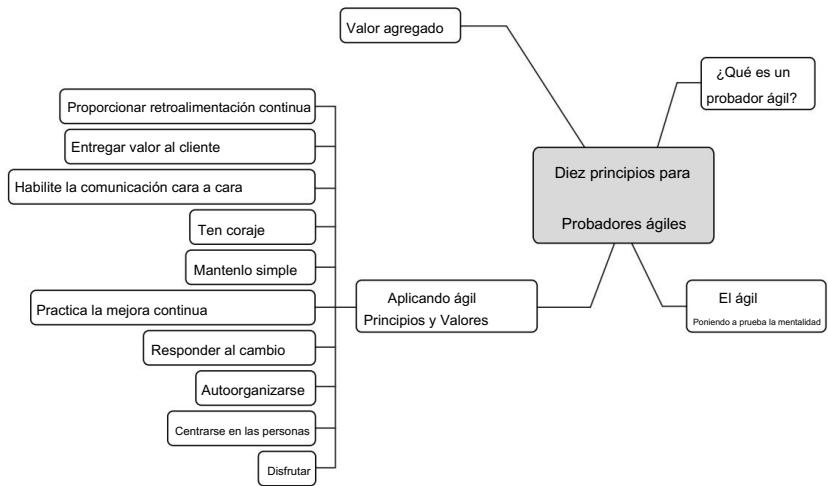
Introdujimos el enfoque de "equipo completo" para las pruebas ágiles, lo que significa que todos los involucrados en la entrega de software son responsables de entregar software de alta calidad.

Recomendamos adoptar un enfoque práctico aplicando valores y principios ágiles para superar los obstáculos de las pruebas ágiles que surgen en su situación particular.

Esta página se dejó en blanco intencionalmente.

Capítulo 2

DIEZ PRINCIPIOS PARA PROBADORES ÁGILES



Todos los miembros de un equipo ágil son evaluadores. Cualquiera puede realizar tareas de prueba. Si eso es cierto, ¿qué tiene de especial un evaluador ágil? Si me defino como tester en un equipo ágil, ¿qué significa eso realmente? Los evaluadores ágiles necesitan conjuntos de habilidades diferentes a los de los equipos tradicionales? ¿Qué los guía en sus actividades diarias?

En este capítulo, hablamos sobre la mentalidad de las pruebas ágiles, mostramos cómo los valores y principios ágiles guían las pruebas y brindamos una descripción general de cómo los evaluadores agregan valor a los equipos ágiles.

¿QUÉ ES UN TESTER ÁGIL ?

Definimos un evaluador ágil de esta manera: un evaluador profesional que acepta el cambio, colabora bien con personal técnico y empresarial y comprende el concepto de utilizar pruebas para documentar los requisitos e impulsar el desarrollo.

Los evaluadores ágiles tienden a tener buenas habilidades técnicas, saben cómo colaborar con

otros para automatizar pruebas y también son probadores exploratorios experimentados.

Están dispuestos a aprender qué hacen los clientes para poder comprender mejor sus requisitos de software.

¿Quién es un evaluador ágil? Ella es un miembro del equipo que impulsa pruebas ágiles. Sabemos muchos probadores ágiles que comenzaron en alguna otra especialización. Un desarrollador se infecta con las pruebas y se ramifica más allá de las pruebas unitarias. Un exploratorio El evaluador, acostumbrado a trabajar de forma ágil, se siente atraído por la idea de un equipo ágil. Los profesionales que desempeñan otros roles, como analistas funcionales o de negocios, pueden compartir los mismos rasgos y realizar gran parte del mismo trabajo.

Las habilidades son importantes, pero la actitud cuenta más. A Janet le gusta decir: "Sin la actitud, la habilidad no es nada". Habiendo tenido que contratar numerosos probadores para nuestros equipos ágiles, hemos pensado mucho en esto y lo hemos discutido con otros en la comunidad ágil. Los evaluadores tienden a ver el panorama general. Ellos miran la aplicación más desde el punto de vista del usuario o del cliente, lo que significa que están generalmente centrado en el cliente.

LA MENTALIDAD DE PRUEBAS ÁGILES

¿Qué hace que un equipo sea "ágil"? Para nosotros, un equipo ágil es aquel que continuamente se enfoca en hacer su mejor trabajo y entregar el mejor producto posible. En nuestra experiencia, esto implica mucha disciplina, aprendizaje, tiempo, experimentación, y trabajando juntos. No es para todos, pero es ideal para aquellos de nosotros que Me gusta la dinámica del equipo y nos centramos en la mejora continua.

Los proyectos exitosos son el resultado de personas buenas a las que se les permite hacer un buen trabajo. Las características que hacen que alguien tenga éxito como evaluador en un equipo ágil son probablemente las mismas características que hacen que un probador sea altamente valorado en cualquier equipo.

Un evaluador ágil no se ve a sí mismo como un oficial de policía de calidad que protege a sus clientes de un código inadecuado. Está lista para recopilar y compartir información, para trabajar con el cliente o propietario del producto para ayudarlo a expresar sus requisitos adecuadamente para que puedan obtener las funciones que necesitan y proporcionar retroalimentación sobre el progreso del proyecto a todos.

Los evaluadores ágiles, y tal vez cualquier evaluador con las habilidades y la mentalidad adecuadas, son buscando continuamente formas en que el equipo pueda hacer un mejor trabajo en la producción de software de alta calidad. A nivel personal, eso podría significar atender a usuarios locales, reuniones de grupo o mesas redondas para saber qué están haciendo otros equipos. Él

También significa probar nuevas herramientas para ayudar al equipo a hacer un mejor trabajo al especificar, ejecutar y automatizar los requisitos del cliente como pruebas.

La conclusión es que los evaluadores ágiles, al igual que sus compañeros de equipo ágiles, disfrutan aprendiendo. nuevas habilidades y asumir nuevos desafíos, y no se limitan a resolviendo solo problemas de prueba. Esto no es sólo un rasgo de los evaluadores; lo vemos en todo ágil miembros del equipo. Los evaluadores ágiles ayudan a los equipos de desarrolladores y clientes a abordar cualquier tipo de problema que pueda surgir. Los evaluadores pueden proporcionar información que ayude el equipo mira hacia atrás y aprende qué funciona y qué no.

Creatividad, apertura a ideas, disposición para asumir cualquier tarea o rol, centrarse en el cliente y una visión constante del panorama general son sólo algunos de los componentes de la mentalidad de prueba ágil. Los buenos evaluadores tienen instinto y comprensión. dónde y cómo podría fallar el software y cómo localizar las fallas.

Los evaluadores pueden tener conocimientos y experiencia especiales en pruebas, pero un buen evaluador ágil no tiene miedo de lanzarse a una discusión sobre diseño con sugerencias que ayudará a la capacidad de prueba o creará una solución más elegante. Una mentalidad de prueba ágil es aquella que está orientada a resultados, artesanal, colaborativa y deseosa de aprender y apasionado por entregar valor comercial de manera oportuna.

APLICAR PRINCIPIOS Y VALORES ÁGILES

Los individuos pueden tener un gran impacto en el éxito de un proyecto. esperaríamos un equipo con miembros más experimentados y más capacitados para superar a un equipo con menos talento. Pero un equipo es más que sus miembros individuales. Valores ágiles y principios promueven un enfoque en las personas involucradas en un proyecto y cómo interactúan y se comunican. Un equipo que se guía con valores ágiles y principios tendrán una moral de equipo más alta y una mejor velocidad que un equipo mal equipo funcional de individuos talentosos.

Las cuatro declaraciones de valores del Manifiesto Ágil, que presentamos en la comienzo del primer capítulo, muestre preferencias, no ultimátums, y no haga declaraciones sobre qué hacer o no hacer. El Manifiesto Ágil también incluye un Lista de principios que definen cómo abordamos el desarrollo de software. nuestra lista de los principios de "pruebas" ágiles se deriva parcialmente de esos principios. Porque Ambos venimos de la cultura de programación extrema, hemos adoptado muchos de sus valores y principios subyacentes. También hemos incorporado pautas y principios que han funcionado para nuestros equipos. Los valores propios de su equipo y Los principios le guiarán a la hora de elegir prácticas y tomar decisiones sobre como quieres trabajar.

Los principios que creemos que son importantes para un evaluador ágil son:

- Proporcionar retroalimentación continua.
- Entregar valor al cliente.
- Permitir la comunicación cara a cara.
- Ten coraje.
- Mantenlo simple.
- Practicar la mejora continua.
- Responder al cambio.
- Autoorganizarse.
- Concéntrate en las personas.
- Disfrutar.

Proporcionar retroalimentación continua

Dado que las pruebas impulsan proyectos ágiles, no sorprende que la retroalimentación desempeñe un papel importante formar parte de cualquier equipo ágil. El papel tradicional del evaluador como "proveedor de información" la hace inherentemente valiosa para un equipo ágil. Uno de los probadores más ágiles contribuciones importantes es ayudar al propietario del producto o al cliente a articular requisitos para cada historia en forma de ejemplos y pruebas. el probador luego trabaja junto con sus compañeros de equipo para convertir esos requisitos en pruebas ejecutables. Los evaluadores, programadores y otros miembros del equipo trabajan para ejecutar estos pruebas tempranamente y con frecuencia para que estén continuamente guiados por comentarios significativos. Dedicaremos mucho tiempo en este libro a explicar formas de hacer esto.

Cuando el equipo encuentra obstáculos, la retroalimentación es una forma de ayudar a eliminarlos. a ellos. ¿Ofrecemos una interfaz de usuario que no cumplió del todo con las expectativas del cliente? Escribamos una tarjeta de tareas que nos recuerde colaborar con el cliente. en prototipos en papel de la próxima historia de UI.

¿Está preocupada la dirección por el progreso del trabajo? Mostrar un gran visible tabla de pruebas escritas, ejecutadas y aprobadas todos los días. Muestre una cobertura de funcionalidad de panorama general, como matrices de prueba. ¿Tiene problemas para estabilizar la construcción? El equipo de Lisa mostró el número de días que quedaban hasta el momento de etiquetar el compilar para su lanzamiento a fin de mantener a todos enfocados en terminar las historias en tiempo. Después de que eso se convirtió en un hábito, ya no necesitaron la señal visual.

Entregar valor al cliente

El desarrollo ágil consiste en ofrecer valor en pequeñas versiones que proporcionen exactamente la funcionalidad que el cliente ha priorizado más recientemente. Este generalmente significa limitar el alcance. Es fácil quedar atrapado en el cliente

el deseo del equipo de contar con funciones interesantes. Cualquiera puede cuestionar estas adiciones, pero un evaluador a menudo reconoce el impacto de la historia, porque necesita pensar sobre las repercusiones de las pruebas.

La historia de Lisa

Nuestro propietario de producto participa en reuniones de planificación antes de cada iteración. Sin embargo, después de que la iteración ha comenzado y discutimos más detalles sobre las historias y cómo probarlas, a menudo plantea una idea que no surgió durante la planificación, como: "Bueno, realmente sería bueno". Sería bueno que la selección de este informe pudiera incluir X, Y y Z y ordenarse también por A". Una petición inocente puede añadir mucha complejidad a una historia. A menudo invito a uno de los programadores para hablar sobre si esta adición se puede manejar dentro del alcance de la historia que habíamos planeado. De lo contrario, le pedimos al propietario del producto que escriba una tarjeta para la siguiente iteración.

—Lisa

Los evaluadores ágiles se mantienen centrados en el panorama general. Podemos entregar lo más crítico. funcionalidad en esta iteración y agregarla más adelante. Si dejamos que aparezcan nuevas características corremos el riesgo de no entregar nada a tiempo. Si nos quedamos demasiado atrapados con el borde casos y perdemos la funcionalidad principal en el camino feliz, no proporcionaremos la valorar las necesidades del negocio.

La historia de Lisa

Para garantizar que ofrecemos algo de valor en cada iteración, nuestro equipo analiza cada historia para identificar la "ruta crítica" o la "pequeña porción" de funcionalidad necesaria. Primero completamos esas tareas y luego regresamos y desarrollamos el resto de las funciones. El peor de los casos es que solo se publique la funcionalidad principal. Eso es mejor que no entregar nada o algo que sólo funciona a medias.

—Lisa

Los evaluadores ágiles adoptan el mismo enfoque identificado en la historia de Lisa. Mientras Una de nuestras habilidades es identificar casos de prueba más allá del "camino feliz", todavía necesitamos Para empezar, asegúrese de que el camino feliz funcione. Podemos automatizar pruebas para el camino feliz, y agregue pruebas negativas y de límites más adelante. Considere siempre lo que agrega el mayor valor al cliente y comprende su contexto. Si una aplicación es crítica para la seguridad, es absolutamente necesario agregar pruebas negativas. El El tiempo de prueba debe considerarse durante el proceso de estimación para hacer Asegúrese de que se asigne suficiente tiempo en la iteración para ofrecer una característica "segura".

Habilite la comunicación cara a cara

Ningún equipo funciona bien sin una buena comunicación. Hoy, cuando tantos Los equipos están distribuidos en múltiples ubicaciones geográficas, la comunicación es

aún más vital y más desafiante. El evaluador ágil debe buscar formas únicas de facilitar la comunicación. Es un aspecto crítico para hacerla. trabajo bien.

La historia de Janet

Cuando trabajaba con un equipo, teníamos un problema real con los programadores que hablaban con el propietario del producto y dejaban a los evaluadores fuera de la discusión. A menudo se enteraban de los cambios después del hecho. Parte del problema fue que los desarrolladores no estuvieron sentados con los evaluadores debido a problemas logísticos. Otro problema fue la historia. El equipo de prueba era nuevo y el propietario del producto estaba acostumbrado a acudir directamente a los programadores.

Llevé el problema al equipo y creamos una regla. Encontramos un gran éxito con el “Poder de Tres”. Esto significaba que todas las discusiones sobre una característica necesitaban un programador, un evaluador y el propietario del producto. Era responsabilidad de cada persona asegurarse de que siempre hubiera un representante de cada grupo. Si alguien veía a dos personas hablando, tenía derecho a intervenir en la conversación. No pasó mucho tiempo antes de que se convirtiera en una simple rutina y nadie consideraría dejar al evaluador fuera de una discusión. Esto funcionó para nosotros porque el equipo aceptó la solución.

—Janet

Cada vez que surge una pregunta sobre cómo debería funcionar una característica o cómo debería verse una interfaz, el evaluador puede recurrir a un programador y a un experto en negocios. para hablar de ello. Los evaluadores nunca deben interferir en ninguna comunicación directa entre el cliente y el desarrollador, pero a menudo pueden ayudar a garantizar que se produzca la comunicación.

Los evaluadores ágiles ven cada historia o tema desde el punto de vista del cliente, pero También comprender los aspectos técnicos y las limitaciones relacionadas con la implementación. características. Pueden ayudar a los clientes y desarrolladores a lograr un lenguaje común. La gente de negocios y la gente de software a menudo hablan idiomas diferentes. Tienen que encontrar puntos en común para poder trabajar juntos con éxito. Los evaluadores pueden ayudarlos a desarrollar un idioma compartido, un dialecto del proyecto o jerga del equipo.

Brian Marick (2004) recomienda que utilicemos ejemplos para desarrollar este lenguaje. Cuando el equipo de Lisa se desvía hacia una discusión filosófica durante una En la reunión de planificación de sprint, Lisa le pide al propietario del producto un ejemplo o un escenario de uso. Los evaluadores pueden fomentar las discusiones en la pizarra para que funcionen más ejemplos. Estos ayudan a los clientes a visualizar mejor sus necesidades. claramente. También ayudan a los desarrolladores a producir código bien diseñado para cumplir esos requisitos.

La comunicación cara a cara no tiene sustituto. El desarrollo ágil depende en una colaboración constante. Al igual que otros miembros del equipo ágil, las personas que hacen Las tareas de prueba buscarán continuamente al cliente y a los miembros del equipo técnico. para discutir y colaborar. Cuando un evaluador ágil sospecha que hay una suposición oculta o un requisito mal entendido, buscará un cliente y un desarrollador. hablando de ello. Si personas en un edificio o continente diferente necesitan hablar, buscan formas creativas de reemplazar las conversaciones cara a cara en tiempo real.

Ten coraje

El coraje es un valor fundamental en XP, y prácticas como la automatización de pruebas y la integración continua permiten que el equipo practique este valor. Los desarrolladores tienen el coraje de hacer cambios y refactorizar el código porque tienen la red de seguridad de una suite de regresión automatizada. En esta sección hablamos de el coraje emocional que se necesita al hacer la transición a un equipo ágil.

¿Ha trabajado en una organización donde los evaluadores estaban atrapados en sus propios silo, incapaz de hablar ni con las partes interesadas del negocio ni con otros miembros de la Equipo técnico? Si bien es posible que aproveches la oportunidad de unirte a una colaboración En un entorno ágil, es posible que se sienta incómodo al tener que pedirle ejemplos al cliente o pedirle a un programador que lo ayude a automatizar una prueba o mencionar un control de carretera durante el stand-up diario.

Cuando te unes por primera vez a un equipo ágil, o cuando tu equipo actual hace la primera transición al desarrollo ágil, es normal experimentar miedo y tener una lista de preguntas que necesitan ser respondidas. ¿Cómo diablos vamos a poder para completar las tareas de prueba para cada historia en tan poco tiempo? ¿Cómo serán las pruebas? ¿“mantenerse al día” con el desarrollo? ¿Cómo sabes cuántas pruebas son? ¿suficiente? O tal vez eres un gerente de pruebas funcionales o un proceso de calidad. gerente y no tiene claro dónde encaja ese rol en un equipo ágil, y nadie tiene las respuestas. Los evaluadores ágiles necesitan coraje para encontrar las respuestas a esas preguntas, pero también hay otras razones para tener coraje.

Necesitamos coraje para permitirnos fallar, sabiendo que al menos fallaremos rápido y poder aprender de ese fracaso. Después de haber desperdiciado una iteración porque no obtuvimos una compilación estable, comenzaremos a pensar en formas de asegurarnos de que no vuelva a suceder.

Necesitamos coraje para permitir que otros cometan errores, porque esa es la única manera de aprender la lección.

La historia de Lisa

Trabajé en un proyecto en el que el entrenador ágil insistía en que estuviera en un equipo de pruebas separado (¡a menudo un equipo de una sola persona!) cuyo trabajo no estaba incluido en el equipo de programadores. seguimiento y velocidad. Tenía que seguir adelante y probar esto. Después de que la versión tuvo problemas porque las pruebas no habían terminado, le pregunté al entrenador si podíamos probar las cosas a mi manera durante una iteración o dos. El enfoque de todo el equipo funcionó mucho mejor. Cada historia fue probada y "terminada" al final de la iteración, y los clientes quedaron mucho más contentos con los resultados.

—Lisa

Necesitamos valor para pedir ayuda, especialmente cuando la persona que podría brindarla parece bastante ocupada y estresada. Saliendo de su antiguo silo y unirse a un equipo, la responsabilidad del éxito o el fracaso requiere coraje. Hacer una pregunta o señalar lo que cree que es un defecto requiere coraje, incluso en un equipo apoyado en valores y principios ágiles. no lo seas ¡asustado! Los equipos ágiles son abiertos y generalmente aceptan nuevas ideas.

Mantenlo simple

La programación extrema explicada de Kent Beck nos aconsejó hacer lo más simple algo que posiblemente podría funcionar. Eso no significa que lo primero que intentes Realmente funciona, pero debería ser simple.

Los evaluadores ágiles y sus equipos enfrentan el desafío no solo de producir lo más simple posible implementación de software, pero adoptar un enfoque simple para garantizar que el software cumpla con los requisitos del cliente. Esto no significa que el equipo no debería tomarse algún tiempo para analizar temas e historias y pensar. a través de la arquitectura y el diseño adecuados. Significa que el equipo Es posible que deba recurrir al lado comercial del equipo cuando sus requisitos pueden ser un poco elaborados y una solución más simple entregará la solución. mismo valor.

Algunos de nosotros trabajamos en organizaciones de software donde nosotros, como evaluadores y responsables de calidad. Se pidió al personal de aseguramiento que estableciera estándares de calidad. Creemos que esto es al revés, porque depende del equipo del cliente decidir qué nivel de calidad desea. quieras pagar. Los evaluadores y otros miembros del equipo deben proporcionar información a los clientes y ayudarles a considerar todos los aspectos de la calidad, incluidos los requisitos no funcionales como el rendimiento y la seguridad. Las decisiones finales Dependen del cliente. El equipo puede ayudar al cliente a tomar buenas decisiones. adoptando un enfoque sencillo y gradual para su trabajo. Medios de prueba ágiles



Capítulo 9, "Kit de herramientas para pruebas empresariales que respalden al equipo" y Capítulo 11, "Críticas el producto utilizando pruebas tecnológicas", brinde ejemplos de herramientas de prueba.



La Parte IV, "Automatización de pruebas", explica cómo crear una estrategia de automatización de pruebas "factible".

haciendo las pruebas más simples posibles para verificar que existe una pieza de funcionalidad o que se ha cumplido el estándar de calidad del cliente (por ejemplo, rendimiento).

Simple no significa fácil. Para los evaluadores, significa probar "solo lo suficiente" con el herramientas y técnicas más livianas que podamos encontrar y que hagan el trabajo. Herramientas Puede ser tan simple como una hoja de cálculo o una lista de verificación. Necesitamos automatizar las pruebas de regresión, pero deberíamos reducirlas al nivel más bajo posible para fomentar una retroalimentación rápida. Incluso unas simples pruebas de humo podrían ser suficientes para Automatización de pruebas orientada al negocio.

Las pruebas exploratorias se pueden utilizar para conocer su aplicación y su hurón. elimine errores difíciles de encontrar, pero comience con lo básico, viajes secundarios en el tiempo y evaluar hasta dónde llegar con los casos extremos. La simplicidad nos ayuda a mantener nuestro enfoque sobre el riesgo, el retorno de la inversión y la mejora en las áreas de mayor dolor.

Practica la mejora continua

Buscar formas de hacer un mejor trabajo es parte de la mentalidad de un evaluador ágil. De Por supuesto, todo el equipo debería pensar de esta manera, porque el núcleo central Lo ágil es que el equipo siempre intenta hacer un mejor trabajo. Los probadores participan en retrospectivas del equipo, evaluando qué está funcionando bien y qué debe mejorarse. agregado o modificado. Los evaluadores plantean problemas de prueba para que todo el equipo los aborde. Los equipos han logrado sus mayores mejoras en las pruebas y en todos otras áreas mediante el uso de prácticas de mejora de procesos, como retrospectivas y retrasos por impedimentos. Algunas ideas de mejora podrían convertirse tarjetas de tareas. Para problemas más grandes, los equipos se concentran en uno o dos temas a la vez para asegúrese de que resuelvan el problema real y no solo el síntoma.

Los evaluadores ágiles y sus equipos siempre están buscando herramientas, habilidades o prácticas que podrían ayudarles a agregar más valor u obtener un mejor retorno sobre el inversor del cliente. Las iteraciones cortas del desarrollo ágil lo hacen Es más fácil probar algo nuevo durante algunas iteraciones y ver si vale la pena. adoptar a largo plazo.

Aprender nuevas habilidades y crecer profesionalmente es importante para los evaluadores ágiles. Aprovechan los numerosos recursos gratuitos disponibles para mejorar su Habilidades especializadas, como pruebas exploratorias. Asisten a reuniones y conferencias, se unen a listas de correo y leen artículos, blogs y libros para obtener nuevas ideas. Buscan formas de automatizar (u obtener ayuda de sus compañeros de trabajo para automatizar) tareas mundanas o repetitivas para que tengan más tiempo para contribuir su valiosa experiencia.

Pierre Veragren, líder de SQA en iLevel de Weyerhaeuser, identificó una calidad Nosotros mismos vemos a menudo en equipos ágiles: "AADD", trastorno por déficit de atención ágil. Todo lo que no se aprenda rápidamente puede considerarse inútil. Los miembros del equipo ágil buscan el retorno de la inversión y, si no lo ven rápidamente, se mudan. en. Esta no es una característica negativa cuando se entrega software listo para producción cada dos semanas o incluso con más frecuencia.

Las retrospectivas son una práctica ágil clave que permite al equipo utilizar la experiencia de ayer para hacer un mejor trabajo mañana. Los evaluadores ágiles aprovechan esta oportunidad para plantear problemas relacionados con las pruebas y pedir al equipo que piense en formas de abordarlos a ellos. Esta es una manera para que el equipo se brinde retroalimentación a sí mismo para una evaluación continua. mejora.

La historia de Lisa

Hablaremos más sobre las retrospectivas y cómo pueden ayudar a su equipo a practicar la mejora continua en el Capítulo 19, "Conclusión de la iteración".

Nuestro equipo había utilizado las retrospectivas con gran beneficio, pero sentíamos que necesitábamos algo nuevo que nos ayudara a centrarnos en hacer un mejor trabajo. Sugerí mantener una "acumulación de impedimentos" de elementos que nos impedían ser tan productivos como lo habíamos sido. gustaría ser. Lo primero que escribí en el trabajo pendiente de impedimento fue el lento tiempo de respuesta de nuestro entorno de prueba. Nuestro administrador de sistemas buscó un par de máquinas económicas y las convirtió en servidores nuevos y más rápidos para nuestros entornos de prueba. Nuestro administrador de base de datos analizó el rendimiento de la base de datos de prueba, descubrió que el sistema de un solo disco era el impedimento y nuestro gerente dio el visto bueno para instalar un RAID para un mejor acceso al disco. Pronto pudimos implementar compilaciones y realizar nuestras pruebas exploratorias mucho más rápido.

—Lisa

Responder al cambio

Cuando trabajábamos en un entorno de cascada, nos acostumbramos a decir: "Lo siento, no podemos hacer este cambio ahora; los requisitos están congelados. tendremos que ponlo en el primer lanzamiento del parche". Fue frustrante para los clientes porque Se dieron cuenta de que no habían hecho un gran trabajo al definir todos sus requisitos. en la delantera.

En una iteración ágil de dos semanas, es posible que tengamos que decir: "Está bien, escribe una tarjeta para eso". y lo haremos en la próxima iteración o la próxima versión", pero los clientes saben que pueden obtener su cambio cuando lo deseen porque controlan la prioridad.

Responder al cambio es un valor clave para los profesionales ágiles, pero hemos descubierto que es uno de los conceptos más difíciles para los evaluadores. La estabilidad es lo que los probadores anhelan para que puedan decir: "Lo he probado; está hecho." Cambiando continuamente Los requisitos son la pesadilla de un evaluador. Sin embargo, como evaluadores ágiles, tenemos que Bienvenido el cambio. El miércoles podríamos esperar comenzar las historias A y B.

y luego C el próximo viernes. Para el viernes, el cliente podría haber vuelto a priorizar y ahora quiere las historias A, X e Y. Mientras sigamos hablando con el cliente, podemos manejar cambios como ese porque estamos trabajando al mismo tiempo. ritmo con el resto del equipo.

Algunos equipos ágiles intentan prepararse antes de la siguiente iteración, tal vez escribir casos de prueba de alto nivel, capturar condiciones de satisfacción empresarial o documentar ejemplos. Es un asunto complicado que podría resultar en una pérdida de tiempo. si las historias se vuelven a priorizar o cambian mucho. Sin embargo, los equipos distribuidos en En particular, se necesitan ciclos de retroalimentación adicionales para prepararse para la iteración.

La historia de Lisa

El miembro de nuestro equipo remoto solía ser nuestro administrador en el sitio. Es una pieza clave para ayudar a la empresa a escribir y priorizar historias. Tiene un conocimiento profundo tanto del código como del negocio, lo que le ayuda a encontrar soluciones creativas para las necesidades comerciales. Cuando se mudó a la India, buscamos formas de conservar el beneficio de su experiencia. Las reuniones se programan en horarios en los que él puede participar y mantiene conferencias telefónicas periódicas con el propietario del producto para hablar sobre próximas historias. Hemos tenido que pasar de herramientas de baja tecnología, como fichas, a herramientas en línea que todos podemos utilizar.

Debido a que el equipo estaba dispuesto a realizar cambios en la forma en que trabajábamos y buscó herramientas que lo ayudaran a mantenerse informado sobre los cambios continuos, pudimos conservar el beneficio de su experiencia.

—Lisa

Algunos equipos cuentan con analistas que pueden dedicar más tiempo a los expertos empresariales para realizar una planificación anticipada. Cada equipo debe lograr un equilibrio entre generar ideas de soluciones con anticipación y comenzar desde cero.

el primer día de cada iteración. Los evaluadores ágiles siguen la corriente y trabajan con el equipo para adaptarse a los cambios.

Las pruebas automatizadas son una de las claves de la solución. Una cosa sabemos con certeza: no. El equipo ágil tendrá éxito realizando solo pruebas manuales. Necesitamos una automatización sólida para ofrecer valor empresarial en un plazo que lo haga valioso.

Autoorganizarse

El evaluador ágil es parte de un equipo ágil autoorganizado. La cultura del equipo imbuye la filosofía de pruebas ágiles. Cuando los programadores, administradores de sistemas, analistas, expertos en bases de datos y el equipo del cliente piensan continuamente

En cuanto a las pruebas y la automatización de pruebas, los evaluadores disfrutan de una perspectiva completamente nueva. Automatizar pruebas es difícil, pero es mucho más fácil cuando tienes a todo el equipo

trabajando juntos. Cualquier problema de prueba es más fácil de abordar cuando tienes personas con múltiples conjuntos de habilidades y múltiples perspectivas atacándolo.

La historia de Lisa

Mi equipo es un buen ejemplo de equipo autoorganizado. Cuando implementamos Scrum, teníamos un sistema heredado con errores y no había pruebas automatizadas. Realizar cambios en el código era, en el mejor de los casos, arriesgado. Nuestro gerente probablemente tenía algunas soluciones excelentes para el problema, pero no las sugirió. En cambio, exploramos los problemas y elaboramos un plan.

Los programadores comenzarían a implementar nuevas historias en una arquitectura nueva y comprobable, utilizando un desarrollo basado en pruebas. Los evaluadores escribirían scripts de prueba de regresión manual y todo el equipo (programadores, evaluadores, el administrador del sistema y el administrador de bases de datos) los ejecutaría en los dos últimos días de cada iteración. Los evaluadores (en ese momento, esto significaba yo) trabajarían en un conjunto de pruebas de humo de regresión automatizadas a través de la interfaz de usuario. Con el tiempo, la arquitectura del nuevo código nos permitiría automatizar pruebas funcionales con una herramienta como FitNesse.

Implementamos este plan en pequeños pasos, refinando nuestro enfoque en cada iteración. Utilizar las habilidades de cada miembro del equipo fue un enfoque mucho mejor que decidir la estrategia de automatización por mi cuenta.

—Lisa

Cuando un equipo ágil enfrenta un gran problema, tal vez un obstáculo en la producción o una construcción rota, es problema de todos. Los temas de mayor prioridad son problemas que todo el equipo debe resolver. Los miembros del equipo discuten el tema correctamente y decidir cómo y quién lo arreglará.

No hay duda de que el manager de Lisa podría haber ordenado que el equipo tomara este enfoque para resolver sus problemas de automatización, pero el propio equipo puede idear el plan más viable. Cuando el equipo crea su propio enfoque y se compromete con él, sus miembros adoptan una nueva actitud hacia las pruebas.

Centrarse en las personas

Los proyectos tienen éxito cuando a las buenas personas se les permite hacer su mejor trabajo. Ágil Los valores y principios fueron creados con el objetivo de permitir que individuos y éxito del equipo. Los miembros del equipo ágil deben sentirse seguros y no tener que preocuparse sobre ser culpados por errores o perder sus trabajos. Los miembros del equipo ágil se respetan unos a otros y reconocen los logros individuales. Todos en un Un equipo ágil debe tener oportunidades para crecer y desarrollar sus habilidades. Ágil Los equipos trabajan a un ritmo sostenible que les permite seguir prácticas disciplinadas. y mantener una nueva perspectiva. Como afirma el Manifiesto Ágil, valoramos a las personas y las interacciones por encima de los procesos y las herramientas.

En la historia del desarrollo de software, los evaluadores no siempre han disfrutado de la paridad con otros roles en el equipo de desarrollo. Algunas personas consideraron que los probadores fallaron programadores o ciudadanos de segunda clase en el mundo del desarrollo de software. Los evaluadores que no se molestan en aprender nuevas habilidades y crecer profesionalmente contribuyen a la percepción de que las pruebas son un trabajo poco cualificado. Incluso el término "probador" tiene Se ha evitado, con títulos de trabajo como "Ingeniero de control de calidad" o "Ingeniero de control de calidad". Analista" y nombres de equipos como "Departamento de control de calidad" tienen preferencia.

Los equipos ágiles que se adhieren a la verdadera filosofía ágil brindan a todos los miembros del equipo igual peso. Los evaluadores ágiles saben que aportan un valor único a sus equipos, y los equipos de desarrollo han descubierto que tienen más éxito cuando sus El equipo incluye personas con experiencia y habilidades específicas en pruebas. Por ejemplo, un evaluador exploratorio experto puede descubrir problemas en el sistema que no se pudo detectar mediante pruebas funcionales automatizadas. Alguien con amplia experiencia en pruebas podría hacer preguntas importantes que no se le ocurrieron al equipo. miembros sin experiencia en pruebas. Probar el conocimiento es un componente de la capacidad de cualquier equipo para ofrecer valor.

Disfrutar

Trabajar en un equipo donde todos colaboran, donde estás involucrado el proyecto de principio a fin, donde las partes interesadas del negocio trabajan juntas con el equipo de desarrollo, donde todo el equipo asume la responsabilidad de La calidad y las pruebas, en nuestra opinión, son nada menos que la utopía de un probador. No somos los únicos que creemos que todo el mundo debería disfrutar de su trabajo. El desarrollo ágil recompensa la pasión del evaluador ágil por su trabajo.

Nuestro trabajo como probadores ágiles es particularmente satisfactorio porque nuestro punto de vista y nuestras habilidades nos permiten añadir valor real a nuestros equipos. En la siguiente sección, exploraremos cómo.

VALOR AGREGADO

¿Qué aportan estos principios al equipo? Juntos, hacen negocios valor. En el desarrollo ágil, todo el equipo asume la responsabilidad de entregar software de alta calidad que deleite a los clientes y haga que el negocio sea más rentable. más rentable. Esto, a su vez, aporta nuevas ventajas para el negocio.

Los miembros del equipo desempeñan muchas funciones y el desarrollo ágil tiende a evitar clasificar a las personas por especialidad. Incluso con iteraciones cortas y lanzamientos frecuentes, es Es fácil desarrollar una brecha entre lo que el equipo del cliente espera y lo que el el equipo cumple. El uso de pruebas para impulsar el desarrollo ayuda a prevenir esto, pero usted Todavía necesita las pruebas adecuadas.

Los evaluadores ágiles no sólo piensan en el sistema desde el punto de vista de las partes interesadas que vivirán con la solución, sino que también tienen conocimientos técnicos.

limitaciones y detalles de implementación que enfrenta el equipo de desarrollo. Los programadores se centran en hacer que las cosas funcionen. Si codifican según los requisitos correctos, los clientes estarán satisfechos. Desafortunadamente, los clientes generalmente no son buenos para articular sus necesidades. Impulsar el desarrollo con lo equivocado

Las pruebas no darán el resultado deseado. Los evaluadores ágiles hacen preguntas sobre ambos clientes y desarrolladores de manera temprana y frecuente, y ayudar a dar forma a las respuestas en el pruebas correctas.

Los evaluadores ágiles adoptan un enfoque mucho más integrado y orientado al equipo que probadores en proyectos tradicionales en cascada. Adaptan sus habilidades y experiencia al equipo y al proyecto. Un evaluador que ve a los programadores como adversarios, o se sienta y espera que le llegue el trabajo, o espera pasar más tiempo planificar que hacer, es probable que se afierre a las habilidades que aprendió en las clases tradicionales. proyectos y no durarán mucho en un equipo ágil.

Peligro: no eres "realmente" parte del equipo

Si es un tester y no está invitado a asistir a sesiones de planificación, reuniones de stand-up o reuniones de diseño, es posible que se encuentre en una situación en la que los testers sean vistos como algo aparte del equipo de desarrollo. Si lo invitan a estas reuniones pero no habla, probablemente esté creando la percepción de que en realidad no es parte del equipo. Si los expertos en negocios escriben historias y definen requisitos por sí solos, usted no participa como evaluador y es miembro de un equipo ágil.

Si esta es tu situación, tu equipo está en riesgo. Es probable que las suposiciones ocultas pasen desapercibidas hasta el final del ciclo de lanzamiento. Los efectos dominó de una historia en otras partes del sistema no se identifican hasta que es demasiado tarde. El equipo no está haciendo el mejor uso de las habilidades de cada miembro del equipo, por lo que no podrá producir el mejor software posible. La comunicación podría fallar y será difícil mantenerse al día con lo que hacen los programadores y los clientes. El equipo corre el riesgo de dividirse de forma poco saludable entre desarrolladores y evaluadores, y existe una mayor posibilidad de que el equipo de desarrollo quede aislado del equipo del cliente.

¿Cómo puedes evitar este peligro? Vea si puede hacer arreglos para ubicarse cerca de los desarrolladores. Si no puedes, al menos acércate a su zona para hablar y hacer pruebas de pareja. Pídale que le muestren en qué están trabajando. Pídale que revisen los casos de prueba que ha escrito. Invítate a reuniones si nadie más te ha invitado.

Sea útil probando y brindando comentarios, y conviértase en una necesidad para el equipo.

Ayude a los clientes a desarrollar sus historias y pruebas de aceptación. Empuje el "todo actitud de equipo" y pedirle al equipo que trabaje en los problemas de prueba. Si su equipo tiene problemas para adaptarse al desarrollo ágil, sugiera experimentar con algunas ideas nuevas durante una iteración o dos. Proponen adoptar el "Poder de Tres" regla para promover la buena comunicación. Utilice la información de este libro para demostrar que los evaluadores pueden ayudar a los equipos ágiles a tener éxito más allá de sus expectativas más descabelladas.

Durante las sesiones de planificación y estimación de historias, los evaluadores ágiles analizan cada característica desde múltiples perspectivas: negocio, usuario final, soporte de producción y programador. Consideran los problemas que enfrenta la empresa y cómo el software podría abordarlos. Plantean preguntas que desmienten las suposiciones hechas por los equipos de clientes y desarrolladores. Al comienzo de cada iteración, ayudan a garantizar que el cliente proporcione requisitos claros y ejemplos y ayudan al equipo de desarrollo a convertirlos en pruebas. El Las pruebas impulsan el desarrollo y los resultados de las pruebas proporcionan retroalimentación sobre el desempeño del equipo. progreso. Los evaluadores ayudan a plantear problemas para que no se pase por alto ninguna prueba; es más que las pruebas funcionales. Los clientes no siempre saben que deben mencionar sus necesidades de rendimiento y confiabilidad o sus preocupaciones de seguridad, pero los evaluadores Piensa en preguntar sobre esos. Los evaluadores también mantienen el enfoque y las herramientas de prueba como lo más simple y liviano posible. Al final de la iteración, los evaluadores verifican que se completaron las pruebas mínimas.

Las líneas entre los roles en un equipo ágil son borrosas. Otros miembros del equipo podrían tener habilidades en las mismas actividades que realizan los evaluadores. Por ejemplo, los analistas y programadores también escriben pruebas orientadas al negocio. Mientras todas las pruebas Se realizan actividades, un equipo ágil no necesariamente requiere miembros que se identifican principalmente como evaluadores. Sin embargo, hemos encontrado que Los equipos se benefician de las habilidades que los evaluadores profesionales han desarrollado. Los principios y valores ágiles que hemos discutido ayudarán a cualquier equipo a hacer un buen trabajo. probando y entregando valor.

RESUMEN

En este capítulo, cubrimos los principios para los evaluadores ágiles y los valores que creemos que son El evaluador ágil debe poseer para contribuir de manera efectiva a un equipo ágil.

Una "mentalidad de prueba ágil" está centrada en el cliente, orientada a los resultados, artesanal, colaborativa, creativa, con ganas de aprender y apasionada por ofrecer valor empresarial de manera oportuna.

La actitud es importante y desdibuja las líneas entre evaluadores, programadores y otros roles en un equipo ágil.

Los evaluadores ágiles aplican valores y principios ágiles como retroalimentación, comunicación, coraje, simplicidad, disfrute y entrega de valor para ayudar al equipo a identificar y cumplir los requisitos del cliente para cada historia.

Los evaluadores ágiles agregan valor a sus equipos y organizaciones con su punto de vista único y su enfoque orientado al equipo.

Parte II

DESAFÍOS ORGANIZACIONALES

Cuando las organizaciones de desarrollo de software implementan un desarrollo ágil, el equipo de pruebas o de control de calidad suele ser el que tarda más en realizar la transición. Los equipos de control de calidad independientes se han arraigado en muchas organizaciones. Cuando comienzan a adaptarse a una nueva organización ágil, se topan con diferencias culturales que les resultan difíciles de aceptar. En la Parte II, hablamos sobre la introducción del cambio y algunas de las barreras que puede encontrar durante la transición.

a ágil. La capacitación es una gran parte de lo que las organizaciones que hacen la transición necesita y a menudo se olvida. También es difícil ver cómo procesos existentes como ya que las auditorías y los marcos de mejora de procesos funcionarán en un entorno ágil. Pasar de un equipo de control de calidad independiente a un equipo ágil integrado es un gran cambio.

El capítulo 4, “Logística del equipo”, habla sobre la estructura del equipo, como dónde el tester realmente encaja en el equipo, y la interminable pregunta sobre la proporción tester-desarrollador. También hablaremos sobre la contratación de evaluadores y qué buscar en un probador ágil exitoso.

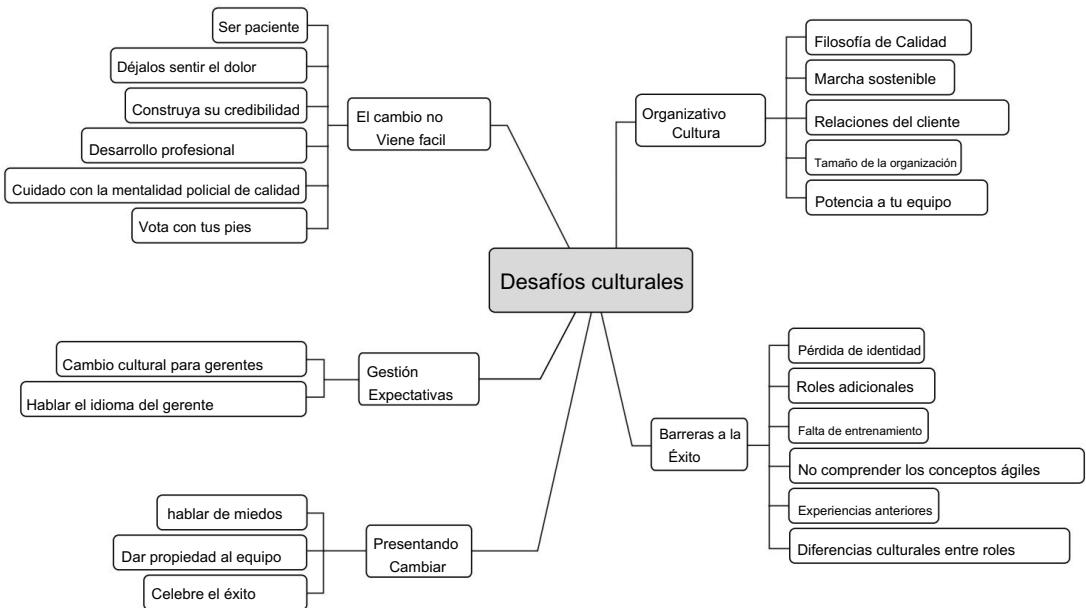
Actividades de prueba tradicionales, como registrar errores, realizar un seguimiento de las métricas, y escribir planes de prueba, puede no parecer una buena opción en un proyecto ágil. Nosotros presentar algunos de los procesos típicos que podrían necesitar cuidado y atención especiales y discutir cómo adaptar los procesos de calidad existentes.

Puede esperar encontrar formas en que los evaluadores y los equipos de prueba acostumbrados a un entorno de desarrollo tradicional en cascada puedan cambiar su estructura y cultura organizacional para beneficiarse y agregar valor a la metodología ágil.

Esta página se dejó en blanco intencionalmente.

Capítulo 3

DESAFÍOS CULTURALES



Muchas influencias organizacionales pueden afectar un proyecto, ya sea que utilice un enfoque ágil o tradicional por fases o por etapas. La cultura organizacional y de equipo puede bloquear una transición fluida hacia un enfoque ágil. En este capítulo, analizamos los factores que pueden afectar directamente el papel de un evaluador en un equipo ágil.

CULTURA ORGANIZACIONAL

Una cultura organizacional se define por sus valores, normas y supuestos.

La cultura de una organización gobierna cómo las personas se comunican, se interrelacionan y toman decisiones, y se ve fácilmente al observar el comportamiento de los empleados.

La cultura de una organización puede afectar el éxito de un equipo ágil. Ágil equipos son más adecuados para organizaciones que permiten el pensamiento independiente. Para Por ejemplo, si una empresa tiene una estructura jerárquica y fomenta una dirección estilo de gestión para todos sus proyectos, los equipos ágiles probablemente tendrán dificultades. Pasado Las experiencias de la organización también afectarán el éxito de una nueva estrategia ágil. equipo. Si una empresa intentó ser ágil y obtuvo malos resultados, la gente sospechará de intentarlo de nuevo, citando ejemplos de por qué no funcionó. Incluso podrían hacer campaña activa en su contra.

Con demasiada frecuencia no se tiene en cuenta la cultura organizacional cuando se intentan hecho para implementar un proceso ágil, dejando a la gente preguntándose por qué no fue así Funciona según lo prometido. Es difícil cambiar los procesos establecidos, especialmente si los individuos sienten que tienen un interés en el status quo. Cada grupo funcional desarrolla una subcultura y procesos que satisfacen sus necesidades. Son comodos con su forma de trabajar. El miedo es una emoción poderosa y si no se aborda, puede poner en peligro la transición a la metodología ágil. Si los miembros del equipo sienten que un nuevo ágil El proceso amenaza sus puestos de trabajo, se resistirán al cambio.

Hablaremos específicamente sobre cómo la cultura organizacional afecta a los evaluadores que trabajan en un entorno ágil. La bibliografía contiene recursos que tratan con otros aspectos culturales que pueden afectar a los equipos.

Filosofía de Calidad

Considere la filosofía de calidad de una organización en términos de cómo determina el nivel aceptable de calidad del software. ¿Tolerar la mala calidad? Lo hace tener en cuenta los requisitos de calidad de los clientes, o simplemente se refiere ¿Con poner el producto en manos de los clientes lo más rápido posible?

Cuando una organización carece de una filosofía y presiones generales de calidad. Cuando los equipos sacan el producto sin tener en cuenta la calidad, los evaluadores sienten la presión. Un equipo que intenta utilizar el desarrollo ágil en un entorno de este tipo se enfrenta a un batalla cuesta arriba.

Algunas organizaciones tienen equipos de prueba sólidos e independientes que ejercen una gran cantidad de fuerza. Estos equipos, y sus gerentes, podrían percibir que el desarrollo ágil les quitará ese poder. Podrían temer que la agilidad vaya en contra de su filosofía de calidad. Evalúe la filosofía de calidad de su organización y la filosofía de los equipos que la aplican.

Peligro: mentalidad policial de calidad

Si un equipo de control de calidad existente ha asumido el papel de "Policía de calidad", sus miembros generalmente hacen cumplir la calidad asegurándose de que se completen las revisiones de código y de que los errores se ingresen religiosamente en los sistemas de seguimiento de defectos. Mantienen métricas sobre la cantidad de errores encontrados y luego se encargan de realizar la evaluación final. decisión sobre si lanzar o no el producto.

Hemos hablado con evaluadores que se jactan de logros como pasar por alto a un gerente de desarrollo para obligar a un programador a seguir estándares de codificación. Incluso hemos oido hablar de evaluadores que dedican su tiempo a escribir errores sobre requisitos que no cumplen con sus estándares. Este tipo de actitud no funcionará. en un equipo ágil y colaborativo. Fomenta un comportamiento antagónico.

Otro riesgo del rol de "Policía de Calidad" es que el equipo no acepta el concepto de construir calidad y los programadores comienzan a utilizar los evaluadores como red de seguridad. El equipo comienza a comunicarse a través del sistema de seguimiento de errores, que no es un medio muy eficaz de comunicación, por lo que el equipo nunca "se solidifica".

Siga leyendo para conocer formas de ayudar a evitar este peligro.

A las empresas en las que todos valoran la calidad les resultará más fácil hacer la transición a la metodología ágil. Si algún grupo ha asumido la propiedad de la calidad, Tienes que aprender a compartir eso con todos los demás miembros del equipo para tener éxito.

Propiedad de la calidad por parte de todo el equipo

En el Capítulo 1, "¿Qué son las pruebas ágiles, de todos modos?", hablamos sobre el enfoque de calidad de todo el equipo. Para muchos evaluadores y equipos de control de calidad, esto significa una mente pasar de ser dueños de la calidad a tener un papel participativo en la definición y manteniendo la calidad. Un cambio de actitud tan drástico es difícil para muchos evaluadores y equipos de control de calidad.

Los evaluadores que han estado trabajando en un entorno tradicional podrían tener dificultades tiempo para adaptarse a sus nuevos roles y actividades. Si provienen de una organización donde el desarrollo y el control de calidad tienen una relación de confrontación, puede ser Será difícil pasar de ser una idea de último momento (si es que se piensa en ello) a ser una parte integral del equipo. Puede resultar difícil tanto para los programadores como para probadores para aprender a confiar unos en otros.

Habilidades y adaptabilidad

Se ha observado mucho acerca de los programadores que no pueden adaptarse a las prácticas ágiles, pero ¿qué pasa con los evaluadores que están acostumbrados a crear scripts de prueba de acuerdo con sus necesidades? a un documento de requisitos? ¿Pueden aprender a hacer las preguntas como el código?

¿está siendo construido? Los evaluadores que no cambian su enfoque de las pruebas tienen dificultades tiempo trabajando estrechamente con el resto del equipo de desarrollo.

Probadores que están acostumbrados a realizar únicamente pruebas manuales a través de la interfaz de usuario. Es posible que no entiendan el enfoque automatizado que es intrínseco al desarrollo ágil. Estos evaluadores necesitan mucho coraje para afrontar sus cambios. roles, porque cambiar significa desarrollar nuevas habilidades fuera de sus zonas de confort.

Factores que ayudan

Aunque hay muchas cuestiones culturales a considerar, la mayoría de los equipos de control de calidad tienen una se centran en la mejora de procesos, y los proyectos ágiles fomentan mejoras continuas y adaptabilidad mediante el uso de herramientas como las retrospectivas.

La mayoría de los profesionales de control de calidad están deseosos de aplicar lo que han aprendido. y hacerlo mejor. Estas personas son lo suficientemente adaptables no sólo para sobrevivir, sino prosperar en un proyecto ágil.

Si su organización se centra en el aprendizaje, fomentará el proceso continuo. mejora. Probablemente adoptará la metodología ágil mucho más rápidamente que las organizaciones que valoran más cómo reaccionan ante las crisis que mejorar su desempeño. procesos.

Si usted es un evaluador en una organización que no tiene una filosofía de calidad efectiva, probablemente tenga dificultades para que se acepten prácticas de calidad. El enfoque ágil proporcionarle un mecanismo para introducir buenas prácticas orientadas a la calidad.

Los evaluadores necesitan tiempo y formación, como todos los demás que están aprendiendo a trabajar. en un proyecto ágil. Si dirige un equipo que incluye evaluadores, asegúrese de darles mucho apoyo. Los evaluadores a menudo no llegan al principio de un proyecto totalmente nuevo y luego se espera que encajen en un equipo que tiene estado trabajando juntos durante meses. Para ayudar a los evaluadores a adaptarse, es posible que necesite Traiga a un entrenador de pruebas ágiles con experiencia. Contratar a alguien que haya trabajado previamente en un equipo ágil y que pueda servir como mentor y maestro será ayudar a los evaluadores a integrarse con la nueva cultura ágil, ya sea que estén haciendo la transición a la metodología ágil junto con un equipo existente o uniéndose a un nuevo desarrollo ágil equipo.

Marcha sostenible

Los equipos de prueba tradicionales están acostumbrados a pruebas rápidas y furiosas al final de un proyecto, que se traduce en trabajar los fines de semana y las tardes. Durante esto fase de prueba de final del proyecto, algunas organizaciones piden periódicamente a sus equipos que

Dedique 50, 60 o más horas cada semana para tratar de cumplir con una fecha límite. Las organizaciones a menudo consideran las horas extras como una medida del compromiso de un individuo.

Esto entra en conflicto con los valores ágiles que giran en torno a permitir que las personas hagan su mejor trabajo en todo momento.

En proyectos ágiles, se le anima a trabajar a un ritmo sostenible. Esto significa que los equipos trabajan a un ritmo constante que mantiene una velocidad constante que permite mantener un alto estándar de calidad. Los nuevos equipos ágiles tienden a ser demasiado optimistas sobre lo que pueden lograr y a aceptar demasiado trabajo. Después de una iteración o dos, aprenden a inscribirse en el trabajo suficiente para que no necesiten horas extras para completar sus tareas. Una semana de 40 horas es el ritmo normal sostenible para los equipos XP; es la cantidad de esfuerzo que, si se realiza semana tras semana, permite a las personas realizar la mayor cantidad de trabajo a largo plazo y, al mismo tiempo, ofrecer un buen valor.

Es posible que los equipos necesiten trabajar durante períodos cortos de ritmo insostenible de vez en cuando, pero debería ser la excepción, no la norma. Si se requieren horas extras por períodos cortos, todo el equipo debería trabajar horas extra. Si es el último día del sprint y algunas historias no se prueban, todo el equipo debería quedarse hasta tarde para terminar las pruebas, no solo los evaluadores. Utilice las prácticas y técnicas recomendadas a lo largo de este libro para aprender cómo planificar las pruebas junto con el desarrollo y permitir que las pruebas "se mantengan al día" con la codificación. Hasta que su equipo mejore en la gestión de su carga de trabajo y velocidad, reserve tiempo adicional para ayudar a igualar el ritmo.

Relaciones con los clientes En el

desarrollo de software tradicional, la relación entre los equipos de desarrollo y sus clientes se parece más a una relación vendedor-proveedor.

Incluso si el cliente es interno, puede parecer más dos empresas separadas que dos equipos trabajando con el objetivo común de producir valor empresarial.

El desarrollo ágil depende de la estrecha participación de los clientes o, al menos, de sus representantes. Los equipos ágiles han invitado a los clientes a colaborar, trabajar en las mismas ubicaciones si es posible y participar íntimamente en el proceso de desarrollo. Ambas partes aprenden las fortalezas y debilidades de cada una.

Este cambio en las relaciones debe ser reconocido por ambas partes, y no importa si el cliente es interno o externo. Una relación abierta es fundamental para el éxito de un proyecto ágil, donde la relación entre el equipo del cliente y el equipo de desarrollo se parece más a una asociación que a una relación proveedor-proveedor.

La historia de Janet

En un gran proyecto en el que estuve recientemente, el cliente era en realidad un consorcio de cinco empresas, siendo una de ellas la empresa de software que crea el software.

Cada una de las empresas proporcionó tres de sus mejores expertos en el campo para representar sus necesidades. Existía una comunicación regular entre estos usuarios del sitio y sus propias organizaciones, y también eran una parte integral del equipo con el que trabajaban a diario.

Un comité directivo con representantes de las cinco empresas se mantuvo informado sobre los avances y se convocó cuando era necesario tomar decisiones a un nivel superior.

—Janet

Tener algunos expertos en el dominio representativos, manteniendo al mismo tiempo a todas las partes interesadas estar continuamente informado es un enfoque para lograr una colaboración exitosa entre desarrollador y cliente. Hablaremos de otros en la Parte V. Los clientes son fundamentales para el éxito de su proyecto ágil. Priorizan lo que se construirá y tienen la última palabra en la calidad del producto. Los evaluadores trabajan estrechamente con los clientes para conocer los requisitos y definir pruebas de aceptación que demuestren que se cumplen las condiciones de satisfacción. Las actividades de prueba son clave para el desarrollo.

Relación equipo-equipo cliente. Por eso la experiencia en pruebas es tan esencial a equipos ágiles.

Tamaño de la organización

El tamaño de una organización puede tener un gran impacto en la forma en que se ejecutan los proyectos. y cómo madura la estructura de una empresa. Cuanto más grande sea la organización, cuanto más jerárquica tiende a ser la estructura. Como comunicación de arriba hacia abajo Se desarrollan canales, las estructuras de informes se vuelven directivas y menos compatible con la colaboración entre tecnología y empresas.

Desafíos de comunicación

Algunos procesos ágiles proporcionan formas de facilitar la comunicación entre equipos. Por ejemplo, Scrum tiene el “Scrum de Scrums”, donde representantes de Varios equipos se coordinan diariamente.

Si trabaja en una organización grande donde los equipos de prueba u otros especialistas Los recursos están separados de los equipos de programación, trabajan para encontrar formas de manténgase en contacto constante. Por ejemplo, si su equipo de base de datos está completamente separado, necesita encontrar una manera de trabajar estrechamente con los especialistas en bases de datos en para obtener lo que necesita en el momento oportuno.

Capítulo 16, "Golpe la carrera terrestre-ning", describe cómo una gran organización utiliza analistas funcionales para mitigar los problemas. problemas debidos a re-motear a los clientes.

Otro problema que suele ser más común en las grandes empresas es que Es posible que los clientes no sean tan accesibles como lo son en las empresas más pequeñas. Esto es un gran obstáculo cuando se intenta reunir requisitos y ejemplos y se busca Conseguir la participación del cliente durante todo el ciclo de desarrollo. Una solución es tener evaluadores o analistas con experiencia en el campo que actúen como representantes del cliente. Las herramientas de comunicación también pueden ayudar a afrontar este tipo de situaciones. Buscar formas creativas de superar los problemas inherentes a las grandes empresas.

Culturas en conflicto dentro de la organización

En el caso de los grandes talleres de desarrollo de software, el desarrollo ágil a menudo se implementa primero en un equipo o en unos pocos equipos. Si su equipo ágil tiene que coordinarse con otros equipos utilizando otros enfoques, como por fases o cerrados desarrollo, tienes un conjunto adicional de desafíos. Si algunos de los externos Los equipos tienden a ser disfuncionales, es aún más difícil. Incluso cuando toda una empresa Adopta la metodología ágil, algunos equipos realizan la transición con más éxito que otros.

Su equipo también podría encontrar resistencia por parte de equipos de especialistas que se sienten protectores de sus silos particulares. Lisa habló con un equipo cuyos miembros no pudo obtener ninguna ayuda de la gestión de configuración de su empresa equipo, lo que obviamente fue un gran obstáculo. Algunos equipos de desarrollo están prohibido hablar directamente con los clientes.

Si hay terceros trabajando en el mismo sistema en el que trabaja su equipo, sus culturas también pueden causar conflictos. Quizás tu equipo sea el tercero, y estás desarrollando software para un cliente. Tendrás que pensar en cómo para mitigar las diferencias culturales. La Parte V entra en más detalles sobre trabajando con otros equipos y terceros, pero aquí hay algunas ideas para obtener tú empezaste.

Capítulo 15, "Actividades del probador en la planificación de lanzamientos o temas" y Capítulo 16, "Herramientas de la Ground Run-ning", habla sobre lo que los evaluadores pueden hacer para ayudar con la planificación y coordinación con otros equipos.

Planificación avanzada Si tiene que coordinar con otros equipos, necesitará dedicar tiempo durante la planificación del lanzamiento, o antes del inicio de una iteración, para trabajar con ellos. Necesita tiempo para adaptar sus propios procesos para que funcionen con los procesos de otros, y es posible que ellos necesiten cambiar sus procesos para adaptarse a ellos. Considere organizar el acceso a recursos compartidos, como especialistas en pruebas de rendimiento o entornos de pruebas de carga, y planifique su propio trabajo. en torno a los horarios de los demás. Sus partes interesadas podrían esperar ciertos resultados, como planes de prueba formales, que su propio proceso ágil no incluye. Un poco de planificación adicional le ayudará a superar estas diferencias culturales.

Actúe ahora, discúlpese más tarde. Dudamos en hacer sugerencias que puedan causar problemas, pero a menudo en una organización grande, las ruedas burocráticas giran tan

lentamente que su equipo podría tener que descubrir e implementar sus propias soluciones. Por ejemplo, el equipo que no pudo obtener la cooperación del equipo de gestión de configuración simplemente implementó su propio proceso de construcción interno y Continuó trabajando para integrarlo al proceso sancionado oficialmente.

Si no existen canales oficiales para conseguir lo que necesitas, es hora de ser creativo. Quizás los evaluadores nunca antes hayan hablado directamente con los clientes. Intentar Organice una reunión usted mismo o busque a alguien que pueda actuar como representante del cliente. o intermediario.

Potencia a tu equipo

El capítulo 4, "Logística de equipos", habla más sobre equipos funcionales separados y cómo afectan al evaluador ágil.

En un proyecto ágil, es importante que cada equipo de desarrollo se sienta capacitado para tomar decisiones. Si eres gerente y quieres que tus equipos ágiles tener éxito, déjelos libres para actuar y reaccionar creativamente. La cultura de una organización debe adaptarse a este cambio para que un proyecto ágil tenga éxito.

BARRERAS PARA UNA ADOPCIÓN ÁGIL EXITOSA

POR EQUIPOS DE PRUEBA/QA

Cualquier cambio enfrenta barreras para lograr el éxito. Cultura organizacional, como comentamos en la sección anterior, podría ser el mayor obstáculo a superar. Una vez que la cultura organizacional está bien establecida, es muy difícil cambiarla. Él Tomó tiempo para que se formara y, una vez implementado, los empleados se comprometen con la cultura, lo que la hace extremadamente resistente a la alteración.

Esta sección analiza las barreras específicas para la adopción del desarrollo ágil. métodos que pueden encontrar sus evaluadores y equipos de control de calidad.

Pérdida de identidad

Los evaluadores se aferran al concepto de un equipo de control de calidad independiente por muchas razones, pero el motivo principal es el miedo, concretamente:

Miedo a perder su identidad de control de calidad.

Miedo a que si reportan a un gerente de desarrollo, perderán apoyo y los programadores tendrán prioridad.

Miedo a carecer de las habilidades para trabajar en un equipo ágil y perder sus puestos de trabajo.

Miedo a que cuando estén dispersos en equipos de desarrollo no obtengan el apoyo que necesitan.

Miedo a que ellos y sus gerentes se pierdan en la nueva organización.

Capítulo 4, "Equipo Logística", cubre ideas que pueden ser

Se utiliza para ayudar a las personas a adaptarse.

A menudo escuchamos a los gerentes de control de calidad hacer preguntas como: "Mi empresa está implementando un desarrollo ágil. ¿Cómo encaja mi papel? Esto está directamente relacionado con los temores de "pérdida de identidad".

Roles adicionales

Sabemos por experiencia que a los nuevos equipos a menudo les faltan especialistas o experiencia que podrían ser clave para su éxito. El equipo de Lisa se ha topado con obstáculos.

tan grande que lo único que podía hacer era sentarse y preguntar: "¿Qué papel desempeñamos?"

¿Falta en nuestro equipo lo que nos está frenando? ¿Qué necesitamos? ¿Otro desarrollador, otro evaluador, un diseñador de bases de datos? Todos sabemos que las pruebas son una vasto campo. Quizás necesite a alguien con experiencia en pruebas en un equipo ágil.

O tal vez necesite un especialista en pruebas de rendimiento. Es fundamental que tomes

Es el momento de analizar qué roles necesita su producto para tener éxito y si

Necesitas llenarlos desde fuera del equipo, hazlo.

Es fundamental que todos los que ya forman parte del equipo de producto comprendan su función.

o descubrir cuál es su función ahora que son parte de un nuevo equipo ágil. Hacer esto requiere tiempo y entrenamiento.

Falta de entrenamiento

Organizamos una sesión en la "Conferencia dentro de una Conferencia" en Agile 2007 que preguntó a las personas qué problemas relacionados con las pruebas tenían en su equipos ágiles. Uno de los asistentes nos dijo que dividieron su organización de pruebas como lo recomienda la literatura ágil. Sin embargo, pusieron a los probadores en unidades de desarrollo sin ningún tipo de capacitación; en tres meses, todos los Los evaluadores habían renunciado porque no entendían sus nuevos roles. Problemas como Estos se pueden prevenir con la formación y el asesoramiento adecuados.

Cuando comenzamos a trabajar con nuestros primeros equipos ágiles, no había muchos recursos disponibles para ayudarnos a aprender qué deberían hacer los evaluadores ágiles o cómo deberíamos trabajar junto con nuestros equipos. Hoy en día, puede encontrar muchos profesionales que pueden ayudar a capacitar a los evaluadores para que se adapten a un entorno ágil y ayuden a realizar pruebas. los equipos hacen la transición ágil. Grupos de usuarios locales, conferencias, seminarios, La instrucción en línea y las listas de correo brindan recursos valiosos a los evaluadores. y directivos con ganas de aprender. No tengas miedo de buscar ayuda cuando la necesites. él. Un buen entrenamiento proporciona un buen retorno de su inversión.

No comprender los conceptos ágiles

No todos los equipos ágiles son iguales. Hay muchos enfoques diferentes para la agilidad. desarrollo, como XP, Scrum, Crystal, FDD, DSDM, OpenUP y varios

mezclas de esos. Algunos equipos autodenominados "ágiles" no son, en nuestra opinión, realmente practicando agilidad. Muchos equipos simplemente adoptan prácticas que les funcionan independientemente de la fuente original, o inventan la suya propia. Está bien, pero si Si no siguen ninguno de los valores y principios ágiles fundamentales, cuestionamos darles una etiqueta ágil. Publicar cada mes y prescindir de la documentación no equivale a un desarrollo ágil.

Si diferentes miembros del equipo tienen nociones opuestas sobre lo que constituye "ágil", qué prácticas deberían utilizar, o cómo se supone que deben ser esas prácticas. practicado, habrá problemas. Por ejemplo, si eres un evaluador que Si presionas al equipo para que implemente la integración continua, pero los programadores simplemente se niegan a intentarlo, estás en una mala situación. Si eres programador Quien no logra involucrarse en algunas prácticas, como impulsar el desarrollo con pruebas orientadas al negocio, también se verá envuelto en un conflicto.

El equipo debe llegar a un consenso sobre cómo proceder para realizar una transición exitosa a la metodología ágil. Muchas de las prácticas de desarrollo ágil son sinérgicas, por lo que si se utilizan de forma aislada, es posible que no proporcionen los beneficios que ofrecen. Los equipos están buscando. Quizás el equipo pueda aceptar experimentar con ciertos prácticas para un número determinado de iteraciones y evaluar los resultados. Podría deciden buscar aportaciones externas que les ayuden a comprender las prácticas y cómo encajan. Diversos puntos de vista son buenos para un equipo, pero todos necesitan ir en la misma dirección.

Varias personas con las que hemos hablado describieron el fenómeno de la "minicascada" Esto ocurre a menudo cuando una organización tradicional de desarrollo de software implementa un proceso de desarrollo ágil. La organización reemplaza un ciclo de desarrollo de seis meses o un año por uno de dos o cuatro semanas, y solo intenta exprimir todas las fases tradicionales del SDLC en ese corto período. Naturalmente, siguen teniendo los mismos problemas que antes. Figura 3-1 muestra una versión "ideal" de la minicascada donde hay un código y corrección fase y luego prueba: la prueba se realiza después de que se completa la codificación, pero antes de que comience la siguiente iteración. Sin embargo, lo que realmente sucede es que las pruebas se vuelven se aprieta al final de la iteración y generalmente se arrastra hasta la siguiente iteración. Los programadores no tienen mucho que arreglar todavía, así que empiezan a trabajar en la siguiente iteración. En poco tiempo, algunos equipos siempre están una iteración "detrás" con sus pruebas y las fechas de lanzamiento se posponen como siempre.

Todos los involucrados en la entrega del producto necesitan tiempo y capacitación para comprender los conceptos detrás de la metodología ágil y las prácticas básicas. Se pueden utilizar entrenadores experimentados para brindar capacitación práctica en prácticas nuevas para el equipo, como el desarrollo basado en pruebas. En organizaciones más grandes, funcional

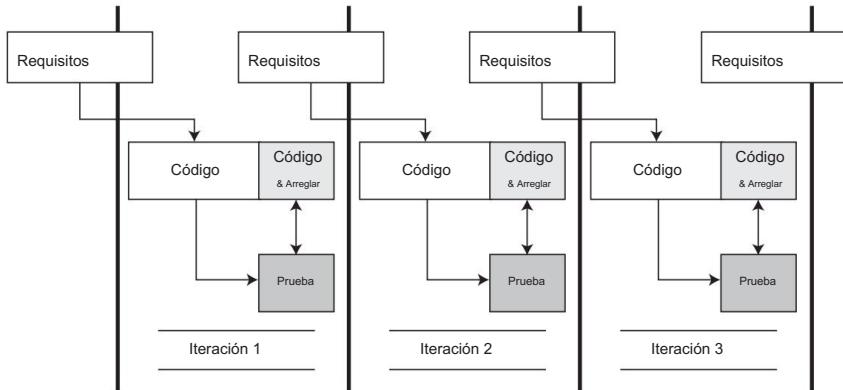


Figura 3-1 Un proceso en minicascada

Los gerentes de pruebas pueden convertirse en líderes de práctica y brindar apoyo y recursos para que los evaluadores aprendan a comunicarse y colaborar con sus nuevos equipos. Los programadores y otros miembros del equipo necesitan ayuda similar de sus responsables funcionales. Un liderazgo fuerte ayudará a los equipos a encontrar formas de migrar de la “minicascada” al verdadero desarrollo colaborativo, donde La codificación y las pruebas se integran en un solo proceso.



Consulte la bibliografía para obtener un enlace a más información sobre las cartas de radar XP.

XP ha desarrollado un gráfico de radar para ayudar a los equipos a determinar su nivel de adaptación a las prácticas clave de XP. Miden cinco prácticas clave diferentes: equipo, programación, planificación, cliente y emparejamiento, y muestran el nivel de adaptación a prácticas por equipos. La Figura 3-2 muestra dos de estos gráficos. El gráfico

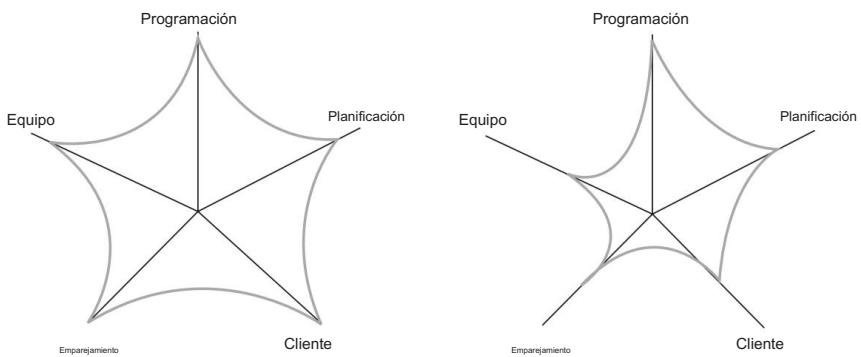


Figura 3-2 Cartas de radar XP

a la izquierda muestra una adaptación exitosa, mientras que el gráfico de la derecha muestra que hay algunas áreas problemáticas.

Experiencia/actitud pasada

Mucha gente ha pasado por cambios que no se mantuvieron. Algunas organizaciones de desarrollo han vivido una sucesión de la "metodología del día." Levantan las manos y se preguntan: "¿Por qué deberíamos hacerlo de nuevo?" La gente se queda estancada en sus viejos patrones fallidos. Incluso cuando lo intentan algo nuevo, podrían volver a malos hábitos cuando se encuentran bajo estrés. Los siguientes son sólo algunos ejemplos de personas que se resisten al cambio debido a experiencias pasadas y su percepción de "cómo son las cosas":

Un evaluador se sentó en su cubo y no quiso hablar con los programadores sobre los problemas que tenía. Se quejó de que los programadores no entendían lo que quería.

Un evaluador no podía deshacerse de su actitud actual de que los programadores no sabían cómo escribir buen código o cómo probarlo. Su actitud condescendiente fue clara para todos y su credibilidad como evaluador fue cuestionada.

Un cliente se dio por vencido cuando los programadores hicieron algo que no le gustó, porque de todos modos "siempre" hacen lo que quieren.

Cuando se enfrenta a una transición hacia el desarrollo ágil, personas como esta a menudo abandonan sin darle oportunidad al nuevo proceso. El desarrollo ágil no es para todos, pero la capacitación y el tiempo para experimentar pueden ayudar a ajustar las actitudes. Pide a todos que sean parte de la solución y trabajen juntos para descubrir qué procesos y prácticas funcionan mejor para sus situaciones particulares. El equipo autoorganizado puede ser una poderosa herramienta para asegurar a todos los miembros del equipo de desarrollo que tienen el control de su propio destino.

Diferencias culturales entre roles

Cada nuevo miembro del equipo ágil está haciendo la transición desde una perspectiva diferente. Los programadores suelen estar acostumbrados a escribir código de producción y obtener se lanzó lo más rápido posible. Administradores de sistemas y expertos en bases de datos. podría estar acostumbrado a trabajar en su propio silo, realizando solicitudes en su propio horario. Es posible que los clientes nunca hayan hablado directamente con los miembros del equipo de desarrollo. Los evaluadores podrían estar acostumbrados a llegar al final del proyecto y no interactuar mucho con los programadores.

No es de extrañar que una transición a la metodología ágil pueda resultar aterradora. Los equipos pueden idear reglas y pautas para ayudarlos a comunicarse y trabajar bien juntos. Para

Por ejemplo, Lisa se unió a un nuevo equipo ágil cuya regla era que si alguien preguntaba Para emparejarte con ella, tenías que estar de acuerdo. Quizás no puedes hacerlo bien En ese momento, pero tan pronto como pudiste liberarte, tuviste que ir a ayudar. tu compañero de equipo.

Identificar lo que necesitan las personas que realizan diferentes actividades y encontrar formas de proporcionarlo. él. Los clientes necesitan alguna forma de saber cómo avanza el desarrollo y si se cumplen sus condiciones de satisfacción. Los desarrolladores necesitan conocer las prioridades y requisitos del negocio. Los evaluadores necesitan formas de capturar ejemplos y convertirlos en pruebas. Todos los miembros del equipo quieren sentirse valorados y miembros del equipo de primera clase. Cada miembro del equipo también necesita sentirse seguro y sentirse libre de plantear cuestiones y probar nuevas ideas. Comprender el punto de vista de cada rol ayuda a los equipos durante la transición.

INTRODUCIENDO EL CAMBIO

Al implementar cualquier cambio, tenga en cuenta los efectos secundarios. El primer escenario puede ser caos; su equipo no está seguro de cuáles son los nuevos procesos, algunos grupos Son leales a las viejas costumbres y algunas personas son inseguras y perturbadoras. La gente confunde esta etapa caótica con el nuevo status quo. Para evitar esto, explique cambiar el modelo desde el principio y establecer expectativas. Esperar y aceptar lo percibido. caos al implementar procesos ágiles. Encuentre las áreas de mayor dolor y determine qué prácticas resolverán el problema para que pueda obtener algún progreso inmediato fuera del caos.

hablar de miedos

Cuando comience el desarrollo iterativo, utilice retrospectivas para proporcionar a las personas con un lugar para hablar de sus miedos y un lugar en el que puedan dar su opinión. Hágale saber a la gente que es normal tener miedo. Estar abierto; enséñales que es Es aceptable decir que tienen miedo o se sienten incómodos. Discuta cada fuente de miedo, aprender de la discusión, tomar decisiones y seguir adelante. El miedo es una respuesta común al cambio. Obligar a las personas a hacer algo que no quieren es perjudicial para el cambio positivo. Predicar con el ejemplo.

La historia de Lisa

Janet y yo nos unimos a nuestros primeros equipos de XP en un momento en el que muchos profesionales de XP no veían ningún lugar para los evaluadores en un equipo de XP. XP tenía una "Declaración de derechos del cliente" y una "Declaración de Derechos del Programador", pero la "Declaración de Derechos del Probador" estaba notoriamente ausente. Tip House y yo elaboramos nuestra propia "Declaración de derechos de los probadores" para brindarles a los probadores el apoyo y el coraje para tener éxito en equipos ágiles. A lo largo de los años, muchos evaluadores nos han dicho cuánto les ayudó esto a ellos y a sus equipos a aprender cómo trabajan los evaluadores junto con otros miembros del equipo. No me gustan demasiadas reglas, pero pueden

Será algo bueno cuando ayuden al equipo a superar las barreras culturales y a comprender cómo trabajar de nuevas maneras. La siguiente lista presenta una "Declaración de derechos del evaluador". Le recomendamos que lo utilice para ayudar a los evaluadores a integrarse en equipos ágiles.

- Tiene derecho a plantear cuestiones relacionadas con las pruebas, la calidad y el proceso en en cualquier momento.
- Tiene derecho a hacer preguntas a clientes, programadores y otros miembros del equipo y recibir respuestas oportunas.
- Tienes derecho a solicitar y recibir ayuda de cualquier persona en el proyecto. equipos, incluidos programadores, gerentes y clientes.
- Tiene derecho a estimar las tareas de prueba y a incluirlas en la historia. estimados.
- Tiene derecho a las herramientas que necesita para realizar tareas de prueba en el momento oportuno. manera.
- Tiene derecho a esperar que todo su equipo, no sólo usted mismo, sea responsable. ble para calidad y pruebas.

—Lisa

Dar propiedad al equipo

Un factor crítico de éxito es si el equipo se responsabiliza y tiene la capacidad de personalizar su enfoque. Las personas pueden cambiar sus actitudes y sus

percepciones si se les brinda la ayuda adecuada. Lisa pudo observar a Mike.

Cohn trabaja con su equipo como entrenadora. Como equipo autoorganizado, el equipo tenía identificar y resolver sus propios problemas. Mike se aseguró de que tuvieran tiempo.

y recursos para experimentar y mejorar. Se aseguró de que el negocio

Entendí que la calidad era más importante que la cantidad o la velocidad. Cada

Un equipo, incluso un equipo autoorganizado, necesita un líder que pueda interactuar eficazmente. con el equipo directivo de la organización.

Celebre el éxito

Implementar cambios lleva tiempo y puede resultar frustrante, así que asegúrese de celebrar todos los éxitos que logre su equipo. Date unas palmaditas en la espalda cuando cumplir con su objetivo de escribir casos de prueba de alto nivel para todas las historias antes del cuarto día de la iteración. Reúne al equipo para un juego de preguntas o un almuerzo cuando hayas acaba de entregar el trabajo de una iteración. El reconocimiento es importante si Quieres que el cambio se mantenga.

Capítulo 18,
"Codificación y pruebas",
Cubre cómo los
evaluadores y
programadores
trabajan juntos
durante todo el desarrollo.
proceso.

Integrar a los evaluadores en los equipos de desarrollo y al mismo tiempo permitirles continuar informando a un gerente de control de calidad que los apoye es una forma de facilitar la transición al desarrollo ágil. Los evaluadores pueden encontrar formas de superar una relación de confrontación con programadores a uno colaborativo. Pueden mostrar cómo pueden ayudar.

Superar la resistencia a lo ágil

Mark Benander, líder del equipo de control de calidad de Quickoffice, estaba en su cuarto proyecto en un equipo ágil. La primera fue una reescritura importante de toda su aplicación, con un equipo de ocho desarrolladores, un evaluador y ninguna herramienta de automatización de pruebas. Nos contó sus experiencias al superar sus preocupaciones sobre el desarrollo ágil, especialmente acerca de informar a un gerente de desarrollo.

Estábamos en un tipo de sistema de gestión matricial, donde un evaluador informa a un gerente de desarrollo, pero el gerente de pruebas sigue siendo oficialmente el supervisor. Esto me consoló un poco, pero la mayoría de los problemas que esperaba que ocurriera, como ser anulado cada vez que encontraba un problema, nunca sucedieron. Mi preocupación no era que realmente terminaría pensando como un desarrollador y simplemente lanzando algo, sino que a mi gerente, que no era un evaluador, no le importaría tanto y tal vez no respaldara mis inquietudes con la aplicación.

Al final, creí que terminé pensando un poco más como un desarrollador, preocupándome menos por algunos de los pequeños errores. Mi mejor comprensión del funcionamiento de la aplicación me hizo comprender que el riesgo y el costo de arreglarla eran potencialmente mucho más riesgosos que el beneficio. Creo que pensar así no es malo siempre y cuando siempre seamos conscientes del impacto en el cliente final, no sólo del coste interno.

El corolario de mi forma de pensar más como desarrollador es que los desarrolladores comenzaron a pensar más como evaluadores. De hecho, soy partidario del papel adversario del evaluador, pero de forma relajada. De hecho, les doy a los desarrolladores estrellas doradas (el tipo de calcomanías pequeñas que solías recibir en tu examen de ortografía en segundo grado) cuando implementan un área de código que es especialmente sólida y fácil de usar, y les doy estrellas rosas cuando "implementan" un error que es especialmente atroz. Se quejan cuando me acerco, se preguntan qué he encontrado ahora y se alegran mucho de "hacer que mi trabajo sea aburrido" al probar su código ellos mismos y no darme nada que encontrar. No hace falta decir que se necesita el grupo adecuado para poder trabajar con este tipo de actitud falsamente hostil. Nunca he estado en otra empresa en la que esto hubiera funcionado, pero tampoco he trabajado en otra empresa en la que estallaran tiroteos espontáneos con nerfs.

La experiencia de Mark coincide con la nuestra y con la de muchos otros evaluadores que hemos conocido y que han pasado del desarrollo tradicional al ágil. Si es un evaluador que acaba de unirse a un equipo ágil, mantenga la mente abierta y considere cómo sus compañeros de equipo podrían tener diferentes puntos de vista.

El equipo comprende las necesidades de los clientes y ofrece servicios comerciales adecuados. valor. Pueden organizar actividades divertidas para generar buenas interacciones en equipo. Tener galletas o chocolate disponibles para los compañeros de equipo es una buena manera de lograr que ¡Camina hacia tu escritorio! La paciencia y la sensación de diversión son grandes ventajas.

EXPECTATIVAS DE LA GESTIÓN

Cuando pensamos en los desafíos que implica la adopción ágil, generalmente pensamos piense en el equipo real y los problemas que encuentra. Sin embargo, para una adopción ágil exitosa, la aceptación de la gerencia es fundamental. En un proyecto por fases, la dirección recibe actualizaciones periódicas y documentos de aprobación que indican el final del proyecto. cada fase. Es posible que los gerentes de nivel superior no comprendan cómo podrán para medir el progreso ágil del proyecto. Podrían temer una pérdida de control o falta de "proceso."

Cambios culturales para los gerentes

En un proyecto ágil, las expectativas cambian. En su vida anterior en cascada. proyectos, Janet recuerda haber escuchado comentarios como "esta función está completa en un 90%" por semanas. Ese tipo de métricas no tienen sentido en proyectos ágiles. Hay no hay aprobaciones que marquen el final de una fase, y el "estado de finalización" de un proyecto no está medido por puertas.

Cada equipo de proyecto determina métricas significativas. En Scrum, corre y los gráficos de avance de lanzamiento rastrean la finalización de la historia y pueden brindar a los gerentes una medida del progreso, pero no hay "fechas" difíciles de usar para facturar a los clientes. Las matrices de prueba se pueden utilizar para realizar un seguimiento de la cobertura de las pruebas de funcionalidad, pero no proporcionan documentación de aprobación.

El otro cambio que resulta difícil de entender para algunos directivos es permitir los equipos tomar sus propias decisiones técnicas y gestionan sus propias cargas de trabajo. Ya no es el gerente quien decide qué es lo suficientemente bueno. Es el equipo (que incluye al cliente) que define el nivel de calidad necesario para entregar una solicitud exitosa.

Los equipos ágiles estiman y trabajan en períodos de tiempo más pequeños que los tradicionales equipos. En lugar de aprovechar la contingencia, los equipos deben planificar suficiente tiempo para un buen diseño y ejecución con el fin de garantizar que la deuda técnica no aumentar. En lugar de gestionar las actividades del equipo a un nivel bajo, los directivos de los equipos ágiles se centran en eliminar obstáculos para que los miembros del equipo puedan hacer su mejor trabajo.

La historia de Janet

Le pregunté al vicepresidente a cargo de un gran proyecto ágil cuál consideraba la parte más difícil del nuevo entorno ágil desde una perspectiva de gestión. Dijo que en un proyecto tradicional tipo cascada, todos los informes mostraban que todo iba según lo planeado hasta el final, y luego todo entró en un estado de pánico y “nada funcionó”.

En el proyecto ágil, todos los días había problemas que debían abordarse.

Los proyectos ágiles requerían más trabajo de manera constante, pero al menos obtenía informes realistas. No hubo sorpresas al final del proyecto.

—Janet

A las partes interesadas del negocio no les gustan las sorpresas. Si se les puede convencer para que den Si el equipo tiene suficiente tiempo y recursos para hacer la transición, descubrirán que El desarrollo ágil les permite planificar con mayor precisión y alcanzar objetivos comerciales. en incrementos constantes.

A veces es en realidad la gestión la que impulsa la decisión de empezar a hacer desarrollo ágil. Los líderes empresariales de la empresa de Lisa optaron por probar la metodología ágil. desarrollo para resolver su crisis de software. Para ser efectivos, necesitaban tener un conjunto diferente de expectativas de gestión. Tenían que ser sensibles a la dificultad de realizar grandes cambios, especialmente en una organización que no estaba funcionando bien.

En todos los casos, los gerentes necesitan mucha paciencia durante lo que podría ser una transición larga hacia un equipo ágil de alto funcionamiento. Es su trabajo asegurarse de proporcionar los recursos necesarios y permitir que cada individuo aprenda. cómo hacer un trabajo de alta calidad.

La historia de la transición de un director de pruebas

Tae Chang dirige un equipo en DoubleClick que realiza pruebas de un extremo a otro para garantizar que todos los puntos de integración, tanto ascendentes como descendentes del objetivo del cambio, estén cubiertos. Cuando implementaron Scrum, los equipos de desarrollo se reorganizaron en numerosos equipos de aplicaciones. Los problemas de comunicación provocaron que se pasaran por alto dependencias, por lo que el equipo de Tae dio un paso al frente para ayudar a garantizar que los problemas se detectaran tempranamente.

Tae nos dijo: “Creo que el desarrollo ágil magnificó efectivamente la importancia de la comunicación entre equipos y un esfuerzo coordinado de prueba de un extremo a otro. No fue fácil desarrollar una estrategia no invasiva (en términos de adaptación al sprint actual).

estructura) proceso de prueba de integración; de hecho, todavía lo estamos modificando, pero el beneficio general de tal esfuerzo de prueba es evidente". Sus equipos comenzaron a caer en la trampa de la "minicascada". "En retrospectiva", explica Tae, "una de las razones de esto es que comenzamos con el proceso ágil antes de internalizar las prácticas ágiles".

Sabiendo que la automatización de pruebas y la integración continua eran clave, los equipos de DoubleClick propusieron nuevas ideas, como un equipo especializado en creación y automatización para ayudar a los equipos de desarrollo a afrontar la situación. Trajeron capacitación de expertos para ayudarlos a aprender TDD y programación en pares. Comenzaron a tomar medidas para abordar la deuda técnica de su sistema heredado.

El equipo de Tae asiste a todas las sesiones de revisión y planificación de sprints, utilizando comunicación formal e informal para facilitar la comunicación multifuncional y coordinar pruebas y lanzamientos. Ha descubierto que ayuda que las reuniones sean pequeñas, breves y relevantes. También es partidario de que todos se sienten juntos en un área de trabajo abierta, en lugar de cubos separados.

Tae ofrece los siguientes consejos a los evaluadores que realizan la transición a la metodología ágil:

"El desarrollo ágil en general frustrará inicialmente a los evaluadores porque no tendrán acceso a la documentación completa de los requisitos o a las etapas definidas de las pruebas. Desde mi punto de vista del desarrollo ágil, en cualquier momento dado, el evaluador participará en tareas de múltiples etapas del proceso de desarrollo tradicional. Un evaluador puede estar sentado en una sesión de diseño con ingeniería y administración de productos (debería tomar notas aquí y comenzar a pensar en áreas de riesgo donde el cambio de código propuesto probablemente tendrá un impacto) y el mismo día trabajar en la automatización y ejecución de casos de prueba para los cambios propuestos. Es un cambio de mentalidad y algunas personas se adaptan más rápido que otras".

La experiencia de Tae refleja la nuestra y la de muchos otros equipos que hemos tenido.
Hablado a.

Si es gerente de control de calidad, esté preparado para ayudar a sus evaluadores a superar sus frustraciones al pasar de etapas de prueba secuenciales definidas a etapas de ritmo rápido en las que realizan tareas muy variadas en un día determinado. Ayudarles a adaptarse a la idea de que las pruebas ya no son una actividad separada que ocurre después del desarrollo, pero que las pruebas y la codificación son actividades integradas.

Si eres un evaluador u otro miembro del equipo que no recibe el soporte que necesitas necesitas en su transición al desarrollo ágil, piense en las dificultades sus gerentes podrían estar teniendo para comprender el desarrollo ágil. Ayuda Infórmemeles qué tipo de apoyo necesita.

Hablar el idioma del gerente

¿Qué entienden mejor los directivos de empresas? Es el resultado final: el retorno de la inversión (Retorno de la inversión). Para obtener el apoyo que necesita de su dirección, encuadre sus necesidades en un contexto que puedan comprender. el de tu equipo

La velocidad se traduce en nuevas funciones para hacer el negocio más rentable. Si necesita tiempo y fondos para aprender e implementar una herramienta de prueba automatizada, explique a la gerencia que, con el tiempo, las pruebas de regresión automatizadas le permitirán a su equipo va más rápido y ofrece más funcionalidad en cada iteración.

La historia de Lisa

Mi equipo necesita grandes bloques de tiempo para realizar refactorizaciones arriesgadas, como intentar dividir la base del código en varios módulos que se pueden construir de forma independiente. También necesitamos tiempo para actualizar a las últimas versiones de nuestras herramientas existentes o para probar nuevas herramientas. Todas estas tareas son difíciles de integrar en un sprint de dos semanas cuando también intentamos ofrecer historias para el negocio.

Le explicamos a nuestra dirección que si estas tareas de "ingeniería" se posponían demasiado, nuestra deuda técnica se acumularía y nuestra velocidad se ralentizaría. La cantidad de puntos de historia entregados en cada iteración disminuiría y las nuevas historias tardarían más en codificarse. La empresa tardaría cada vez más en obtener las nuevas funciones que necesitaba para atraer clientes.

Fue difícil para la empresa aceptar permitirnos dedicar una iteración de dos semanas cada seis meses para realizar el trabajo interno que necesitábamos para gestionar nuestra deuda técnica, pero con el tiempo pudieron ver los resultados en nuestra velocidad. Recientemente, uno de los gerentes preguntó si sería necesario realizar "sprints de ingeniería" con más frecuencia. Tanto el producto como el equipo están creciendo, y la empresa quiere asegurarse de que nuestra infraestructura y herramientas también crezcamos.

—Lisa

Como todos los miembros de un equipo ágil, los gerentes necesitan aprender muchos conceptos nuevos y descubrir cómo encajan como miembros del equipo. Utilice gráficos grandes y visibles (o sus equivalentes virtuales, según sea necesario) para asegurarse de que puedan seguir el progreso de cada iteración y lanzamiento. Busque formas de maximizar el retorno de la inversión. A menudo, la empresa solicitará una característica compleja y costosa cuando existe una más simple y solución más rápida que ofrece un valor similar. Asegúrate de explicar cómo tu trabajo del equipo afecta el resultado final. Colabora con ellos para encontrar lo mejor. manera para que las partes interesadas expresen los requisitos para cada nueva característica.

Las limitaciones presupuestarias son una realidad que enfrentan la mayoría de los equipos. Cuando los recursos son limitados, Tu equipo necesita ser más creativo. El enfoque de todo el equipo ayuda. Tal vez, como el equipo de Lisa, su equipo tenga un presupuesto limitado para comprar software, por lo que

Se tiende a buscar herramientas de automatización de pruebas de código abierto que generalmente no tienen un gran costo de compra inicial. Una herramienta que utiliza el mismo lenguaje que la aplicación no ayudará a los evaluadores que no son programadores a menos que los programadores colaboren con ellos para automatizar las pruebas. Aprovechar toda la experiencia del equipo le ayuda a trabajar dentro de las limitaciones del negocio.

Como ocurre con todos los desafíos que enfrenta su equipo, experimente con nuevas formas en que el equipo de desarrollo y la gerencia puedan ayudarse mutuamente para crear un producto valioso. Al mismo tiempo, independientemente de su enfoque de desarrollo, es posible que deba asegurarse de que algunos procesos, como el cumplimiento de los requisitos de auditoría, reciban la atención necesaria.

EL CAMBIO NO ES FÁCIL

El desarrollo ágil puede parecer acelerado, pero el cambio puede parecer glacial. Los equipos que son nuevos en Agile tardarán en dominar algunas prácticas que se han comprometido a utilizar. Hemos conocido a muchos evaluadores que se sienten frustrados porque sus ciclos de desarrollo “ágiles” son en realidad ciclos en minicascada. Estos evaluadores todavía están siendo presionados; simplemente sucede más a menudo. Las iteraciones terminan antes de que se puedan probar las historias. Los programadores se niegan o no pueden adoptar prácticas críticas como TDD o emparejamiento. El equipo deja la responsabilidad de la calidad en manos de los evaluadores, quienes no pueden realizar cambios en el proceso.

No existe ninguna magia que puedas utilizar para lograr que tu equipo realice cambios positivos, pero tenemos algunos consejos para los evaluadores que quieran lograr que sus equipos cambien de manera positiva.

Ser paciente

Nuevas habilidades como TDD son difíciles. Encuentre formas de ayudar a su equipo a tener tiempo para dominarlos. Encuentre cambios que pueda realizar de forma independiente mientras espera. Por ejemplo, mientras los programadores aprenden a escribir pruebas unitarias, implemente una herramienta de prueba GUI que pueda utilizar con una mínima ayuda. Ayuda al equipo a dar pequeños pasos. Recuerde que cuando las personas entran en pánico, vuelven a sus viejos hábitos, aunque esos hábitos no hayan funcionado. Concéntrese en pequeños incrementos positivos.

Déjalos sentir dolor

A veces sólo hay que ver el choque de trenes. Si sus sugerencias de mejora fueron rechazadas y el equipo falla, mencione su sugerencia nuevamente y pídale al equipo que considere probarla durante algunas iteraciones. Las personas están más dispuestas a cambiar en las áreas donde sienten más dolor.

Construya su credibilidad

Es posible que ahora esté trabajando con programadores que no han trabajado estrechamente con probadores antes. Muéstrelles cómo puede ayudar. Acude a ellos si tienes problemas que ha encontrado en lugar de abrir informes de errores. Pídale que revisen el código con usted antes de que lo registren. Cuando se den cuenta de que está aportando valor real, es más probable que escuchen sus ideas.

Trabaja en tu propio desarrollo profesional

Leer libros y artículos, asistir a reuniones y conferencias de grupos de usuarios, y aprender una nueva herramienta o lenguaje de programación. Comience a aprender el idioma en el que está codificada su aplicación y pregunte a los programadores de su equipo si puede emparejarse con ellos o si te darán clases particulares. Tus compañeros de trabajo respetarán tu deseo de mejorar tus habilidades. Si su grupo de usuarios local está dispuesto a escuchar su presentación sobre pruebas ágiles, o un boletín de software publica su artículo sobre automatización, tus compañeros de equipo podrían notar que tienes algo

También vale la pena escucharlo.

Cuidado con la mentalidad policial de calidad

Sea un colaborador, no un ejecutor. Podría molestarte si los programadores no lo hacen siga los estándares de codificación, pero no es su trabajo asegurarse de que lo hagan. Plantea tus problemas al equipo y pide su ayuda. Si ignoran una crítica problema que realmente está perjudicando al equipo, es posible que tengas que acudir a tu entrenador o gerente para obtener ayuda. Pero hazlo en una línea de "por favor, ayúdame a encontrar una solución". en lugar de uno de "hacer que estas personas se porten bien". Si estás viendo un problema, Hay muchas posibilidades de que otros también lo vean.

Vota con tus pies

Consulte la bibliografía para obtener algunos buenos recursos sobre cómo ser un agente de cambio eficaz para su equipo.

Has sido paciente. Has probado todos los enfoques que puedes imaginar, pero tu La gerencia no entiende el desarrollo ágil. Los programadores todavía arroja un código defectuoso y no comprobable "por encima de la pared" y ese código se publica tal cual a pesar de sus mejores esfuerzos, incluido trabajar jornadas de 14 horas. A nadie le importa sobre la calidad y te sientes invisible a pesar de tus mejores esfuerzos. tal vez sea el momento para buscar un mejor equipo. Algunos equipos están contentos como son y simplemente No siento suficiente dolor como para querer cambiar. Lisa trabajó en un equipo que prosperó sobre el caos, porque había frecuentes oportunidades de descubrir por qué El servidor falló y sé un héroe. A pesar de un proyecto exitoso que utilizó prácticas ágiles, volvieron a sus viejos hábitos y Lisa finalmente dejó de intentarlo. cámbialos.

RESUMEN

En este capítulo, hablamos sobre cómo las cuestiones culturales pueden afectar si los evaluadores y sus equipos pueden realizar una transición exitosa hacia el desarrollo ágil.

Consideré la cultura organizacional antes de realizar cualquier tipo de cambio.

A los evaluadores les resulta más fácil integrarse en equipos ágiles cuando toda su organización valora la calidad, pero los evaluadores con una mentalidad de "policía de calidad" tendrán dificultades.

Algunos evaluadores pueden tener problemas para adaptarse a la propiedad de la calidad de "todo el equipo", pero un enfoque de equipo ayuda a superar las diferencias culturales.

Los equipos de clientes y los equipos de desarrolladores deben trabajar en estrecha colaboración, y mostramos cómo los evaluadores pueden ser clave para facilitar esta relación.

Las grandes organizaciones que tienden a tener equipos de especialistas más aislados enfrentan desafíos culturales particulares en áreas como la comunicación y la colaboración.

Las principales barreras para el éxito de los evaluadores de la adopción ágil incluyen el miedo, la pérdida de identidad, la falta de capacitación, experiencias negativas previas con nuevos procesos de desarrollo y diferencias culturales entre roles.

Para ayudar a introducir cambios y promover la comunicación, sugerimos alentar a los miembros del equipo a discutir sus miedos y celebrar cada éxito, por pequeño que sea.

Directrices como la "Declaración de derechos de los evaluadores" brindan a los evaluadores confianza para plantear problemas y los ayudan a sentirse seguros mientras aprenden y prueban nuevas ideas.

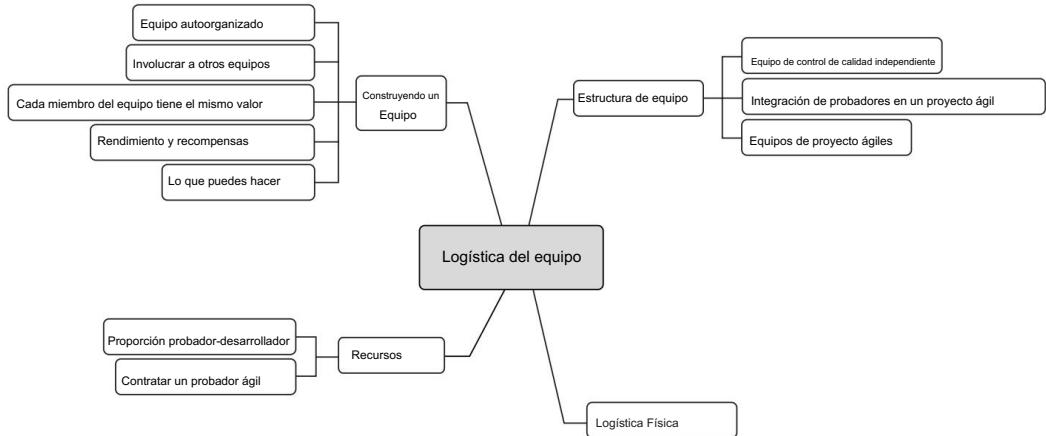
Los gerentes enfrentan sus propios desafíos culturales y necesitan brindar apoyo y capacitación para ayudar a los evaluadores a tener éxito en equipos ágiles.

Los evaluadores pueden ayudar a los equipos a satisfacer las expectativas de los gerentes brindándoles la información que los gerentes necesitan para realizar un seguimiento del progreso y determinar el retorno de la inversión.

El cambio no es fácil, así que tenga paciencia y trabaje para mejorar sus propias habilidades para poder ayudar a su equipo.

Capítulo 4

LOGÍSTICA DEL EQUIPO



Los equipos ágiles enfatizan que la comunicación cara a cara es fundamental para el éxito de un proyecto.

También alientan el uso del enfoque de "todo el equipo". ¿Qué significa esto para los evaluadores? Este capítulo trata sobre algunas de las cuestiones relacionadas con la estructura del equipo y la logística física. Crear un equipo cohesionado implica mucho más que simplemente mover sillas y escritorios.

ESTRUCTURA DE EQUIPO

Tener grupos funcionales separados puede dificultar la vida de los equipos ágiles.

La comunicación constante es fundamental. Los miembros del equipo deben trabajar en estrecha colaboración entre sí, ya sea que el trabajo se realice virtualmente o en el mismo lugar físico. ubicación.

Aquí utilizamos los términos "equipo de control de calidad" y "equipo de prueba" indistintamente. Puede ser discutido si los "equipos de control de calidad" realmente aseguran la calidad o no, pero el término se ha vuelto común en los equipos de prueba, por lo que nosotros también lo usamos.

Equipos de control de calidad independientes

Muchas organizaciones, tanto grandes como pequeñas, piensan que es importante tener un equipo de pruebas o control de calidad independiente para poder obtener una opinión honesta sobre la calidad. de un producto. A menudo nos preguntan: "¿Hay lugar para una organización de pruebas en el enfoque de todo el equipo?" y "Si es así, ¿cuál es su función?"

Algunas de las razones que nos dan para querer mantener separado el equipo de control de calidad del equipo de desarrollo son:

Es importante tener esa función de verificación y auditoría independiente.

El equipo puede proporcionar una visión imparcial y externa en relación con la calidad del producto.

Si los evaluadores trabajan demasiado estrechamente con los desarrolladores, empezarán a pensar como desarrolladores y perderán el punto de vista del cliente.

Si los evaluadores y desarrolladores informan a la misma persona, existe el peligro de que la prioridad sea entregar cualquier código en lugar de entregar código probado.

Los equipos suelen confundir "independiente" con "separado". Si la estructura de informes, los presupuestos y los procesos se mantienen en áreas funcionales discretas, una división entre programadores y evaluadores es inevitable. Esto puede provocar fricción, competencia y una actitud de "nosotros contra ellos". Se pierde tiempo en duplicados Las reuniones, los programadores y los evaluadores no comparten un objetivo común y el intercambio de información es inexistente.

Hay motivos para tener un responsable de control de calidad y un equipo de pruebas independiente.

Sin embargo, sugerimos cambiar tanto los motivos como la estructura. Bastante

En lugar de mantener a los evaluadores separados como un equipo independiente para probar la aplicación después de codificar, piense en el equipo como una comunidad de evaluadores. Proporcionar una organización de aprendizaje para ayudar a sus evaluadores con el desarrollo profesional y una lugar para compartir ideas y ayudarse unos a otros. Si el gerente de control de calidad se convierte en un líder de práctica en la organización, esa persona podrá enseñar las habilidades que

Los evaluadores deben volverse más fuertes y más capaces de hacer frente a los cambios constantes. ambiente.

No creemos que la integración de los testers con los equipos del proyecto impida que los evaluadores hagan bien su trabajo. De hecho, los evaluadores de equipos ágiles se sienten muy creen firmemente en su papel como defensores del cliente y también sienten que pueden influir en el resto del equipo en el pensamiento de calidad.

Integración de Testers en un Proyecto Ágil

El enfoque de equipo completo en el desarrollo ágil ha provocado que muchas organizaciones que han adoptado el desarrollo ágil disuelvan sus organizaciones independientes. equipos de control de calidad y envían a sus evaluadores a trabajar con los grupos de proyecto. Mientras esto suena genial, algunas organizaciones han descubierto que no funciona como se esperaba. Más de una organización ha tenido a la mayoría, si no a todos, sus evaluadores renunciaron cuando se encontraron en un equipo de desarrollo ágil sin idea lo que deberían estar haciendo.

Los desarrolladores reciben capacitación sobre programación en pares, desarrollo basado en pruebas y otras prácticas ágiles, mientras que los evaluadores a menudo parecen no recibir ninguna formación. Muchos Las organizaciones no reconocen que los evaluadores también necesitan capacitación en pruebas de pares, trabajar con requisitos incompletos y cambiantes, automatización y todo las otras nuevas habilidades que se requieren. Es fundamental que los evaluadores reciban capacitación y entrenamiento para que puedan adquirir las habilidades y la comprensión que ayudarles a tener éxito, por ejemplo, cómo trabajar con los clientes para redactar pruebas orientadas al negocio. Los programadores también pueden necesitar capacitación para comprender la importancia de las pruebas comerciales y el enfoque de redacción de todo el equipo. y automatizar pruebas.

Janet ha ayudado a integrar varios equipos de pruebas independientes en proyectos ágiles. Ella descubre que la mayoría de los evaluadores pueden tardar hasta seis meses en comenzar a sentirse Confiado en trabajar con el nuevo proceso.

La combinación de programadores y testers solo puede mejorar la comunicación. sobre la calidad del producto. Los desarrolladores a menudo necesitan observar el comportamiento de la aplicación en la estación de trabajo del evaluador si ese comportamiento no puede ser reproducido en el entorno de desarrollo. A veces los evaluadores pueden sentarse Acordarse con el desarrollador para reproducir un problema más fácil y rápidamente. que lo que pueden hacer al intentar registrar los pasos de un defecto. Esta interacción reduce el tiempo dedicado a la comunicación no oral.

Los comentarios que hemos escuchado de los evaluadores sobre este tema incluyen los siguientes:

"Estar más cerca del desarrollo del producto me convierte en un mejor evaluador".

"Ir a almorcazar con desarrolladores forma un mejor equipo, uno que quiere y le gusta trabajar juntos".

Una ventaja importante de un equipo de proyecto integrado es que sólo hay un presupuesto y un cronograma. No hay tiempo de "prueba" que recortar si no se termina toda la funcionalidad. Si no hay tiempo para probar una nueva característica, entonces hay. No hay tiempo para desarrollarlo en primer lugar. El enfoque de todo el equipo para tomar La responsabilidad por la calidad es muy poderosa, como señalamos a lo largo de este libro.

La historia de Lisa

Una vez me uní a un equipo de XP que dependía únicamente de las pruebas a nivel de unidad y nunca antes había tenido a nadie en el rol de probador. Su cliente no quedó muy satisfecho con los resultados, por lo que decidieron contratar a un evaluador. Si bien asistía a reuniones diarias, no se me permitía hablar sobre las tareas de prueba. El tiempo de prueba no se incluyó en las estimaciones de la historia y las tareas de prueba no formaron parte de la planificación de la iteración. Las historias se marcaron como "terminadas" tan pronto como se completaron las tareas de codificación.

Después de que el equipo no cumpliera con la fecha de lanzamiento, que estaba planeada para después de tres iteraciones de dos semanas, le pedí al entrenador del equipo que probara el enfoque de prueba de todo el equipo. Las tareas de prueba subieron al tablero junto con las tareas de codificación. Las historias ya no se consideraban terminadas hasta que se finalizaran las tareas de prueba. Los programadores asumieron tareas de prueba y yo participé plenamente en las reuniones diarias. El equipo no tuvo más problemas para cumplir con los planes de lanzamiento que establecieron.

—Lisa

Los evaluadores deben ser miembros de pleno derecho del equipo de desarrollo y las tareas de prueba deben recibir la misma atención que otras tareas. De nuevo, el enfoque de todo el equipo para las pruebas contribuye en gran medida a garantizar que las pruebas se completan al final de cada iteración y lanzamiento. Asegúrate de usar retrospectivas para evaluar qué necesitan los evaluadores integrar con su nueva metodología ágil. equipo y qué habilidades podrían necesitar adquirir. Por ejemplo, los evaluadores podrían Necesito más apoyo de programadores, o de alguien que sea un experto en un tipo particular de prueba.

Un enfoque inteligente para planificar los cambios organizacionales para un desarrollo ágil marca la diferencia para una transición exitosa. Pida a los gerentes de desarrollo y control de calidad que determinen sus propios roles en la nueva organización ágil. Permítales planificar cómo ayudarán a sus evaluadores y desarrolladores a ser productivos los nuevos equipos ágiles. Proporcionar formación en prácticas ágiles que el equipo no saber. Asegúrese de que todos los equipos puedan comunicarse entre sí. Proporcionar una marco que permite a cada equipo aprender sobre la marcha, y los equipos encontrarán una manera de tener éxito.

Transición de equipos de ingeniería y control de calidad: estudio de caso

Christophe Louvion es CTO y coach ágil para empresas de Internet de alto perfil. Nos contó una experiencia que tuvo mientras ayudaba a su empresa a implementar un desarrollo ágil. Como entrenador ágil, quería implementar verdaderamente un desarrollo ágil y evitar el error común de la “pequeña cascada”, donde los desarrolladores pasan una semana escribiendo código y los evaluadores pasan la semana siguiente probándolo.

Su empresa en ese momento era una organización de unos 120 ingenieros, incluidos los departamentos internos de TI. Antes de la transición a Scrum, la empresa estaba organizada funcionalmente. Había directores de control de calidad e ingeniería, y la idea de equipos basados en productos era difícil de aceptar para la gerencia. Los gerentes de estos equipos lucharon con la siguiente pregunta: “¿Cuál es mi trabajo ahora?” Christophe les dio la vuelta a los gerentes y dijo: “Díganmelo ustedes”.

Trabajó con los gerentes de ingeniería y control de calidad para ayudarlos a determinar cuáles serían sus trabajos en el nuevo entorno ágil. Sólo cuando pudieron hablar con una sola voz fueron todos a los equipos y explicaron sus hallazgos.

En la nueva organización ágil, los gerentes se ocupan de conocimientos de dominios específicos, recursos, priorización y problemas que surgen. Los responsables de Ingeniería y Control de Calidad trabajan mano a mano a diario para resolver este tipo de problemas. Christophe y los dos gerentes observaron qué impedía que los evaluadores fueran productivos en la primera semana de la iteración de dos semanas y les enseñaron cómo ayudar con el diseño.

Para los programadores, la pregunta fue “¿Cómo hago para que el código sea fácil de probar?” Los ingenieros no estaban capacitados en integración continua porque estaban acostumbrados a trabajar en ciclos escalonados. Necesitaban mucha capacitación en diseño basado en pruebas, integración continua y otras prácticas.

Sus gerentes se aseguraron de que recibieran esta capacitación.

Se contrató a expertos en gestión de configuración (CM) para ayudar con el proceso de construcción. El equipo de CM está separado de Ingeniería y Control de calidad en la empresa y proporciona el marco para todo el proceso de construcción, incluidos los objetos de la base de datos, el hardware y las configuraciones. Una vez que se implementó el marco del proceso de construcción, fue mucho más fácil hablar de la integración de la codificación y las pruebas.

Consulte la bibliografía para obtener un enlace a algunos de los escritos de Christophe sobre la gestión de equipos ágiles.

Hacer que la gerencia descubriera sus nuevos roles primero y luego implementar un marco de proceso de construcción con todo bajo control del código fuente fue clave para la transición exitosa a la metodología ágil. Otro factor de éxito fue que representantes de todos los equipos (ingeniería, control de calidad, CM, redes y grupos de administradores de sistemas y equipos de productos) participaran en reuniones diarias y actividades de planificación. De esta manera, cuando surgieran problemas con las pruebas, todos los que pudieran ayudar podrían solucionarlos. Como dice Christophe, su enfoque integra a todos y se centra en las pruebas.

Equipos de proyecto ágiles

Los equipos de proyectos ágiles generalmente se consideran multifuncionales, porque cada uno El equipo tiene miembros de diferentes orígenes. La diferencia entre un equipo multifuncional tradicional y un equipo ágil es el enfoque al esfuerzo de todo el equipo. Los miembros no sólo “representan” sus funciones en el equipo, sino que se convierten en verdaderos miembros del equipo mientras duren. el proyecto o equipo permanente existe (ver Figura 4-1).

Debido a que los proyectos varían en tamaño, los equipos de proyecto pueden variar en estructura. Las organizaciones con grandes proyectos o muchos proyectos que suceden simultáneamente son tener éxito utilizando una estructura de tipo matricial. Personas de diferentes áreas funcionales se combinan para formar un equipo virtual y al mismo tiempo informan a sus estructuras organizativas individuales. En una organización grande, un grupo de evaluadores podría pasar de un proyecto a otro. Algunos especialistas, como los evaluadores de seguridad o rendimiento, pueden compartirse entre varios equipos. Si estás empezando un proyecto, identifique todos los recursos que necesitará. Determina el número de probadores necesarios y el conjunto de habilidades necesarias antes de comenzar. El Los evaluadores comienzan con el equipo y continúan trabajando hasta que se completa el proyecto. y en ese momento pasan al siguiente proyecto.

Si bien los evaluadores son parte del equipo, su trabajo diario se gestiona igual que el resto del trabajo del equipo del proyecto. Un evaluador puede intercambiar nuevas ideas de la comunidad de testers más amplia, que incluye testers de diferentes proyectos equipos en una gran organización. Todos los evaluadores pueden compartir conocimientos e ideas. En organizaciones que practican revisiones de desempeño, el gerente de control de calidad (si es uno) podría impulsar las revisiones y obtener aportes del equipo del proyecto.

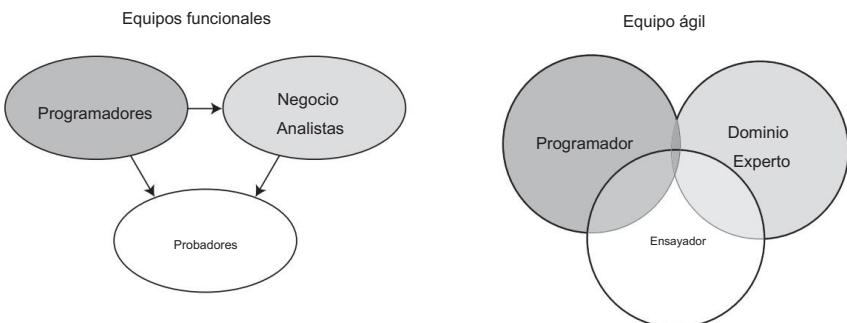


Figura 4-1 Estructura de equipos funcionales tradicionales versus estructura de equipo ágil

Como ocurre con cualquier equipo nuevo, un equipo tarda un tiempo en consolidarse. Si la longitud del proyecto es corto y los equipos cambian constantemente, la organización debe tener en cuenta que la primera o segunda iteración de cada proyecto incluirá los nuevos miembros del equipo se acostumbran a trabajar entre sí. Refactorizar su organización según sea necesario y recuerde incluir a sus clientes. El Los mejores equipos son aquellos que han aprendido a trabajar juntos y han desarrollado confianza unos con otros.

LOGÍSTICA FÍSICA

Muchas organizaciones que están pensando en adoptar metodologías ágiles intentan crear proyectos equipos sin coubicar el equipo en un entorno de plan abierto. Para respaldar los valores y principios ágiles, los equipos trabajan mejor cuando tienen acceso fácil a todos los miembros del equipo, fácil visibilidad de todos los gráficos de progreso del proyecto y una entorno que favorezca la comunicación.

Los probadores y clientes sentados cerca de los programadores permiten lo necesario comunalización. Si la logística prohíbe la coubicación, los equipos pueden ser inventivos.

La historia de Janet

Trabajé en un equipo donde el espacio impedía que todos los miembros del equipo se sentaran juntos. Los programadores tenían un área donde podían programar en pareja con facilidad, pero los evaluadores y el cliente estaban sentados en otra área. Al principio, fueron los evaluadores quienes hicieron el viaje al área del guión gráfico donde los programadores se sentaban para participar en stand-ups y cada vez que tenían una pregunta para uno de los programadores. Pocos de los programadores hicieron el viaje (unos 50 pies) hasta la casa de los evaluadores área. Comencé a tener a mano un plato de dulces con golosinas y animé a los desarrolladores a tomar algunos tan a menudo como quisieran. Pero había una regla: Necesitaba hacerle una pregunta a uno de los evaluadores si venían por dulces. Con el tiempo, la caminata se hizo más corta para todos los miembros del equipo. Ninguno de los lados hacía todo el recorrido y la comunicación floreció.

—Janet

El tamaño del equipo ofrece diferentes tipos de desafíos para la organización. Equipos pequeños significan áreas pequeñas, por lo que generalmente es más fácil ubicar a los miembros. Equipos grandes podría extenderse globalmente y se necesitan herramientas de comunicación virtual. La ubicación conjunta de equipos grandes generalmente significa renovar el espacio existente, algo que algunas organizaciones se muestran reacias a hacer. Comprenda sus limitaciones e intente encontrar soluciones a los problemas que encuentra su equipo en lugar de simplemente aceptar cosas como "como son".

La historia de Janet

Un equipo en el que trabajé comenzó en una esquina del piso, pero se expandió en el transcurso de tres años, ocupando gradualmente más del 70% del piso. Se derribaron muros, se eliminaron oficinas y se crearon grandes áreas abiertas. Las áreas abiertas y los grupos de equipos funcionaron bien, pero todo el espacio abierto significó que se perdiera espacio en las paredes. Las ventanas se convirtieron en guiones gráficos y pizarras blancas, y se encargaron pizarras blancas rodantes que podían usarse cuando los equipos las necesitaran.

—Janet

Los equipos ubicados conjuntamente no siempre viven en un mundo perfecto y los equipos distribuidos tenemos otro conjunto de desafíos. Los equipos distribuidos necesitan tecnología para ayudar se comunican y colaboran. Teleconferencias, videoconferencias, cámaras web y mensajería instantánea son algunas herramientas que pueden promover la comunicación en tiempo real. Colaboración para equipos en múltiples ubicaciones.

Ya sea que los equipos estén ubicados en el mismo lugar o distribuidos, las mismas preguntas generalmente averigüe qué recursos se necesitan en un equipo ágil y cómo obtenerlos. Discutiremos esto en la siguiente sección.

RECURSOS

Los nuevos miembros del equipo ágil y sus gerentes tienen muchas preguntas sobre el composición del equipo. ¿Podemos utilizar los mismos probadores que teníamos en nuestros proyectos tradicionales o necesitamos contratar un tipo diferente de probador? Cuántos probadores necesitaremos? ¿Necesitamos personas con otras habilidades especializadas? En esto En este apartado hablamos un poco de estas cuestiones.

Proporción probador-desarrollador

Ha habido muchas discusiones sobre la proporción "correcta" del número de probadores al número de desarrolladores. Esta proporción ha sido utilizada por organizaciones para determinar cuántos probadores se necesitan para un proyecto para que puedan contratar en consecuencia. Como ocurre con los proyectos tradicionales, no existe una proporción "correcta" y cada proyecto debe ser evaluado por sí solo. El número de probadores necesarios variará y dependerá de la complejidad de la aplicación, el conjunto de habilidades del los probadores y las herramientas utilizadas.

Hemos trabajado en equipos con un ratio tester-desarrollador de entre 1:20 a 1:1. Aquí tenéis un par de nuestras experiencias.

La historia de Janet

Trabajé en un proyecto con una proporción de 1:10 que desarrolló un sistema de manejo de mensajes. Había muy poca GUI y probé manualmente esa parte de la aplicación, observando la usabilidad y qué tan bien cumplía con las expectativas del cliente. Los programadores hicieron todas las pruebas de regresión automatizadas mientras yo trabajaba con ellos para validar la efectividad de los casos de prueba escritos. Probé historias en pareja con los desarrolladores, incluidas pruebas de carga de historias específicas.

Nunca sentí que no tuviera tiempo suficiente para realizar las pruebas que necesitaba, porque los desarrolladores creían que la calidad era responsabilidad de todo el equipo.

—Janet

La historia de Lisa

Una vez fui el único evaluador profesional en un equipo de hasta 20 programadores que desarrollaban un sistema de gestión de contenidos en un sitio web de compras por Internet. El equipo comenzó a volverse realmente productivo cuando los programadores asumieron la responsabilidad tanto de las pruebas manuales como de la automatización de las pruebas. Uno o dos programadores llevaban un "probador hat" para cada iteración, escribiendo pruebas orientadas al cliente antes de codificar y realizar pruebas manuales. Programadores adicionales asumieron las tareas de automatización de pruebas durante la iteración.

Por el contrario, mi equipo actual ha tenido dos probadores por cada tres o cinco programadores. La aplicación financiera basada en web que producimos tiene una lógica empresarial muy compleja, es de alto riesgo y requiere muchas pruebas. Las tareas de prueba a menudo suman la misma cantidad de tiempo que las tareas de programación. Incluso con un tester-programador relativamente alto En proporción, los programadores realizan gran parte de la automatización de pruebas funcionales y, a veces, realizan tareas de prueba manuales. Los evaluadores suelen realizar tareas de prueba especializadas, como escribir casos de prueba de alto nivel y pruebas detalladas de cara al cliente.

—Lisa

En lugar de centrarse en una proporción, los equipos deberían evaluar las habilidades de prueba que tienen. necesitar y encontrar los recursos adecuados. Un equipo que asume la responsabilidad de Las pruebas pueden evaluar continuamente si tiene la experiencia y el ancho de banda que necesita. necesidades. Utilice retrospectivas para identificar si existe un problema en la contratación. más probadores resolverían.

Contratar un probador ágil

Como analizamos en el Capítulo 2, "Diez principios para los evaluadores ágiles", existen ciertas cualidades que hacen que un evaluador sea apto para trabajar en un equipo ágil. nosotros no Quiero entrar en muchos detalles sobre qué tipo de probador contratar, porque cada La necesidad del equipo es diferente. Sin embargo, creemos que la actitud es un factor importante. factor. Aquí hay una historia de cómo el equipo de Lisa tuvo dificultades para contratar un nuevo evaluador ágil.

La historia de Lisa

Nuestro primer intento de contratar a otro evaluador no tuvo mucho éxito. La primera oferta de trabajo generó muchas respuestas y entrevistamos a tres candidatos sin encontrar uno que encajara bien. Los programadores querían a alguien "técnico", pero también necesitábamos a alguien con las habilidades para colaborar con gente de negocios y ayudarlos a producir ejemplos y requisitos. Nos esforzamos por determinar el contenido de la oferta de trabajo para atraer candidatos con la actitud y la mentalidad adecuadas.

Después de solicitar opiniones y sugerencias de Janet y otros colegas de la comunidad de pruebas ágiles, decidimos buscar un evaluador con la mentalidad que se describe en el Capítulo 2. Cambiamos la oferta de trabajo para incluir elementos como estos:

- Experiencia escribiendo casos de prueba de GUI y cajas negras, diseñando pruebas para mitigar riesgos y ayudando a expertos empresariales a definir requisitos.
- Experiencia escribiendo consultas SQL simples y declaraciones de inserción/actualización y conocimientos básicos. Conocimiento de Oracle u otra base de datos relacional.
- Al menos un año de experiencia con algún scripting o lenguaje de programación.
y/o herramientas de prueba de código abierto
- Capacidad para utilizar comandos básicos de Unix.
- Experiencia colaborando con programadores y expertos en negocios.
- Experiencia en pruebas basadas en contexto, exploratorias o de escenarios es una ventaja
- Capacidad para trabajar como parte de un equipo autoorganizado en el que usted determina sus tareas diariamente en coordinación con sus compañeros de trabajo en lugar de esperar a que le asignen el trabajo.

Estos requisitos trajeron candidatos más aptos para un trabajo de pruebas ágil. Procedí con cuidado en la selección, descartando a las personas con mentalidad de "policía de calidad". Los evaluadores que buscaron desarrollo profesional y mostraron interés en el desarrollo ágil tenían más probabilidades de tener la mentalidad adecuada. El equipo necesitaba a alguien que fuera sólido en el área de herramientas de prueba y automatización, por lo que la pasión por el aprendizaje era primordial.

Este enfoque más creativo para contratar un evaluador dio sus frutos. En ese momento no era Fue fácil encontrar buenos candidatos para "probadores ágiles", pero las búsquedas posteriores se realizaron sin problemas. Descubrimos que publicar el puesto de tester en lugares menos obvios, como una lista de correo de Ruby o el grupo local de usuarios ágiles, ayudó a llegar a una gama más amplia de candidatos adecuados.

Contratar a un evaluador ágil me enseñó mucho sobre la mentalidad de prueba ágil. Hay evaluadores con muy buenas habilidades que serían valiosos para cualquier equipo de pruebas tradicional, pero que no encajarían bien en un equipo ágil debido a su actitud hacia las pruebas.

—Lisa

Necesitamos considerar algo más que las funciones que desempeñan los evaluadores y programadores en el equipo. No importa qué rol intente desempeñar, la consideración más importante es cómo encajará esa persona en su equipo. Con el enfoque ágil de todo el equipo, se podría pedir a los especialistas del equipo que salgan de sus áreas de especialización y colaboren en otras actividades. Cada miembro del equipo debe tener un fuerte enfoque en la calidad y en la entrega de valor comercial. Considere algo más que las habilidades técnicas cuando esté ampliando su equipo.

CONSTRUYENDO UN EQUIPO

Hemos hablado mucho sobre el enfoque de todo el equipo. Pero cambios como ese no suceden por casualidad. Nos hacen preguntas como: "¿Cómo conseguimos que el equipo se solidifique?" o "¿Cómo promovemos el enfoque de equipo completo?" Uno de los más importantes es: "¿Cómo mantenemos a todos motivados y enfocados en el objetivo de generar valor comercial?"

Equipo autoorganizado

Según nuestra experiencia, los equipos logran mejores avances cuando están capacitados para identificar y resolver sus propios problemas. Si es gerente, resista la tentación de imponer todas sus buenas ideas al equipo. Hay problemas, como los de personal, que los gerentes resuelven mejor, y hay ocasiones en que un entrenador necesita brindar un fuerte estímulo y liderar al equipo cuando este necesita liderazgo. A un nuevo equipo ágil le lleva tiempo aprender a priorizar y resolver sus problemas, pero está bien que el equipo cometa errores y tropiece algunas veces. Creemos que un equipo de alto funcionamiento tiene que crecer por sí mismo. Si es un evaluador, está en una buena posición para ayudar al equipo a descubrir formas de obtener comentarios rápidos, utilizar prácticas como retrospectivas para priorizar y abordar problemas y encontrar técnicas que ayuden a su equipo a producir un mejor software.

Involucrar a otros equipos

Es posible que necesites incorporar a otros equipos para ayudar a tu equipo a tener éxito. Organizar reuniones; Encuentre formas de comunicarse tanto como sea posible. Utilice un Scrum de Scrums para mantener coordinados a varios equipos o simplemente involúrcese con los otros equipos. Si tiene que contratar a un experto para que le ayude con las pruebas de seguridad, asóciense con ese experto y aprenda todo lo que pueda y ayúdello a conocer su proyecto.

Si los equipos están dispersos en diferentes lugares y zonas horarias, descubra cómo lograr la mayor comunicación directa posible. Tal vez representantes de

El capítulo 9, "Kit de herramientas para pruebas empresariales que respaldan al equipo", ofrece ejemplos de herramientas que ayudan a los equipos remotos a colaborar.

cada equipo puede ajustar sus horarios una o dos veces por semana para poder realizar teleconferencias una vez por semana. Haz una llamada telefónica en lugar de enviar un correo electrónico cuando sea posible. El equipo de Lisa ajustó los horarios de sus reuniones de planificación para incluir un miembro del equipo remoto que trabaja hasta altas horas de la noche. Programan reuniones para un momento en el que su día se superpone con el resto del día del equipo.

Cada miembro del equipo tiene el mismo valor

Cada miembro del equipo tiene el mismo valor para el equipo. Si los probadores o cualquier otro equipo Los miembros se sienten excluidos o menos valorados, el enfoque de todo el equipo está condenado al fracaso. Asegúrese de que los evaluadores estén invitados a todas las reuniones. Si eres un tester y alguien se olvida de invitarte a una reunión, invítate tú mismo. Los evaluadores no técnicos podrían Creo que estarán fuera de lugar o abrumados en una reunión de diseño, pero a veces hacen buenas preguntas en las que los técnicos no pensaron.

Los evaluadores tienen derecho a solicitar y obtener ayuda. Si eres un evaluador atrapado en un problema de automatización, ten el coraje de pedirle ayuda a un miembro del equipo. Esa persona puede estar ocupada en este momento, pero debe comprometerse a ayudarlo de una manera cantidad de tiempo razonable. Si eres gerente o líder de tu equipo, asegúrate Asegúrese de que esto esté sucediendo y, si no es así, plantee el problema al equipo.

Rendimiento y recompensas

Medir y calificar el desempeño de manera individual corre el riesgo de socavar Colaboración en equipo. No queremos que un programador sienta que no debería tomar en una tarea de prueba porque está calificada para entregar código de producción. nosotros no Quiero que un administrador del sistema esté tan ocupado asegurándose de que sus objetivos individuales se enteran de que no puede ayudar con un problema del entorno de prueba.

Por el contrario, un buen jugador que intentaba trabajar bien con el equipo. No debería ser derribado porque el resto del equipo no se esforzó. Esto es un momento en el que un gerente necesita dar un paso adelante y ayudar al equipo a encontrar su camino. Si errores importantes llegaron a producción, nadie debería culpar a los evaluadores. En cambio, todo el equipo debe analizar lo sucedido y comenzar a tomar medidas para prevenir desahogar una recurrencia.

El equipo de desarrollo debe tener en cuenta las necesidades comerciales. Fijar metas que sirven al negocio, aumentan la rentabilidad y hacen más felices a los clientes. Trabajar estrechamente con el negocio para que sus éxitos ayuden al conjunto. la empresa tenga éxito.

Como mencionamos en el Capítulo 3, "Desafíos culturales", celebre cada éxito, por pequeño que sea. Una celebración podría consistir en chocar esos cinco, una reunión proporcionada por la empresa, almorzar, o tal vez simplemente salir temprano del trabajo para socializar un poco. El ScrumMaster Un miembro del equipo de Lisa reparte estrellas doradas en reuniones de stand-up por logros especiales. Reconoce a las personas que te ayudan a ti y a tu equipo.

Lea acerca de la idea de "Caja de zapatos que grite" en el Capítulo 19, "Conclusión de la iteración".

Los equipos pueden encontrar formas novedosas de reconocer las contribuciones de los demás. Iteración reuniones de revisión y demostración, donde tanto el equipo de desarrollo como El equipo del cliente está presente, son un buen escenario para reconocer tanto a los individuos como a sus clientes y logros del equipo.

¿Qué puedes hacer?

Si es un evaluador nuevo en un equipo ágil, especialmente en un equipo ágil nuevo, ¿qué puede hacer para ayudar al equipo a superar los desafíos organizacionales y tener éxito? ¿Cómo puedes encajar en el equipo y aportar tus habilidades particulares y experiencia?

Ponga en práctica los diez principios que describimos en el Capítulo 2. El coraje es especialmente importante. Levántate y ve a hablar con la gente; Pregunta cómo puedes ayudar. Alcanzar a los miembros del equipo y otros equipos con comunicación directa. Aviso impedimentos y pedirle al equipo que le ayude a eliminarlos.

El desarrollo ágil funciona porque elimina los obstáculos de nuestro camino y nos permite hacer nuestro mejor trabajo. Podemos sentirnos orgullosos y satisfechos, individualmente y como equipo. Cuando seguimos principios ágiles, colaboramos bien, utilizamos la retroalimentación para ayudar a mejorar la forma en que trabajamos y siempre buscamos nuevas y mejores formas de lograr nuestros objetivos. Todo esto significa que podemos mejorar continuamente la calidad de nuestro producto.

RESUMEN

En este capítulo, analizamos formas de crear un equipo y una estructura para realizar pruebas y desarrollo ágiles exitosos.

Considere la importancia de la estructura del equipo; Si bien los evaluadores pueden necesitar una mentalidad independiente, incluirlos en un equipo separado puede ser contraproducente.

Los evaluadores necesitan acceso a una comunidad más grande de evaluadores para aprender y probar nuevas ideas. Los equipos de control de calidad podrían crear esta comunidad dentro de su organización.

Es importante que todo el equipo esté ubicado junto, para fomentar la colaboración; si el equipo está distribuido, proporcionar herramientas para promover la comunicación.

Contratar por actitud.

No existe una proporción adecuada entre probadores y desarrolladores. La respuesta correcta es: "Depende de tu situación".

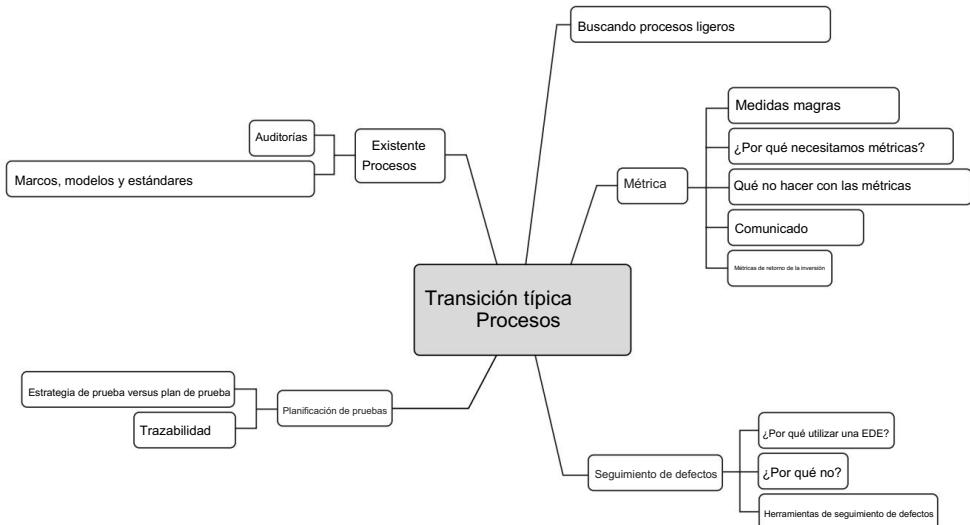
Los equipos necesitan autoorganizarse, identificar y encontrar soluciones a sus propios problemas y buscar formas de mejorar. No pueden esperar a que alguien les diga qué hacer.

La gerencia debe recompensar el desempeño de una manera que promueva el esfuerzo del equipo para entregar valor al negocio, pero no penalizar el buen desempeño individual si el equipo tiene dificultades.

Los evaluadores pueden utilizar principios ágiles para mejorar sus propias habilidades y aumentar su valor para el equipo. Deben ser proactivos y encontrar formas de contribuir.

Capítulo 5

TRANSICIÓN PROCESOS TÍPICOS



Hay muchos procesos en un proyecto tradicional que no hacen una buena transición a ágil porque requieren documentación pesada o son una parte inherente del proceso por fases y cerrado y requieren aprobaciones al final de cada etapa.

Como todo lo demás, no existen reglas estrictas y rápidas para realizar la transición de sus procesos a un proceso más ágil o liviano. En este capítulo, analizamos algunos de esos procesos y le brindamos alternativas y orientación sobre cómo trabajar con ellos en un proyecto ágil. Encontrará más ejemplos y detalles sobre estas alternativas en las Partes III, IV y V.

BUSCANDO PROCESOS LIGEROS

Cuando los equipos aprenden a utilizar procesos ágiles, algunos de los procesos más tradicionales pueden perderse en la confusión. La mayoría de los evaluadores que están acostumbrados a trabajar.

con metodologías tradicionales de desarrollo por fases y cerradas están acostumbrados hasta producir y utilizar métricas, registrar defectos en un seguimiento de defectos formal sistema y redactar planes de prueba detallados. ¿Dónde encajan en el desarrollo ágil?

Muchas organizaciones de software deben cumplir con sistemas de auditoría o modelos de procesos de calidad. Esos requisitos no suelen desaparecer sólo porque usted Comience a utilizar prácticas de desarrollo ágil. De hecho, a algunas personas les preocupa que la agilidad El desarrollo será incompatible con modelos y estándares como CMMI. e ISO 9000.

Quizás sea más divertido hablar sobre todo lo nuevo y diferente cuando pruebas en un proyecto ágil, pero todavía necesitamos formas de medir el progreso, rastrear defectos y planificar pruebas. También debemos estar preparados para trabajar con los modelos de calidad de nuestra organización. La clave es mantener estos procesos lo suficientemente livianos como para ayúdenos a entregar valor de manera oportuna. Comencemos mirando las métricas.

MÉTRICA

Las métricas pueden ser controvertidas y pasamos mucho tiempo hablando de ellas. Las métricas pueden ser un pozo de esfuerzo desperdiciado, números por el bien de los números. Ellos A veces se utilizan de forma dañina, aunque no tienen por qué ser malos. Pueden guiar a su equipo y ayudarlo a medir el progreso de su equipo hacia sus objetivos. Echemos un vistazo a cómo utilizar métricas para ayudar a los evaluadores ágiles y sus equipos.

Medidas magras

Los profesionales del desarrollo de software lean buscan formas de reducir el número de mediciones y encontrar mediciones que impulsen los comportamientos correctos. Implementación del desarrollo de software ajustado: del concepto al efectivo, por Mary y Tom Poppendieck, es un excelente recurso que enseña cómo aplicar lean iniciativas para sus esfuerzos de prueba y desarrollo.

Según Poppendiecks [2007], una medida magra fundamental es el tiempo que lleva pasar “del concepto al efectivo”, desde la solicitud de función de un cliente hasta el software entregado. A esta medida la llaman “tiempo de ciclo”. La atención se centra en la capacidad del equipo para generar nuevos negocios de manera “repetida y confiable”. valor. Luego, el equipo intenta mejorar continuamente su proceso y reducir el tiempo del ciclo.

Es más probable que se realicen mediciones como el tiempo de ciclo que involucran a todo el equipo. para conducirlo hacia el éxito que las medidas confinadas a roles aislados o

grupos. ¿Cuánto tiempo suele tardar en arreglar un defecto? ¿Qué puede hacer el equipo para reducir esa latencia, la cantidad de tiempo que lleva? Este tipo de métricas fomentan la colaboración para realizar mejoras.

Otra medida ajustada que los Poppendiecks explican en su libro es el rendimiento financiero. Si el equipo está desarrollando un producto rentable, necesita comprender cómo puede funcionar para lograr el mayor beneficio. Incluso si el equipo está desarrollando software interno o algún otro producto cuyo objetivo principal no es obtener ganancias, aún necesita analizar el retorno de la inversión (ROI) para asegurarse de que está entregando el mejor valor.

Identifique los objetivos comerciales y encuentre formas de medir lo que ofrece el equipo. ¿La empresa está intentando atraer nuevos clientes? Mantenga un registro de cuántas cuentas nuevas inician sesión a medida que se lanzan nuevas funciones.

El desarrollo eficiente busca maneras de complacer a los clientes, lo que debería ser el objetivo de todo desarrollo de software. Los Poppendieck dan ejemplos de formas sencillas de medir si sus clientes están satisfechos.

Nos gustan las métricas lean porque son congruentes con nuestro objetivo de ofrecer valor empresarial. ¿Por qué estamos interesados en las métricas? Hablaremos de eso en la siguiente sección.

Por qué necesitamos métricas

Hay buenas razones para recopilar y realizar un seguimiento de las métricas. También hay algunos realmente malos. Cualquiera puede utilizar buenas métricas de maneras terribles, como utilizarlas como base para la evaluación del desempeño de un miembro individual del equipo. Sin embargo, sin métricas, ¿cómo mides tu progreso?

Cuando las métricas se utilizan como guías (indicándole al equipo cuándo se está desviando o proporcionando comentarios de que está en el camino correcto), vale la pena recopilarlas.

¿Nuestra cantidad de pruebas unitarias aumenta todos los días? ¿Por qué la cobertura del código cayó del 75% al 65%? Podría haber sido una buena razón: tal vez nos deshicimos del código no utilizado que estaba cubierto por las pruebas. Las métricas pueden alertarnos sobre problemas, pero de forma aislada no suelen aportar valor.

Las métricas que miden los hitos a lo largo del camino para lograr los objetivos del equipo son útiles. Si nuestro objetivo es aumentar la cobertura del código de prueba unitaria en un 3 %, podríamos ejecutar la cobertura del código cada vez que realicemos el check-in para asegurarnos de no aflojar en las pruebas unitarias. Si no logramos la mejora deseada, es más importante descubrir por qué que lamentar la cantidad que nuestro bono se redujo como resultado. En lugar de centrarnos en mediciones individuales, deberíamos centrarnos en el objetivo y la tendencia hacia alcanzar ese objetivo.

Las métricas ayudan al equipo, incluidos los clientes, a realizar un seguimiento del progreso dentro de la iteración y dentro de la versión o épica. Si utilizamos un gráfico de evolución y Estamos quemando hacia arriba en lugar de hacia abajo, eso es una señal de alerta para detenernos, eche un vistazo a qué está pasando y asegurarnos de que entendemos y abordamos los problemas.

Quizás al equipo le faltaba información importante sobre una historia. Las métricas, incluidos los gráficos de evolución, no deben utilizarse como forma de castigo o fuente de culpa. Por ejemplo, preguntas como "¿Por qué sus estimaciones también fueron ¿bajo?" o "¿Por qué no puedes terminar todas las historias?" sería mejor venir del equipo y redactado como "¿Por qué nuestras estimaciones fueron tan bajas?" y por qué ¿No terminamos nuestras historias?

Las métricas, utilizadas correctamente, pueden resultar motivadoras para un equipo. El equipo de Lisa rastrea el Número de pruebas unitarias ejecutadas en cada compilación. Grandes hitos: 100 pruebas, 1000 pruebas, 3.000 pruebas son motivo de celebración. Hacer que ese número de pruebas unitarias aumente Cada día hay una buena retroalimentación para los equipos de desarrollo y de clientes. Sin embargo, es importante reconocer que el número en sí no significa nada. Por ejemplo, las pruebas pueden estar mal escritas o, para tener un producto bien probado, tal vez necesitemos 10.000 pruebas. Los números no funcionan de forma aislada.

La historia de Lisa

Pierre Veragen me habló de un equipo en el que trabajaba que era alérgico a las métricas. Los miembros del equipo decidieron dejar de medir cuánto código cubrían sus pruebas. Cuando decidieron volver a medir después de seis meses, quedaron atónitos al descubrir que la tasa había caído del 40% al 12%.

¿Cuánto te está costando no utilizar las métricas adecuadas?

—Lisa

Cuando esté tratando de descubrir qué medir, primero comprenda qué problema que estás tratando de resolver. Cuando conoces el enunciado del problema, puede establecer una meta. Estos objetivos deben ser mensurables. "Reducir la respuesta promedio tiempo en la aplicación XYZ a 1,5 segundos con 20 usuarios simultáneos" funciona mejor que "Mejorar el rendimiento de la aplicación XYZ". Si tus objetivos son mensurable, las medidas que necesitas recopilar para realizar un seguimiento de las métricas ser obvio.

Recuerde utilizar las métricas como fuerza motivadora y no para derrotar a un la moral del equipo. Vale la pena repetir esta sabiduría: céntrese en el objetivo, no en las métricas. Tal vez no estés usando las métricas correctas para medir si estás logrando los objetivos de tu equipo, o quizás no los estás interpretando de manera contexto. Un mayor número de informes de defectos podría significar que el equipo está haciendo un mejor trabajo de prueba, no que estén escribiendo más código con errores. Si tu

las métricas no le ayudan a comprender su progreso hacia su objetivo, es posible que tenga las métricas incorrectas.

Qué no hacer con las métricas

Mark Twain popularizó el dicho, que atribuyó a Benjamin Disraeli: "Hay tres tipos de mentiras: mentiras, malditas mentiras y estadísticas". Las metas mensurables son algo bueno; Si no puedes medirlos de alguna manera, no puedes decirlo.

si los lograste. Por otro lado, el uso de métricas para juzgar a los individuos o el desempeño del equipo es peligroso. Las estadísticas por sí solas pueden tergiversarse en cualquier interpretación y utilizado de manera perjudicial.

Tomemos como ejemplo las líneas de código, una vara de medir del software tradicional. ¿Hay más líneas de codificar algo bueno, lo que significa que el equipo ha sido productivo, o algo malo, ¿Significa que el equipo está escribiendo código estilo espagueti ineficiente?

¿Qué pasa con el número de defectos encontrados? ¿Tiene algún sentido juzgar a los evaluadores? ¿Por el número de defectos que encontraron? ¿Cómo les ayuda eso a hacer su trabajo? ¿mejor? ¿Es seguro decir que un equipo de desarrollo que produce una mayor cantidad de defectos por línea de código está haciendo un mal trabajo? O que un equipo que encuentre ¿Más defectos está haciendo un buen trabajo? Incluso si ese pensamiento se mantiene, ¿cuán motivador es para un equipo recibir un golpe en la cabeza con números? ¿Eso ¿Hacer que los miembros del equipo comiencen a escribir código libre de defectos?

Comunicación de métricas

Sabemos que cualquier cosa que midamos está destinada a cambiar. cuantas pruebas son corriendo y pasando? ¿Cuántos días faltan para que necesitemos una "construcción disponible"? Es ¿Pasa la construcción completa? Las métricas que no podemos ver e interpretar fácilmente no valen la pena teniendo. Si desea realizar un seguimiento del número de pruebas aprobadas, asegúrese de que la métrica sea visible de la manera correcta y para las personas adecuadas. Los grandes gráficos visibles son la forma más eficaz de mostrar métricas que conocemos.

La historia de Lisa

Mi equipo anterior tenía objetivos relacionados con la cantidad de pruebas unitarias. Sin embargo, la cantidad de pruebas unitarias aprobadas no se comunicó a nadie; no había grandes gráficos visibles ni correos electrónicos de compilación que hicieran referencia a ese número. Curiosamente, el equipo nunca logró automatizar las pruebas unitarias.

En mi empresa actual, todos los miembros de la empresa reciben periódicamente un informe del número de pruebas aprobadas en los niveles de unidad, detrás de la GUI y GUI (consulte las Tablas 5-1 y 5-2 para ver ejemplos). Los empresarios se dan cuenta cuando ese número disminuye en lugar de aumentar. Con el tiempo, el equipo ha desarrollado una gran cantidad de pruebas útiles.

—Lisa

Tabla 5-1 Métricas iniciales y finales

Métrico	Al principio	Al final
NCSS-Whitney	69943	69887
NCSS - Ghidrah	41044	41978
Número de pruebas JUnit	3001	3062
Número de afirmaciones de Canoo/Watir	3215	3215
Número de afirmaciones de FitNesse	57319	61585

Tabla 5-2 Resultados diarios de compilación

Fecha	Resultado de la construcción
Viernes 25/01/2008	Pasó 3026 JUnits
Lunes 28/01/2008	Pasó 3026 JUnits
Martes 29/01/2008	Pasó 3027 JUnits
Miércoles 30/01/2008	Pasó 3033 JUnits
Jueves 31/01/2008	Pasó 3040 JUnits
Viernes 01/02/2008	Pasó 3058 JUnits
Lunes 4/02/2008	Pasó 3059 JUnits
Martes 2/5/2008	Pasó 3060 JUnits
Miércoles 2/6/2008	Pasó 3062 JUnits
Jueves 07/02/2008	Pasó 3062 JUnits

¿Valen la pena sus métricas? No midas por producir números. Piensa en lo que aprenderás de esos números. En el próximo En esta sección, consideraremos el retorno de la inversión que puede esperar de las métricas.

Métricas de retorno de la inversión

Cuando identifique las métricas que necesita, asegúrese de poder obtenerlas en un costo razonable. Si su construcción continua arroja números útiles, entrega buen valor. Estás ejecutando la compilación de todos modos, y si nos da información adicional, eso es salsa. Si necesita mucho trabajo adicional para obtener información, pregúntese si vale la pena.

El equipo de Lisa se tomó muchas molestias para realizar un seguimiento del tiempo real invertido por Historia versus tiempo estimado. ¿Qué aprendieron aparte del hecho obvio?

¿Que las estimaciones son solo eso? Poco. Algunos equipos experimentados descubren que Puede prescindir del gráfico de evolución de sprint porque el tablero de tareas proporciona suficiente información para medir su progreso. Pueden usar el tiempo dedicamos a estimar tareas y calcular las horas restantes en actividades más productivas.

Esto no significa que le recomendamos que deje de realizar el seguimiento de estas mediciones. Los nuevos equipos necesitan comprender su velocidad y su tasa de consumo, por lo que que puedan mejorar constantemente.

Las tasas de defectos son métricas de software tradicionales y es posible que no tengan mucha importancia. valor en un equipo que apunta a cero defectos. No hay mucho valor en conocer la tasa de errores encontrados y corregidos durante el desarrollo, porque encontrarlos y corregirlos es una parte integral del desarrollo. Si un evaluador muestra un defecto al programador que está trabajando en el código y se escribe una prueba unitaria y el error se soluciona de inmediato, a menudo no es necesario registrar un defecto. Sobre el Por otro lado, si muchos defectos llegan a producción sin ser detectados, puede haber valor. en el seguimiento del número para saber si el equipo mejora.

Cuando comenzó a reescribir su aplicación heredada con errores, el equipo de Lisa se fijó una meta de no más de seis errores de alta gravedad en el nuevo código informados después de que el código sea en producción durante un período de seis meses. Tener un objetivo que fuera sencillo y fácil de rastrear ayudó a motivar al equipo a encontrar formas de solucionar los errores. durante el desarrollo y superar este objetivo.

Calcule el retorno de la inversión de cada métrica y decida si realizar un seguimiento o mantenerlo. ¿El esfuerzo invertido en recopilarlo justifica el valor que ofrece? ¿Se puede comunicar y entender fácilmente? Como siempre, haz lo que funcione. para su situación. Experimente manteniendo una métrica particular durante algunos sprints y evaluar si está dando sus frutos.

Una métrica común relacionada con la calidad del software es la tasa de defectos. En el En la siguiente sección, analizamos las razones para realizar un seguimiento de los defectos o para no realizar un seguimiento de los mismos, y qué podemos aprender de ellos.

SEGUIMIENTO DE DEFECTOS

Una de las preguntas que se hace todo nuevo equipo ágil es: "¿Todavía ¿Rastrear errores en un sistema de seguimiento de defectos? No hay una respuesta sencilla, pero darte nuestra opinión al respecto y ofrecerte algunas alternativas para que puedas puede determinar qué se adapta a su equipo.

¿Por qué deberíamos utilizar un sistema de seguimiento de defectos (DTS)?

Muchos de nosotros, los evaluadores, hemos utilizado el seguimiento de defectos como única forma de comunicarnos. los problemas que vimos y es fácil seguir usando las herramientas con las que estamos familiarizados. A DTS es un lugar conveniente para realizar un seguimiento no sólo del defecto sino también de las prioridades, la gravedad y el estado, y para ver a quién está asignado. Muchos profesionales ágiles dicen que ya no necesitamos hacer esto, que podemos rastrear los defectos. en tarjetas o algún otro mecanismo sencillo. Podríamos escribir una prueba para mostrar la falla, corrja el código y mantenga la prueba en nuestra suite de regresión.

Sin embargo, existen razones para seguir utilizando una herramienta para registrar defectos y cómo estaban arreglados. Exploraremos algunos de ellos ahora.

Conveniencia

Una de las preocupaciones de no mantener un sistema de seguimiento de defectos es que hay No hay lugar para guardar todos los detalles del error. Los probadores están acostumbrados a registrar un error con mucha información, como cómo reproducirlo, en qué entorno en qué se encontró o qué sistema operativo o navegador se utilizó. Toda esta información no cabe en una tarjeta, entonces, ¿cómo se capturan esos detalles? Si usted es Confiando sólo en las cartas, también necesitas conversación. Pero con la conversación, los detalles se pierden y, a veces, el evaluador olvida exactamente lo que se hizo, especialmente si el error se encontró unos días antes de que el programador solucionara el problema.

Una EDE también es un lugar conveniente para guardar toda la documentación complementaria, como impresiones de pantalla o archivos cargados.

Base de conocimientos

Hemos escuchado razones para rastrear defectos como: "Necesitamos poder observar informes de errores antiguos". Intentamos pensar en razones por las que alguna vez necesitarías miré informes de errores antiguos y, mientras trabajábamos en este capítulo, Janet descubrió un ejemplo.

La historia de Janet

Cuando estaba probando el algoritmo de asiento previo en WestJet, encontré una anomalía. Le pregunté a Sandra, otra evaluadora, si alguna vez se había encontrado con el problema antes. Sandra recordaba vagamente algo al respecto, pero no exactamente cuáles fueron las circunstancias. Rápidamente hizo una búsqueda en Bugzilla y encontró el problema de inmediato. Se había cerrado por no ser válido porque la empresa había decidido que no valía la pena el tiempo que llevaría arreglarlo y el impacto fue bajo.

Poder buscarlo me salvó de correr de un lado a otro tratando de hacer preguntas o volver a ingresar el error y cerrarlo nuevamente. Porque los miembros del equipo se sientan.

Cerca uno del otro, nuestra conversación condujo a otra conversación con el analista de negocios del equipo. Esta conversación generó la idea de una página de preguntas frecuentes, una lista de problemas pendientes o algo parecido que proporcionaría a los nuevos evaluadores un lugar para encontrar todos los problemas que se habían identificado pero que se había tomado la decisión de no abordar. a ellos.

—Janet

Esta historia muestra que aunque la base de datos de errores se puede utilizar como base de conocimiento base, podría haber otros mecanismos para mantener las decisiones comerciales y sus información de contexto. Si un problema es lo suficientemente antiguo como para haberlo perdido de vista, tal vez deberíamos reescribirlo y mencionarlo nuevamente. Las circunstancias pueden haber cambiado y la empresa podría decidir que ahora vale la pena corregir el error.

Los tipos de errores que es útil mantener en un DTS son los que son intermitentes y lleva mucho tiempo localizarlos. Estos errores se presentan con poca frecuencia y generalmente hay lapsos de tiempo durante los cuales la investigación se estanca por falta de información. Una EDE es un lugar donde la información puede ser capturado sobre lo que se descubrió hasta ahora. También puede contener registros, rastros, etcétera. Esta puede ser información valiosa cuando alguien del equipo finalmente tiene tiempo para analizar el problema o cuando el asunto se vuelve más crítico.

La información de los informes de errores se puede utilizar más adelante para varios propósitos. Aquí está una historia del equipo de Lisa sobre cómo utiliza su información.

La historia de Lisa

Un desarrollador de nuestro equipo participa en una rotación de "soporte de producción" para cada iteración. Las solicitudes de soporte de producción provienen del lado comercial para corregir manualmente errores pasados o problemas de producción que requieren intervención manual. La "persona de soporte de producción" investiga el problema y anota en el informe de error todo lo que se hizo para solucionarlo. Estas notas suelen incluir una declaración SQL e información sobre la causa. Si alguien se encuentra con el mismo error o situación más adelante, la solución se puede encontrar fácilmente en DTS. Si ciertos tipos de problemas parecen ocurrir con frecuencia, el equipo puede utilizar el DTS para investigación y análisis. Aunque nuestro equipo es pequeño, manejamos una gran cantidad de código heredado y no podemos confiar en la memoria de las personas para realizar un seguimiento de cada problema y solucionarlo.

—Lisa

Recordar la causa de los defectos o lo que se hizo para cumplir con una solicitud especial es aún más difícil cuando el equipo es particularmente grande o no comparte ubicación.

Los clientes también podrían estar interesados en las soluciones a sus problemas.

Equipos grandes o distribuidos

Si los proyectos son tan grandes que los defectos encontrados por un equipo podrían afectar a otros equipos, un DTS es probablemente una buena opción. Por supuesto, para ser útil debe serlo. accesible a todos los miembros del equipo. La comunicación cara a cara es siempre nuestra primera opción, pero cuando las circunstancias lo hacen impráctico, necesitamos ayuda como por ejemplo un DTS.

Atención al cliente Cuando

hay defectos que el cliente ha informado después del nuevo arrendamiento, el cliente normalmente quiere saber cuándo se han solucionado. Es muy valioso para la mesa de ayuda o el soporte técnico saber qué se solucionó en un liberación dada. También pueden encontrar defectos que aún están pendientes en el momento del lanzamiento. tiempo y avisar a los clientes. Un DTS hace que sea mucho más sencillo realizar esto. información en conjunto.

Métrica

Hay razones para realizar un seguimiento de las tasas de defectos. También hay razones por las que no lo harías. rastrear un defecto. Por ejemplo, no creemos que un error deba contarse como un defecto si nunca sale de la iteración. Esto, por supuesto, trae a colación otra discusión sobre qué debemos rastrear y por qué, pero no discutiremos eso aquí.

Capítulo 18,
"Codificación y pruebas",
explora métricas
relacionadas con el defecto
tarifas.

Trazabilidad

Otra razón que hemos escuchado para tener un DTS es la trazabilidad, vinculando defectos para probar casos. No estamos seguros de que esta sea una razón válida. No todos los defectos son vinculados a casos de prueba, ni deberían estarlo. Por ejemplo, es posible que errores como las faltas de ortografía no necesiten casos de prueba específicos. Quizás el producto no era intuitivo. usar; Este es un error muy real que a menudo no se informa. ¿Cómo se escribe un ¿Prueba para determinar si algo es utilizable? Las pruebas exploratorias pueden encontrar errores en condiciones de borde que no valen la pena el esfuerzo de crear pruebas automatizadas.

Si se trata de un caso de prueba automatizado que detectó un error, entonces es necesario registrarla. El defecto se reduce aún más, porque se volverá a detectar si alguna vez se reintroduce. La necesidad de trazabilidad ha desaparecido. Entonces, tal vez no necesitemos rastrear los defectos.

¿Por qué no deberíamos utilizar una EDE?

Agile y Lean nos proporcionan prácticas y principios que ayudan a reducir el necesidad de una EDE. Si el proceso es sólido y todas las personas están comprometidas Al entregar un producto de calidad, los defectos deben ser raros y rastrearse de manera muy sencilla.

Como herramienta de comunicación

Los sistemas de seguimiento de defectos ciertamente no promueven la comunicación entre programadores y probadores. Pueden hacer que sea más fácil evitar hablar directamente con entre sí.

Pérdida de tiempo e inventario

Tendemos a incluir mucha información en la EDE además de todos los pasos. para reproducir el defecto. Dependiendo del error, puede llevar mucho tiempo solucionarlo. Escriba estos pasos para que el programador pueda reproducirlos también. Entonces allí es la clasificación, y alguien tiene que hacer comentarios, interpretar el defecto, intentar reproducirlo, (idealmente) arreglarlo, escribir más comentarios y asignarlo. a la persona que lo informó. Finalmente, se puede verificar la solución. todo este El ciclo puede duplicarse si el programador entendió mal el problema en la primera lugar. El coste de un único informe de defecto puede llegar a ser exorbitante.

En el Capítulo 15, "Codificación y pruebas", explicaremos cómo los evaluadores y los programadores trabajan juntos para solucionar los errores.

¿Cuánto más fácil sería si nosotros, como evaluadores, pudiéramos hablar con el programador y mostrarle lo que encontramos, y luego el desarrollador solucionaría el defecto correctamente? ¿Lejos? Hablaremos más sobre eso más adelante.

Los defectos en un DTS se convierten en una cola o una mini acumulación de productos. De acuerdo a principios lean, este inventario de defectos es un desperdicio. Como equipo, deberíamos ser pensando en formas de reducir este desperdicio.

La historia de Janet

Antony compartirá sus ideas sobre el trabajo pendiente oculto cuando cubramos la planificación de iteraciones en el Capítulo 18, "Codificación". y Pruebas".

En 2004, Antony Marcano, autor de TestingReflections.com, escribió una publicación en un blog sobre la idea de no utilizar un sistema de seguimiento de errores. Cuando se discutió en las listas de correo, muchos evaluadores lo criticaron por introducir algo similar a la herejía.

Descubre que ahora tiene una recepción diferente, porque la idea se está abriendo paso en la corriente principal del pensamiento ágil.

Sugiere que los sistemas de seguimiento de errores en equipos ágiles son simplemente "atrasos secretos".

—Janet

Herramientas de seguimiento de defectos

Si decide utilizar una EDE, elíjala con cuidado. Entender sus necesidades y manténlo simple. Querrás que todos los miembros del equipo lo utilicen. si se convierte sobrecargado o difícil de usar, la gente encontrará formas de solucionarlo. Como con todo herramientas utilizadas por su equipo de desarrollo ágil, debe considerar todo opinión del equipo. Si alguien del equipo de atención al cliente ingresa informes de errores, obtenga su o su opinión también.

Una de las herramientas más sencillas que ha utilizado Janet es FIT IssueTrack de Alcea. Es configurable, no requiere seguir un proceso predefinido y es fácil de obtener métricas. Haga su tarea y encuentre la herramienta que funcione para usted. Existe una variedad de sistemas de seguimiento de defectos de código abierto, sistemas alojados, y sistemas empresariales integrados disponibles.

Independientemente de si utiliza un DTS o no, desea que los defectos sean lo más visibles posible.

Usamos un DTS comercial, pero consideramos valioso mantener los errores visibles. Codificamos los errores con colores y los incluimos como tareas en nuestro guión gráfico, como se muestra en la Figura 5-1. Las tarjetas amarillas indican errores normales y las tarjetas rojas indican errores de alta producción o errores de desarrollo que impiden las pruebas; ambas categorías deben abordarse de inmediato. Un vistazo rápido al tablero nos permite ver cuántos errores hay en las columnas Tareas pendientes, WIP, Verificar y Listo. Otras tarjetas también están codificadas por colores: azul para tarjetas de historia, verde para tarjetas de tareas de prueba y blanco para tareas de desarrollo. Las tarjetas rayadas son para tareas agregadas después de la planificación de la iteración. Las tarjetas de insectos amarillas y rojas se destacan fácilmente.



Figura 5-1 Guión gráfico con tarjetas codificadas por colores

Durante el tiempo que escribíamos este libro, mi equipo se convirtió a un guión gráfico virtual porque uno de los miembros de nuestro equipo se convirtió en miembro remoto del equipo, pero mantuvimos este concepto de codificación de colores.

—Lisa

Generalmente recomendamos experimentar con diferentes herramientas, usando cada una durante algunas iteraciones, pero esto es más complicado con los sistemas de seguimiento de errores, porque necesita portar todos los errores que hay en un sistema al nuevo que te estás probando el tamaño. Dedique algún tiempo a pensar en lo que necesita en un EDE, para qué servirá y evaluar alternativas juiciosamente.

La historia de Lisa

Mi equipo utilizó una DTS basada en web que básicamente le fue impuesta por la gerencia. Nos pareció algo engorroso de usar, carecía de funciones básicas como actualizaciones con marcas de tiempo en los informes de errores, y nos irritaban las restricciones de la licencia. Los evaluadores estábamos especialmente frustrados por el hecho de que nuestra licencia nos limitaba a tres usuarios simultáneos, por lo que las sesiones se agotaban rápidamente.

El equipo reservó tiempo para evaluar diferentes alternativas de DTS. Al principio, la selección parecía alucinante. Sin embargo, no pudimos encontrar una herramienta que cumpliera con todos nuestros requisitos. Parecía que a cada herramienta le faltaba algo importante, o escuchamos informes negativos de personas que habían usado la herramienta. Estábamos preocupados por el esfuerzo necesario para convertir la base de datos de errores existente en un nuevo sistema.

El problema se vio forzado cuando nuestro DTS falló. Habíamos dejado de pagar por soporte un par de años antes, pero el administrador del sistema decidió ver qué mejoras había realizado el proveedor en la herramienta. Descubrió que se habían solucionado muchas de las deficiencias que habíamos experimentado. Por ejemplo, todas las actualizaciones ahora tenían una marca de tiempo. Había disponible una aplicación cliente que no estaba sujeta a tiempos de espera de sesión y tenía características mejoradas que fueron particularmente valiosas para los evaluadores.

Al utilizar nuestra herramienta existente y pagar por la actualización y el mantenimiento, además de una licencia que permite más usuarios simultáneos, obtuvimos ayuda para convertir nuestros datos existentes a la nueva versión y obtuvimos un sistema que funciona fácilmente y a bajo costo. Una ventaja fue que nuestros clientes no tuvieron que aprender un nuevo sistema.

¡A veces la mejor herramienta es la que ya tienes si sólo miras y ves cómo ha mejorado!

—Lisa

Al igual que con todas sus búsquedas de herramientas, busque otras en su comunidad, como usuarios grupos y listas de correo, para recomendaciones. Define tus criterios antes empiezas a buscar y experimentas todo lo que puedas. Si eliges el herramienta equivocada, reduzca sus pérdidas y comience a investigar alternativas.

Mantén tu enfoque

Las decisiones sobre informes y seguimiento de defectos son importantes, pero no pierda Seguimiento de su objetivo principal. Quiere ofrecer el producto de mejor calidad posible, y desea entregar valor al negocio de manera oportuna. Proyectos



Capítulo 18,

"Codificación y pruebas",
cubre alternativas
y le muestra
diferentes formas de
atacar sus problemas
de errores.

tener éxito cuando a las personas se les permite hacer su mejor trabajo. Concéntrate en mejorar la comunicación y generar colaboración. Si encuentras muchos defectos, investiga la fuente del problema. Si necesita un DTS para hacer eso, use él. Si su equipo trabaja mejor documentando defectos en pruebas ejecutables y arreglarlos de inmediato, hazlo. Si alguna combinación te permite continuamente mejorar, seguir adelante. Lo principal que debe recordar es que tiene que funcionar para su todo el equipo.

El seguimiento de defectos es uno de los procesos de calidad típicos que genera más Preguntas y controversias en pruebas ágiles. Otra gran fuente de confusión es si los proyectos ágiles necesitan documentos como planes de prueba o matrices de trazabilidad. Consideremos eso a continuación.

PLANIFICACIÓN DE PRUEBAS

Las metodologías tradicionales de software por fases destacan la importancia de las pruebas. planes como parte de las necesidades generales de documentación. Están destinados a delinear Los objetivos, alcance, enfoque y enfoque del esfuerzo de prueba de software para partes interesadas. El documento completado está destinado a ayudar a personas ajenas El grupo de prueba comprende el "por qué" y el "cómo" de la validación del producto. En esta sección, analizamos los planes de prueba y otros aspectos de la preparación y el seguimiento del Esfuerzo de prueba para un proyecto ágil.

Estrategia de prueba frente a planificación de pruebas

En un proyecto ágil, los equipos no dependen de documentación pesada para comunicar lo que los evaluadores deben hacer. Los evaluadores trabajan mano a mano con el resto de el equipo para que los esfuerzos de prueba sean visibles para todos en forma de tarjetas de tareas. Por eso, la pregunta que a menudo nos plantean es: "¿Siguen siendo necesarios planes de prueba?" Para responder a esa pregunta, primero echemos un vistazo a la diferencia entre un plan de prueba y una estrategia o enfoque de prueba.

Cuanta más información contenga un documento, menos probable será que alguien lo va a leer todo. Considere qué información es realmente necesaria. para las partes interesadas. Piense en la frecuencia con la que se utiliza y para qué se utiliza.

Nos gusta pensar en una estrategia de prueba como un documento estático que rara vez cambia, mientras que se crea un plan de pruebas nuevo y específico para cada nuevo proyecto.

Estrategia de prueba

Una estrategia es un plan de acción a largo plazo, siendo la palabra clave "largo plazo". Si su organización quiere documentación sobre su enfoque general de prueba para

proyectos, considere tomar esta información y ponerla en un documento estático eso no cambia mucho con el tiempo. Hay mucha información que no específico del proyecto y se puede extraer en una estrategia de prueba o un enfoque de prueba documento.

Este documento se puede utilizar como referencia y solo debe actualizarse si los procesos cambian. Se puede utilizar un documento de estrategia de prueba para brindar a los nuevos empleados una comprensión de alto nivel de cómo funcionan sus procesos de prueba.

La historia de Janet

He tenido éxito con este enfoque en varias organizaciones. Los procesos que eran comunes a todos los proyectos se capturaron en un solo documento. El uso de este formato respondió a la mayoría de los requisitos de cumplimiento. Algunos de los temas que se trataron eran:

- Prácticas de prueba
- Prueba de historia
- Pruebas de verificación de soluciones
- Pruebas de aceptación del usuario
- Prueba exploratoria
- Pruebas de carga y rendimiento
- Automatización de pruebas
- Resultados de la prueba
- Proceso de seguimiento de defectos
- Herramientas de prueba
- Entornos de prueba

—Janet

Plan de prueba

El poder de la planificación es identificar posibles problemas y dependencias, traer riesgos a la superficie para ser discutidos y abordados, y para pensar en el panorama. La planificación de pruebas no es diferente. Un equipo debe pensar en los riesgos y dependencias y el panorama general de cada proyecto antes de que comience.

En el Capítulo 15, "Actividades del probador en la planificación del lanzamiento o del tema", mostramos ejemplos y analizamos alternativas que puede utilizar cuando planifica el lanzamiento.

Ya sea que su equipo decida crear un documento de plan de prueba o no, se debe realizar la planificación. Cada proyecto es diferente, así que no esperes que sea igual. La solución se adaptará a todos.

A veces nuestros clientes insisten en un documento de plan de prueba. Si contrata el desarrollo de una aplicación, un plan de prueba podría ser parte de un conjunto de entregables que también incluyan elementos como un documento de requisitos y un diseño.

Hablar de planes de prueba a menudo lleva a hablar de trazabilidad. ¿Alguien ejecutó todos? ¿Pruebas planificadas del comportamiento deseado en el código entregado? ¿Cómo se relacionan los requisitos y los planes de prueba con las pruebas reales y la funcionalidad final?

Trazabilidad

En proyectos tradicionales, solíamos necesitar matrices de trazabilidad para determinar si realmente habíamos probado todos los requisitos. Si un requisito cambiado, necesitábamos saber que habíamos cambiado los casos de prueba apropiados. Con documentos de requisitos muy extensos, esta era la única manera de que una prueba El equipo sabía que tenía buena cobertura.

En un proyecto ágil, no tenemos esas restricciones. Construimos funcionalidad en pasos pequeños y bien definidos. Trabajamos estrechamente con el equipo y sabemos cuándo algo cambia. Si los programadores trabajan primero en las pruebas, sabemos que hay pruebas unitarias para todas las pequeñas porciones de trabajo. Entonces podemos colaborar con el cliente para definir las pruebas de aceptación. Probamos cada historia mientras el programador trabaja en ella, para que sepamos que nada queda sin probar.

Podría haber requisitos para algún tipo de trazabilidad para las industrias reguladas. Si es así, le sugerimos que analice realmente qué problema está tratando de resolver la gestión. Cuando entiendas lo que se necesita, deberías intentarlo. para que la solución sea lo más sencilla posible. Hay varias formas de proporcionar trazabilidad. Los comentarios de registro del código fuente pueden consultar la página wiki que contiene los requisitos o casos de prueba, o a un número de defecto. Puedes poner comentarios en pruebas unitarias que vinculan la prueba con la ubicación o el identificador del requisito. Las pruebas se pueden integrar directamente con los requisitos en un herramienta como FitNesse. Su equipo puede encontrar fácilmente la forma que mejor funcione para las necesidades de sus clientes.

Es posible que se necesiten documentos como matrices de trazabilidad para cumplir con los requisitos impuestos por las normas de auditoría o los modelos de calidad de la organización. vamos Consideré cómo se llevan estas directivas con el desarrollo ágil.

PROCESOS Y MODELOS EXISTENTES

A menudo se plantea esta pregunta: “¿Pueden los modelos y procesos de calidad tradicionales coexistir con métodos de desarrollo ágiles? En teoría, no hay ninguna razón por la que no pueden. En realidad, muchas veces no hay elección. Los modelos de calidad a menudo caen en

el dominio del equipo de control de calidad tradicional, y pueden seguir a los evaluadores hasta el Nueva estructura ágil también. Puede que no sea fácil adaptarlos a una nueva metodología ágil. modelo de desarrollo. Veamos algunos procesos de calidad típicos y cómo Los evaluadores y sus equipos podrían adaptarse a ellos.

Auditorías

Diferentes industrias tienen diferentes requisitos de auditoría. Seguro de calidad Los equipos de las organizaciones de desarrollo tradicionales a menudo tienen la tarea de proporcionar información a los auditores y garantizar el cumplimiento de los requisitos de auditoría. La Ley Sarbanes-Oxley de 2002, promulgada en respuesta a importantes escándalos financieros corporativos, establece requisitos para mantener el negocio registros. Garantizar el cumplimiento suele recaer en los departamentos de TI. SAS 70 es Otra norma de auditoría ampliamente reconocida para organizaciones de servicios. Estos son sólo un par de ejemplos del tipo de controles de auditoría que afectan el desarrollo. equipos de mento.

Las organizaciones más grandes cuentan con equipos especializados que controlan el cumplimiento y Trabajan con auditores, pero a menudo se pide a los equipos de desarrollo que proporcionen información. Los ejemplos incluyen qué pruebas se han realizado en una versión de software determinada o demostrar que diferentes cuentas se concilian. A los probadores se les puede asignar tareas con la redacción de planes de prueba para evaluar la eficacia de las actividades de control.

La historia de Lisa

Nuestra empresa se somete periódicamente a auditorías SAS 70. Cada vez que se programa una, escribimos una tarjeta de historia para brindar apoyo a la auditoría. La mayor parte de este trabajo recae en los administradores del sistema, pero brindo apoyo a los empresarios que trabajan con el auditor. A veces se nos pide que demosbremos la funcionalidad del sistema en nuestro entorno de demostración. Puedo proporcionar datos para las demostraciones y ayudar si surgen preguntas. También es posible que me pidan que proporcione detalles sobre cómo probamos una funcionalidad en particular.

Algunos de nuestros procesos internos deben cumplir con los requisitos de SAS 70. Por ejemplo, cada vez que lanzamos a producción, completamos un formulario con información sobre qué compilación se lanzó, cuántas pruebas en cada nivel se ejecutaron en ella, quién realizó la publicación y quién la verificó.

—Lisa

Los evaluadores que forman parte de un equipo ágil deben estar dedicados a ese equipo. Si su ayuda es necesaria para proporcionar información para una auditoría o ayudar a garantizar

cumplimiento, escriba historias para esto y planifíquelas junto con el resto de la el trabajo del equipo. Trabajar en conjunto con los equipos de cumplimiento y auditoría interna para comprender las responsabilidades de su equipo.

Marcos, modelos y estándares

Hay muchos modelos de calidad, pero veremos dos para mostrarte cómo puedes Adapte su proceso ágil para que se ajuste a sus limitaciones.

1. La Integración del Modelo de Madurez de Capacidades (CMMI) tiene como objetivo ayudar a las organizaciones a mejorar sus procesos, pero no dicta prácticas de desarrollo específicas para lograr las mejoras.
2. La Biblioteca de infraestructura de tecnología de la información (ITIL) es un conjunto de mejores prácticas para la gestión de servicios de TI destinadas a ayudar a las organizaciones a desarrollar un proceso de calidad eficaz.

Ambos modelos pueden coexistir felizmente con un desarrollo ágil. Ellos son arraigados en el mismo objetivo, hacer que los proyectos de desarrollo de software tengan éxito.

Veamos CMMI, un marco para medir la madurez de su proceso. Define cada nivel midiendo si el proceso es desconocido, definido, documentado, permanente u optimizado. Los proyectos ágiles tienen un definido proceso, aunque no todos los equipos documentan lo que hacen. Por ejemplo, gestionar sus requisitos con fichas en una pared de planificación de lanzamientos con un Un solo cliente que toma las decisiones finales es un proceso definido siempre y cuando usted hazlo todo el tiempo.

Las retrospectivas tienen como objetivo la mejora constante de los procesos y los equipos deben Estar siempre buscando formas de optimizar los procesos. Si lo único que tu Al equipo le falta documentación, entonces piense en incluir su proceso. en la documentación de su estrategia de prueba.

Consulte la bibliografía para obtener información sobre CMMI y el desarrollo ágil.

Pregúntate cuál es la cantidad mínima de documentación que podrías entregar satisfacer los requisitos de CMMI sería. Janet ha tenido éxito con el uso diagramas como el de la Figura 5-2.

Si ITIL se ha introducido en su organización y afecta la gestión de cambios, adapte su proceso para adaptarse a ello. Incluso podrías encontrar el nuevo proceso beneficioso.

Iniciación del proyecto	Obtener una comprensión del proyecto.
Planificación de lanzamiento	Participa en historias de tallas Crear planes de prueba
Cada iteración 	Participar en la planificación de sprints, estimando tareas. Escribir y ejecutar pruebas de historias. Prueba en pareja con otros evaluadores y desarrolladores Validación empresarial (clientes) Automatizar nuevos casos de prueba funcionales Ejecutar casos de prueba de regresión automatizados Ejecutar pruebas de carga del proyecto Demostración para las partes interesadas
El final del juego (Prueba del sistema)	Implementación simulada de pruebas de gestión de versiones en preparación Prueba de humo en la puesta en escena Realizar prueba de carga (si es necesario) Prueba de regresión completa Los probadores de negocios realizan UAT Participar en la preparación del lanzamiento.
Liberar a Prod/ Apoyo	Participar en el lanzamiento a producción. Participar en retrospectivas.

Figura 5-2 Documentación de la estrategia de prueba

La historia de Janet

Cuando trabajaba en una organización que tenía un centro de llamadas central para manejar todas las llamadas de soporte de los clientes, la gerencia implementó ITIL para la parte de servicio de la organización. No pensamos que afectaría al equipo de desarrollo hasta que el equipo de gestión de cambios se dio cuenta de que la cantidad de problemas abiertos aumentaba constantemente. Nadie entendía por qué el número seguía aumentando, así que celebramos una serie de sesiones de resolución de problemas. Primero, trazamos el proceso actualmente en vigor.

El personal del call center informó de un incidente en su sistema de seguimiento. Intentaron solucionar el problema del cliente de inmediato. A menudo, eso significaba ofrecer una solución alternativa a un defecto de software. El informe del centro de llamadas se cerró, pero hubo un problema.

Luego se abrió el informe en Remedy y se envió un correo electrónico a alguien del equipo de desarrollo. Si el equipo de desarrollo aceptaba el defecto, se ingresaba un defecto en Bugzilla para solucionarlo.

No hubo ningún bucle de regreso al problema para cerrarlo cuando finalmente se solucionó el defecto. Celebramos varias sesiones de intercambio de ideas con todas las partes interesadas involucradas para determinar la mejor y más sencilla solución a ese problema.

El enunciado del problema a resolver era: "¿Cómo informa el equipo del proyecto sobre el problema y a la gente de gestión de cambios para informarles cuándo se solucionó realmente el error?"

Había un par de formas en que podríamos haber resuelto el problema. Una opción era hacer referencia al ticket de Remedy en Bugzilla y colocar ganchos en Remedy para que cuando cerráramos el defecto de Bugzilla, Remedy lo detectara y cerrara el ticket de Remedy. Por supuesto, algunos de los errores nunca se solucionaron, lo que significó que los tickets de Remedy permanecieron abiertos para siempre.

De hecho, encontramos una mejor solución para todo el equipo, incluida la gente del cambio de problemas. Hicimos una lluvia de ideas diferentes, pero decidimos que cuando se abriera un error en Bugzilla, podríamos cerrar el ticket de Remedy, porque, de manera realista, nunca volveríamos a la queja original y le diríamos al cliente quién lo informó, o cuándo se solucionó. estaba hecho.

La solicitud de cambio que cubría la versión incluiría automáticamente todas las correcciones de software, por lo que también seguía el proceso de gestión de cambios.

—Janet

Si su organización está utilizando algún tipo de modelo de proceso o estándares de calidad gestión, infórmese al respecto y trabaje con los especialistas adecuados en su organización. Mantenga el enfoque del equipo en la entrega de software de alta calidad que proporcione valor comercial real y vea cómo puede trabajar dentro del modelo.

Los modelos y marcos de mejora de procesos enfatizan la disciplina y la conformidad con el proceso. Pocas metodologías de desarrollo de software requieren más disciplina que el desarrollo ágil. Los estándares simplemente le permiten medir su progreso hacia su meta. El enfoque de Agile es hacer tu mejor trabajo y mejorando constantemente. El desarrollo ágil es compatible con el logro de cualquier estándar que usted mismo establezca o que tome prestado de una mejora de proceso. herramienta de medición.

Separe sus objetivos y estándares de medición de sus medios para mejorar esas mediciones. Establezca objetivos y sepá qué métricas necesita medir el éxito en áreas que necesitan mejorar. Intente utilizar tarjetas de tareas para

actividades que proporcionen las mejoras para garantizar que obtengan la visibilidad que necesitan.

Trabajar con procesos y modelos de calidad existentes es uno de los mayores Problemas culturales que puede enfrentar durante la transición al desarrollo ágil. Todo Estos cambios son difíciles, pero cuando todo el equipo se involucra, ninguno es insuperable.

RESUMEN

En este capítulo, analizamos los procesos tradicionales orientados a la calidad y cómo se pueden adaptar a un entorno ágil.

Las métricas correctas pueden ayudarlo a asegurarse de que su equipo esté encaminado para lograr sus objetivos y brindar un buen retorno de su inversión en ellos.

Las métricas deben ser visibles y proporcionar los hitos necesarios sobre los cuales tomar decisiones.

Las razones para utilizar un sistema de seguimiento de defectos incluyen la conveniencia, el uso como base de conocimientos y la trazabilidad.

Los sistemas de seguimiento de defectos se utilizan con demasiada frecuencia como herramienta de comunicación, e introducir y rastrear errores innecesarios puede considerarse un desperdicio.

Todas las herramientas, incluida la DTS, deben ser utilizadas por todo el equipo, así que considere todas las perspectivas al elegir una herramienta.

Una estrategia de prueba es un enfoque de prueba general a largo plazo que se puede incluir en un documento estático; un plan de prueba debe ser exclusivo del proyecto.

Piense en alternativas antes de aceptar ciegamente la necesidad de documentos específicos. Por ejemplo, el enfoque ágil para desarrollar en partes pequeñas e incrementales, trabajando en estrecha colaboración, podría eliminar la necesidad de documentos formales de trazabilidad. Otra forma podría ser vincular los comentarios del sistema de control del código fuente a las pruebas.

Los procesos de calidad tradicionales y los modelos de mejora de procesos, como las auditorías SAS 70 y los estándares CMMI, pueden coexistir con un desarrollo y pruebas ágiles. Los equipos deben estar abiertos a pensar de manera innovadora y trabajar juntos para resolver sus problemas.

Esta página se dejó en blanco intencionalmente.

Parte III

LAS PRUEBAS ÁGILES CUADRANTES

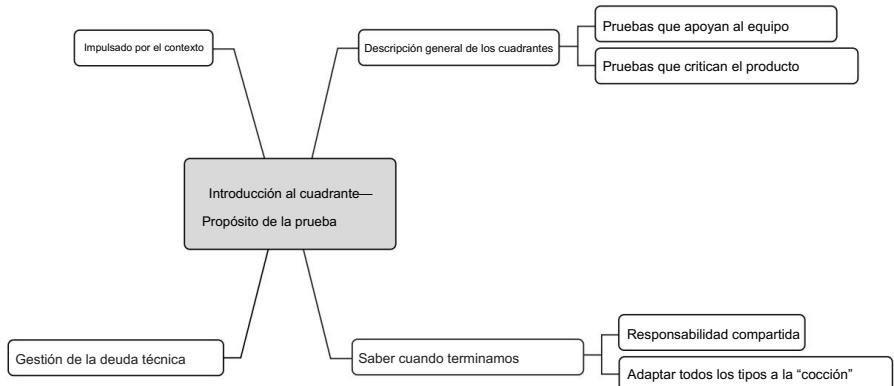
La calidad del software tiene muchas dimensiones, cada una de las cuales requiere un enfoque de prueba diferente. ¿Cómo sabemos todos los diferentes tipos de pruebas que debemos realizar? ¿Cómo sabemos cuando hemos “terminado” la prueba? ¿Quién hace qué pruebas y cómo? En esta parte, explicamos cómo utilizar los cuadrantes de pruebas ágiles para asegurarnos de que su equipo cubra todas las categorías de pruebas necesarias.

Por supuesto, las pruebas requieren herramientas, y hemos incluido ejemplos de herramientas a usar, estrategias para usar esas herramientas de manera efectiva y pautas sobre cuándo usarlas. Las herramientas son más fáciles de usar cuando se usan con código diseñado para la capacidad de prueba. Estas preocupaciones y más se analizan en esta parte del libro.

Esta página se dejó en blanco intencionalmente.

Capítulo 6

EL PROPÓSITO DE LAS PRUEBAS



¿Por qué hacemos pruebas? La respuesta puede parecer obvia, pero en realidad es bastante compleja. Realizamos pruebas por muchas razones: para encontrar errores, para asegurarnos de que el código sea confiable y, a veces, simplemente para ver si el código es utilizable. Realizamos diferentes tipos de pruebas para lograr diferentes objetivos. La calidad del producto de software tiene muchos componentes. En este capítulo, presentamos los cuadrantes de pruebas ágiles. El resto de los capítulos de la Parte III detallan cada uno de los cuadrantes. La matriz Agile Testing Quadrants ayuda a los evaluadores a asegurarse de haber considerado todos los diferentes tipos de pruebas que se necesitan para ofrecer valor.

LOS CUADRANTES DE PRUEBAS ÁGILES

En el Capítulo 1, “¿Qué son las pruebas ágiles, de todos modos?”, presentamos la explicación de Brian Marick. términos para diferentes categorías de pruebas que logran diferentes propósitos. La figura 6-1 es un diagrama de los cuadrantes de pruebas ágiles que muestra cómo cada uno de los cuatro cuadrantes reflejan las diferentes razones que probamos. En un eje dividimos la matriz en pruebas que apoyan al equipo y pruebas que critican el producto. El otro eje los divide en pruebas de negocios y de tecnología.

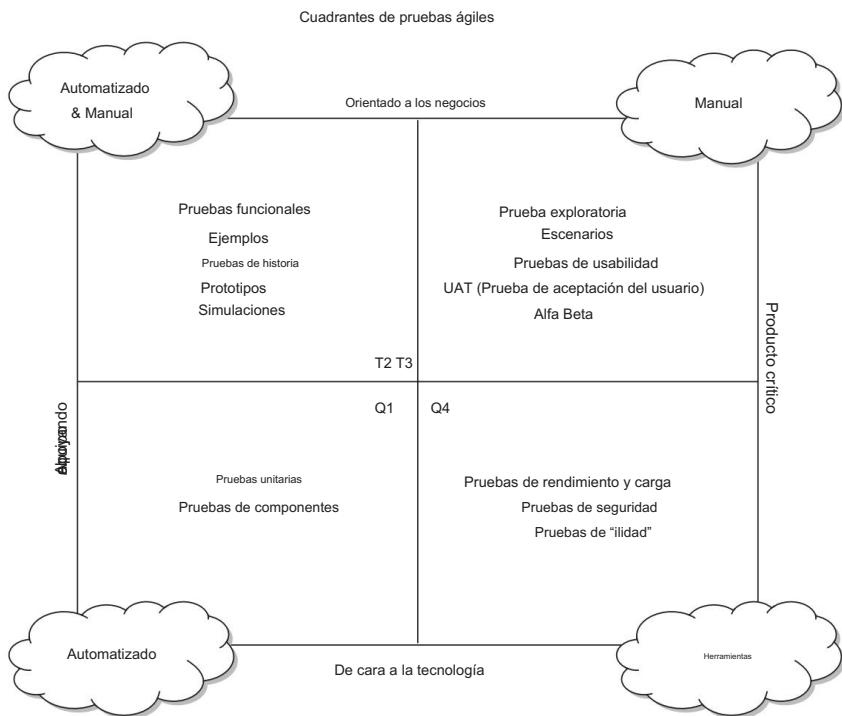


Figura 6-1 Cuadrantes de pruebas ágiles

El orden en el que hemos numerado estos cuadrantes no tiene relación con el momento en que se realizan los diferentes tipos de pruebas. Por ejemplo, el desarrollo ágil comienza con pruebas de clientes, que le indican al equipo qué codificar. El momento de los distintos tipos de pruebas depende de los riesgos de cada proyecto, los objetivos de los clientes para el producto, si el equipo está trabajando con código heredado o en un proyecto totalmente nuevo y cuándo hay recursos disponibles para realizar las pruebas.

Pruebas que respaldan al equipo Los

cuadrantes de la izquierda incluyen pruebas que respaldan al equipo mientras desarrolla el producto. Este concepto de prueba para ayudar a los programadores es nuevo para muchos evaluadores y es la mayor diferencia entre probar en un proyecto tradicional y probar en un proyecto ágil. Las pruebas realizadas en los cuadrantes 1 y 2 son más especificaciones de requisitos y ayudas de diseño que lo que normalmente consideramos pruebas.

Cuadrante 1

El cuadrante inferior izquierdo representa el desarrollo basado en pruebas, que es un núcleo práctica de desarrollo ágil.

Las pruebas unitarias verifican la funcionalidad de un pequeño subconjunto del sistema, como un objeto o método. Las pruebas de componentes verifican el comportamiento de una mayor parte del sistema, como por ejemplo un grupo de clases que brindan algún servicio [Meszaros, 2007]. Ambos Los tipos de pruebas generalmente se automatizan con un miembro de la familia de pruebas xUnit. herramientas de automatización. Nos referimos a estas pruebas como pruebas de programador, pruebas orientadas al desarrollador o pruebas orientadas a la tecnología. Permiten a los programadores medir lo que Kent Beck ha llamado la calidad interna de su código [Beck, 1999].

Un objetivo principal de las pruebas del Cuadrante 1 es el desarrollo basado en pruebas (TDD) o diseño basado en pruebas. El proceso de escribir pruebas primero ayuda a los programadores a diseñar bien su código. Estas pruebas permiten a los programadores escribir código con confianza para ofrecer las características de una historia sin preocuparse por hacer cambios no deseados. cambios en el sistema. Pueden verificar que sus decisiones de diseño y arquitectura sean apropiadas. Las pruebas unitarias y de componentes están automatizadas y escritas. en el mismo lenguaje de programación que la aplicación. Un experto en negocios Probablemente no podría entenderlos leyéndolos directamente, pero estas pruebas no están destinados al uso del cliente. De hecho, la calidad interna no se negocia. Con el cliente; Está definido por los programadores. Las pruebas del programador son normalmente parte de un proceso automatizado que se ejecuta con cada registro de código, brindando al equipo retroalimentación instantánea y continua sobre su calidad interna.

Cuadrante 2

Las pruebas en el Cuadrante 2 también respaldan el trabajo del equipo de desarrollo, pero a un nivel superior. Estas pruebas orientadas al negocio, también llamadas pruebas orientadas al cliente y pruebas de clientes, definen la calidad externa y las características que la ers quieren.

El Capítulo 8, "Pruebas de cara al negocio que respaldan al equipo", explica las condiciones de satisfacción del negocio.

Al igual que las pruebas del Cuadrante 1, también impulsan el desarrollo, pero a un nivel superior. Con un desarrollo ágil, estas pruebas se derivan de ejemplos proporcionados por el equipo del cliente. Describen los detalles de cada historia. Orientado a los negocios Las pruebas se ejecutan a nivel funcional, y cada una verifica una condición de satisfacción empresarial. Están escritos de una manera que los expertos en negocios pueden entender fácilmente usando el idioma del dominio empresarial. De hecho, los expertos en negocios utilizan estas pruebas para definir la calidad externa del producto y suelen ayudar a redactarlas. Es Es posible que este cuadrante pueda duplicar algunas de las pruebas que se realizaron en el nivel de unidad; sin embargo, las pruebas del Cuadrante 2 están orientadas a ilustrar y confirmar el comportamiento deseado del sistema a un nivel superior.

La mayoría de las pruebas empresariales que respaldan al equipo de desarrollo también necesita ser automatizado. Uno de los propósitos más importantes de las pruebas en estos dos cuadrantes es proporcionar información rápidamente y permitir una rápida resolución de problemas. Deben ejecutarse con frecuencia para brindarle al equipo información temprana en caso de que algún comportamiento cambie inesperadamente. Cuando sea posible, estos Las pruebas automatizadas se ejecutan directamente en la lógica empresarial en el código de producción. sin tener que pasar por una capa de presentación. Aún así, algunos automatizados Las pruebas deben verificar las interfaces de usuario y cualquier API que las aplicaciones cliente Puede usar. Todas estas pruebas deben ejecutarse como parte de un proceso automatizado continuo de integración, construcción y prueba.

Hay otro grupo de pruebas que también pertenece a este cuadrante. Los expertos en interacción de usuarios utilizan maquetas y estructuras alámbricas para ayudar a validar las propuestas. Diseños GUI (interfaz gráfica de usuario) con clientes y para comunicarse. esos diseños a los desarrolladores antes de que comiencen a codificarlos. las pruebas en Este grupo son pruebas que ayudan al equipo a crear el producto correctamente. pero no están automatizados. Como veremos en los siguientes capítulos, los cuadrantes ayudarnos a identificar todos los diferentes tipos de pruebas que necesitamos utilizar para ayudar a impulsar la codificación.

Algunas personas utilizan el término "pruebas de aceptación" para describir las pruebas del cuadrante 2, pero Creemos que las pruebas de aceptación abarcan una gama más amplia de pruebas que incluyen los cuadrantes 3 y 4. Las pruebas de aceptación verifican que todos los aspectos del sistema, incluidas cualidades como la usabilidad y el rendimiento, cumplen con los requisitos del cliente. requisitos.

Uso de pruebas para apoyar al equipo

La rápida retroalimentación proporcionada por las pruebas automatizadas de los cuadrantes 1 y 2, que ejecutar con cada cambio o adición de código, forma la base de una metodología ágil equipo. Estas pruebas primero guían el desarrollo de la funcionalidad y, cuando se automatizan, brindan una red de seguridad para evitar la refactorización y la introducción. de código nuevo provoque resultados inesperados.

La historia de Lisa

Ejecutamos nuestras pruebas automatizadas que respaldan al equipo (la mitad izquierda de los cuadrantes) en procesos de compilación separados. Las pruebas de unidades y componentes se ejecutan en nuestra compilación "continua", que tarda unos ocho minutos en finalizar. Aunque los programadores ejecutan las pruebas unitarias antes de registrarse, es posible que la compilación aún falle debido a problemas de integración o diferencias ambientales. Tan pronto como vemos el correo electrónico de "compilación fallida", la persona que ingresó el código infractor soluciona el problema. Las pruebas funcionales orientadas al negocio se ejecutan en nuestra "compilación completa", que también se ejecuta continuamente y se inicia cada vez que se registra un cambio de código. Finaliza en menos de dos horas. Sigue siendo una respuesta bastante rápida y, una vez más, un error de compilación implica una acción inmediata para solucionar el problema.

problema. Con estas compilaciones como red de seguridad, nuestro código es lo suficientemente estable como para publicarlo todos los días de la iteración si así lo decidimos.

—Lisa

Las pruebas de los cuadrantes 1 y 2 están escritas para ayudar al equipo a ofrecer el valor empresarial solicitado por los clientes. Verifican que la lógica de negocios y

Las interfaces de usuario se comportan según los ejemplos proporcionados por los clientes. Hay otros aspectos de la calidad del software, algunos de los cuales los clientes no piensan sin la ayuda del equipo técnico. es el producto

¿competitivo? ¿Es la interfaz de usuario tan intuitiva como debe ser? ¿Es segura la aplicación?

¿Están contentos los usuarios con el funcionamiento de la interfaz de usuario? Nosotros necesitamos diferentes pruebas para responder a este tipo de preguntas.

Pruebas que critican el producto

Si ha desempeñado un rol de cliente y ha tenido que expresar sus requisitos para un carácter del software, sabes lo difícil que puede ser saber exactamente lo que quieras hasta que lo veas. Incluso si está seguro de cómo debería funcionar la función funciona, puede ser difícil describirlo para que los programadores lo comprendan completamente.

La palabra "crítica" no tiene un sentido negativo. Una crítica puede incluir tanto elogios como sugerencias de mejora. Evaluación de un producto de software Implica tanto el arte como la ciencia. Revisamos el software de manera constructiva, con el objetivo de aprender cómo podemos mejorarlo. A medida que aprendemos, podemos alimentar nuevos requisitos y pruebas o ejemplos de regreso al proceso que respalda el desarrollo del equipo y guía.

Cuadrante 3

Los ejemplos comerciales ayudan al equipo a diseñar el producto deseado, pero al menos algunos de nuestros ejemplos probablemente serán incorrectos. Los expertos en negocios podrían pasar por alto la funcionalidad o no hacerlo del todo bien si no es su campo de pericia. Es posible que el equipo simplemente malinterprete algunos ejemplos. Incluso cuando los programadores escriben código que hace que las pruebas de cara al negocio pasen, ellos Es posible que no esté entregando lo que el cliente realmente quiere.

Ahí es donde entran en juego las pruebas para criticar el producto en el tercer y cuarto cuadrante. El cuadrante 3 clasifica las pruebas empresariales que ejercitan el software en funcionamiento para ver si no cumple con las expectativas o no lo hará.

hacer frente a la competencia. Cuando hacemos pruebas de cara al negocio para criticar el producto, intentamos emular la forma en que un usuario real trabajaría la aplicación. Este

Son pruebas manuales que sólo un humano puede realizar. Podríamos utilizar algunos automatizados

scripts para ayudarnos a configurar los datos que necesitamos, pero tenemos que usar nuestros sentidos, nuestra cerebros y nuestra intuición para comprobar si el equipo de desarrollo ha cumplido el valor de negocio requerido por los clientes.

A menudo, los usuarios y clientes realizan este tipo de pruebas. Aceptacion de usuario Las pruebas (UAT) brindan a los clientes la oportunidad de probar las nuevas funciones y ver qué cambios pueden querer en el futuro, y es una buena manera de reunir nuevas ideas para historias. Si su equipo entrega software por contrato para un cliente, la UAT podría ser un paso necesario para aprobar las historias terminadas.

Las pruebas de usabilidad son un ejemplo de un tipo de prueba que tiene toda una ciencia de su propio. Se podrían reunir grupos focales, estudiarlos mientras utilizan la aplicación y entrevistarlos para recopilar sus reacciones. Las pruebas de usabilidad pueden También incluye la navegación de una página a otra o incluso algo tan simple como el orden de tabulación. El conocimiento de cómo las personas usan los sistemas es una ventaja cuando probando la usabilidad.

Las pruebas exploratorias son fundamentales para este cuadrante. Durante las pruebas exploratorias sesiones, el evaluador diseña y realiza pruebas simultáneamente, utilizando elementos críticos pensando en analizar los resultados. Esto ofrece una oportunidad mucho mejor para aprender sobre la aplicación que las pruebas programadas. No estamos hablando de pruebas ad hoc, que son improvisadas e improvisadas. Las pruebas exploratorias son más enfoque reflexivo y sofisticado que las pruebas ad hoc. Está guiado por un estrategia y opera dentro de restricciones definidas. Desde el inicio de cada proyecto e historia, los evaluadores comienzan a pensar en escenarios que quieren probar. tan pequeño fragmentos de código comprobable están disponibles, los evaluadores analizan los resultados de las pruebas y, a medida que aprenden, encuentran nuevas áreas para explorar. Las pruebas exploratorias hacen funcionar el sistema de la misma manera que lo harán los usuarios finales. Los evaluadores usan su creatividad y intuición. Como resultado, es a través de este tipo de pruebas que muchos de los más Generalmente se encuentran errores graves.

Cuadrante 4

Los tipos de pruebas que se encuentran en el cuarto cuadrante son igualmente críticos para la metodología ágil. desarrollo como a cualquier tipo de desarrollo de software. Estas pruebas están orientadas a la tecnología y las analizamos en términos técnicos más que comerciales.

Las pruebas tecnológicas en el Cuadrante 4 tienen como objetivo criticar las características del producto, como el rendimiento, la solidez y la seguridad. Como describiremos en Capítulo 11, "Crítica del producto mediante pruebas tecnológicas", su El equipo ya posee muchas de las habilidades necesarias para realizar estas pruebas. Por ejemplo, los programadores podrían aprovechar las pruebas unitarias para convertirlas en pruebas de rendimiento. con un motor multihilo. Sin embargo, crear y ejecutar estas pruebas podría requerir el uso de herramientas especializadas y experiencia adicional.

En el pasado, hemos escuchado quejas de que el desarrollo ágil parece ignorar las pruebas tecnológicas que critican el producto. Estas quejas podría deberse en parte al énfasis de la metodología ágil en que los clientes escriban y prioricen historias. Los miembros no técnicos del equipo del cliente a menudo suponen que los desarrolladores se encargarán de cuestiones como la velocidad y la seguridad, y que el Los programadores tienen la intención de producir solo la funcionalidad priorizada por los clientes.

Si conocemos los requisitos de rendimiento, seguridad, interacción con otros sistemas y otros atributos no funcionales antes de comenzar a codificar, es más fácil de diseñar y codificar con eso en mente. Algunos de estos podrían ser más importante que la funcionalidad real. Por ejemplo, si un sitio web minorista en Internet tiene un tiempo de respuesta de un minuto, los clientes no esperarán para apreciar el hecho de que todas las funciones funcionan correctamente. Las pruebas tecnológicas que critican el producto deben considerarse en cada paso del ciclo de desarrollo y no dejarse para el final. En muchos casos, tales pruebas deberían incluso realizarse realizado antes de las pruebas funcionales.

En los últimos años hemos visto muchas herramientas nuevas y ligeras apropiadas para una metodología ágil. el proyecto de desarrollo esté disponible para soportar las pruebas. Las herramientas de automatización pueden Se puede utilizar para crear datos de prueba, configurar escenarios de prueba para pruebas manuales, impulsar pruebas de seguridad y ayudar a dar sentido a los resultados. La automatización es obligatoria para algunos esfuerzos como pruebas de carga y rendimiento.

Comprobación de requisitos no funcionales

Alessandro Collino, ingeniero en informática e información de Onion SpA, que trabaja en proyectos ágiles, ilustra por qué ejecutar pruebas que critiquen el producto en las primeras etapas del proceso de desarrollo es fundamental para proyectar éxito.

Nuestro equipo Scrum/XP utilizó TDD para desarrollar una aplicación Java que convertiría una forma de XML en otra. La aplicación realizó cálculos complejos sobre los datos. Para cada historia simple, escribimos una prueba unitaria para verificar la conversión de un elemento al formato requerido, implementamos el código para que la prueba pasara y lo refactorizamos según fuera necesario.

También escribimos pruebas de aceptación que leen subconjuntos de archivos XML originales del disco, los convierten y los reescriben. La primera vez que ejecutamos la aplicación en un archivo real para convertir, obtuvimos un error de falta de memoria. El analizador DOM que utilizamos para la conversión XML no pudo manejar un archivo tan grande. Todas nuestras pruebas utilizaron pequeños subconjuntos de archivos reales; No habíamos pensado en escribir pruebas unitarias utilizando grandes conjuntos de datos.

Hacer TDD nos brindó información rápida sobre si el código estaba funcionando según los requisitos funcionales, pero las pruebas unitarias no probaron ningún requisito no funcional, como capacidad, rendimiento, escalabilidad y usabilidad. Si utiliza TDD para comprobar también los requisitos no funcionales, en este caso la capacidad, obtendrá una respuesta rápida y podrá evitar errores costosos.

La historia de Alessandro es un buen ejemplo de cómo la numeración de cuadrantes no implica el orden en que se realizan las pruebas. Cuando el rendimiento de la aplicación sea crítico, planifique realizar pruebas con cargas de nivel de producción tan pronto como esté disponible el código comprobable.

Cuando usted y su equipo planeen una nueva versión o proyecto, analice qué tipos de pruebas de los cuadrantes 3 y 4 que necesita y cuándo deben realizarse.

No dejes para el final actividades esenciales como pruebas de carga o usabilidad, cuando podría ser demasiado tarde para rectificar los problemas.

Uso de pruebas que critiquen el producto

La información producida durante las pruebas para revisar el producto debe ser retroalimentado en el lado izquierdo de nuestra matriz y utilizado para crear nuevas pruebas para conducir desarrollo futuro. Por ejemplo, si el servidor falla bajo una carga normal, nuevos Se necesitarán historias y pruebas para impulsar una arquitectura más escalable. Usando Los cuadrantes le ayudarán a planificar pruebas que critiquen el producto, así como pruebas que impulsan el desarrollo. Piense por qué está realizando la prueba para asegurarse de que las pruebas se realizan en la etapa óptima de desarrollo.

Las breves iteraciones del desarrollo ágil le dan a su equipo la oportunidad de aprender y experimentar con los diferentes cuadrantes de prueba. Si te enteras demasiado tarde que su diseño no escala, comience las pruebas de carga antes con la siguiente historia o proyecto. Si la demostración de iteración revela que el equipo no entendió bien los requisitos del cliente, tal vez no esté haciendo un trabajo lo suficientemente bueno al escribir. pruebas de clientes para guiar el desarrollo. Si el equipo pospone la refactorización necesaria, tal vez las pruebas unitarias y de componentes no brinden suficiente cobertura. Usar los cuadrantes de pruebas ágiles para ayudar a garantizar que todas las pruebas necesarias se realicen en el tiempo justo.

SABER CUANDO SE HACE UNA HISTORIA

Para la mayoría de los productos, necesitamos las cuatro categorías de pruebas para sentirnos seguros. estamos entregando el valor correcto. No todas las historias requieren pruebas de seguridad, pero no querrás omitirlo porque no lo pensaste.

La historia de Lisa

Mi equipo utiliza tarjetas "stock" para garantizar que siempre consideremos todos los diferentes tipos de pruebas. Cuando las pruebas unitarias aún no eran un hábito, escribimos una tarjeta de prueba unitaria para cada historia en la pizarra. Nuestra tarjeta de prueba "de extremo a extremo" recuerda a los programadores que deben completar el trabajo de prueba de integración y asegurarse de que todas las partes del código funcionen juntas. También se considera una tarjeta de "seguridad" para cada historia y, si corresponde, se coloca en el tablero para que todos sean conscientes de la seguridad de los datos. Una tarjeta de tareas para mostrar la interfaz de usuario a los clientes garantiza que no nos olvidemos de hacer esto lo antes posible y también nos ayuda a comenzar las pruebas exploratorias junto con los clientes con anticipación. Todas estas tarjetas nos ayudan a abordar los diferentes aspectos de la calidad del producto.

Las pruebas tecnológicas que se extienden más allá de una sola historia tienen su propia fila en el guión gráfico. Usamos historias para evaluar herramientas de prueba de carga y establecer líneas de base de rendimiento para iniciar nuestros esfuerzos de prueba de carga y rendimiento.

—Lisa

Las pruebas orientadas a la tecnología y a los negocios que impulsan el desarrollo son fundamental para el desarrollo ágil, ya sea que realmente escriba o no tarjetas de tareas para a ellos. Le dan a su equipo la mejor oportunidad de "terminar" cada historia. Identificar las tareas necesarias para realizar las pruebas tecnológicas y empresariales que critican el producto garantiza que sabrá lo que le falta al producto. Una combinación de pruebas de los cuatro cuadrantes permitirá al

El equipo sabe cuándo cada característica ha cumplido con los criterios de funcionalidad y calidad del cliente.

Responsabilidad compartida

Nuestros equipos de productos necesitan una amplia gama de experiencia para cubrir todos los aspectos ágiles. cuadrantes de prueba. Los programadores deberían escribir las pruebas orientadas a la tecnología. que apoyan la programación, pero es posible que necesiten ayuda en diferentes momentos probadores, diseñadores de bases de datos, administradores de sistemas y especialistas en configuración. Los evaluadores se encargan principalmente de las pruebas comerciales junto con los clientes, pero los programadores participan en el diseño y la automatización pruebas, mientras que se puede llamar a expertos en usabilidad y otros expertos según sea necesario. El cuarto cuadrante, con pruebas tecnológicas que critican el producto, puede requieren más especialistas. No importa qué recursos haya que aportar desde fuera del equipo de desarrollo, el equipo sigue siendo responsable de conseguir Se realizaron los cuatro cuadrantes de las pruebas.

Creemos que un equipo exitoso es aquel en el que todos participan en el elaboración del producto y que todos comparten el dolor interno del equipo cuando las cosas van mal. Implementando las prácticas y herramientas que nos permitan

Abordar los cuatro cuadrantes de las pruebas puede ser doloroso a veces, pero la alegría de Vale la pena el esfuerzo de implementar un producto exitoso.

GESTIÓN DE LA DEUDA TÉCNICA

Ward Cunningham acuñó el término “deuda técnica” en 1992, ¡pero ciertamente lo hemos experimentado a lo largo de nuestras carreras en el desarrollo de software! La deuda técnica se acumula cuando el equipo de desarrollo toma atajos y piratea soluciones rápidas, o se salta la escritura o la automatización de pruebas porque está bajo presión. La base del código se vuelve cada vez más difícil de mantener. Al igual que la deuda financiera, los “intereses” se acumulan en forma de mayores costos de mantenimiento y menor velocidad del equipo. Los programadores tienen miedo de realizar cambios, y mucho menos intentarlo. refactorizar para mejorar el código, por miedo a romperlo. A veces este miedo existe porque, para empezar, no pueden entender la codificación y, a veces, es porque no existen pruebas para detectar errores.

Cada cuadrante de la matriz de pruebas ágiles desempeña un papel en mantener la calidad técnica. deuda a un nivel manejable. Pruebas orientadas a la tecnología que respaldan la codificación y El diseño ayuda a mantener el código mantenable. Una construcción e integración automatizadas El proceso que ejecuta pruebas unitarias es imprescindible para minimizar la deuda técnica. Atrapando Los defectos a nivel de unidad durante la codificación liberarán a los evaluadores para centrarse en el aspecto empresarial. pruebas con el fin de guiar al equipo y mejorar el producto. Carga oportuna y Las pruebas de estrés permiten a los equipos saber si su arquitectura está a la altura del trabajo.

Al tomarse el tiempo y aplicar recursos y prácticas para mantener el nivel técnico deuda al mínimo, un equipo tendrá tiempo y recursos para cubrir las pruebas necesarios para garantizar un producto de calidad. Aplicar principios ágiles para hacer el bien El trabajo de cada tipo de prueba en cada nivel minimizará, a su vez, la deuda técnica.

PRUEBAS EN CONTEXTO

Categorizaciones y definiciones como las que encontramos en la matriz de pruebas ágiles. ayúdenos a asegurarnos de planificar y lograr todos los diferentes tipos de pruebas que necesitamos. Sin embargo, debemos tener en cuenta que cada organización, El producto y el equipo tienen su propia situación única, y cada uno necesita hacer lo que trabaja para ello en su situación individual. Como le gusta al compañero de trabajo de Lisa, Mike Busse decir: "Es una herramienta, no una regla". Las necesidades de un solo producto o proyecto pueden evolucionar drásticamente con el tiempo. Los cuadrantes son una forma útil de asegurarse su equipo está considerando todos los diferentes aspectos de las pruebas que entran en juego "listo".

Podemos tomar prestados principios importantes de la escuela de pruebas impulsadas por el contexto al planificar las pruebas para cada historia, iteración y lanzamiento.

Para obtener más información sobre las pruebas basadas en contexto, consulte www.context-driven-testing.com.

El valor de cualquier práctica depende de su contexto.

Hay buenas prácticas en contexto, pero no hay mejores prácticas.

Las personas, trabajando juntas, son la parte más importante de cualquier proyecto.

contexto.

Los proyectos se desarrollan a lo largo del tiempo de maneras que a menudo no son predecibles.

El producto es una solución. Si el problema no se soluciona, el producto no funciona.

Las buenas pruebas de software son un proceso intelectual desafiante.

Sólo a través del juicio y la habilidad, ejercidos de manera cooperativa durante todo el proyecto, podemos hacer las cosas correctas en el momento adecuado para probar nuestros productos de manera efectiva.

Los cuadrantes ayudan a dar contexto a las prácticas de pruebas ágiles, pero usted y su equipo tendrá que adaptarse a medida que avanza. Los evaluadores ayudan a proporcionar comentarios al equipo. Necesita adaptarse y funcionar mejor. Utilice sus habilidades para atraer a los clientes a lo largo de cada iteración y lanzamiento. Sé consciente de cuando tu equipo necesita roles o conocimientos más allá de los que actualmente tiene disponibles.

Los cuadrantes de pruebas ágiles proporcionan una lista de verificación para asegurarse de haber cubierto todas tus bases de prueba. Examine las respuestas a preguntas como estas:

¿Estamos utilizando pruebas unitarias y de componentes para ayudarnos a encontrar el diseño adecuado para nuestra aplicación?

¿Tenemos un proceso de compilación automatizado que ejecuta nuestras pruebas unitarias automatizadas para obtener comentarios rápidos?

¿Nuestras pruebas comerciales nos ayudan a ofrecer un producto que coincide con las expectativas de los clientes?

¿Estamos capturando los ejemplos correctos del comportamiento deseado del sistema? ¿Necesitamos más? ¿Estamos basando nuestras pruebas en estos ejemplos?

¿Mostramos prototipos de UI e informes a los usuarios antes de comenzar a codificarlos? ¿Pueden los usuarios relacionarlos con cómo funcionará el software terminado?

¿Presupuestamos suficiente tiempo para las pruebas exploratorias? ¿Cómo abordamos las pruebas de usabilidad? ¿Estamos involucrando lo suficiente a nuestros clientes?

¿Consideramos los requisitos tecnológicos como el rendimiento y la seguridad lo suficientemente temprano en el ciclo de desarrollo? ¿Tenemos las herramientas adecuadas para realizar pruebas de "ilidad"?

Utilice la matriz como mapa para comenzar. Experimente y utilice retrospectivas para siga mejorando sus esfuerzos para guiar el desarrollo con pruebas y aprovechar lo que aprende sobre su producto a través de las pruebas.

RESUMEN

En este capítulo presentamos los cuadrantes de pruebas ágiles como una herramienta conveniente. Manera de categorizar las pruebas. Los cuatro cuadrantes sirven como guía para asegurar que Todas las facetas de la calidad del producto se cubren en el proceso de prueba y desarrollo.

Las pruebas que respaldan al equipo se pueden utilizar para impulsar los requisitos.

Las pruebas que critican el producto nos ayudan a pensar en todas las facetas de la calidad de la aplicación.

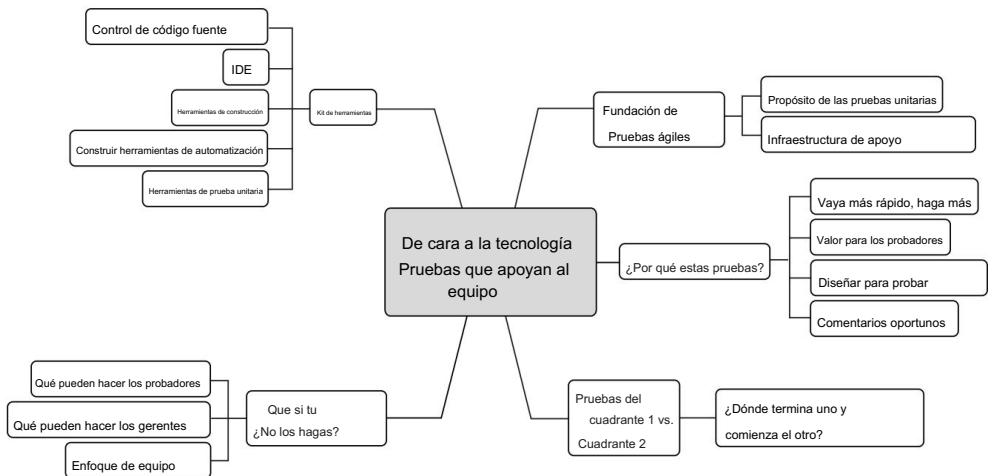
Utilice los cuadrantes para saber cuándo ha terminado y asegúrese de que todo el equipo comparta la responsabilidad de cubrir los cuatro cuadrantes de la matriz.

La gestión de la deuda técnica es una base esencial para cualquier equipo de desarrollo de software. Usa los cuadrantes para pensar en las diferentes dimensiones.

El contexto siempre debe guiar nuestros esfuerzos de prueba.

Capítulo 7

PRUEBAS DE TECNOLOGÍA QUE APOYAN AL EQUIPO



Usamos los cuadrantes de pruebas ágiles como guía para ayudarnos a cubrir todos los tipos de pruebas que necesitamos y para ayudarnos a asegurarnos de tener los recursos adecuados para tener éxito en cada tipo. En este capítulo, analizamos las pruebas en el primer cuadrante, las pruebas tecnológicas que respaldan al equipo y las herramientas para respaldar estas pruebas. Las actividades en este cuadrante forman el núcleo del desarrollo ágil.

UNA BASE DE PRUEBAS ÁGIL

Primero analizamos el Cuadrante 1 porque las pruebas tecnológicas que respaldan el equipo forma la base del desarrollo y las pruebas ágiles. Ver Figura 7-1 para recordar los cuadrantes de pruebas ágiles con este cuadrante resaltado. El cuadrante 1 es mucho más que pruebas. Las pruebas unitarias y de componentes de las que hablamos en el Cuadrante 1 no son las primeras pruebas escritas para cada historia, sino ayudan a guiar el diseño y el desarrollo. Sin una base de pruebas

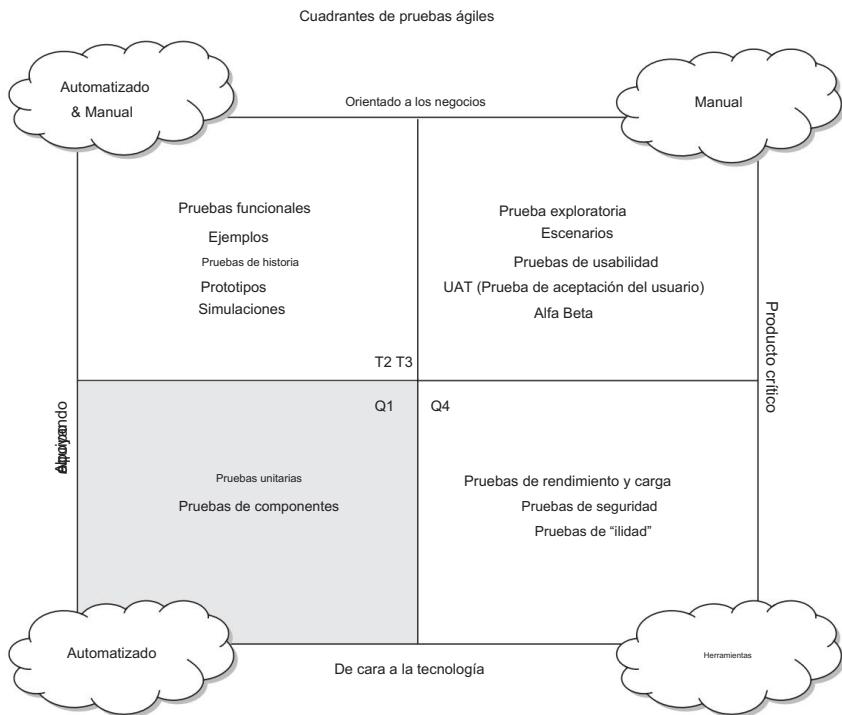


Figura 7-1 Los cuadrantes de pruebas ágiles, destacando el cuadrante 1

diseño, pruebas automatizadas de unidades y componentes y un proceso de integración continua para ejecutar las pruebas, es difícil entregar valor de manera oportuna. Todas las pruebas en los otros cuadrantes no pueden compensar las deficiencias en este.

Hablaremos de los otros cuadrantes en los próximos capítulos y explicaremos cómo encajan todos ellos.

Los equipos necesitan las herramientas y los procesos adecuados para crear y ejecutar pruebas tecnológicas que guíen el desarrollo. Daremos algunos ejemplos de los tipos de herramientas necesarias en la última sección de este capítulo.

El propósito de las pruebas del cuadrante 1

Las pruebas unitarias y las pruebas de componentes garantizan la calidad al ayudar al programador a comprender exactamente qué debe hacer el código y al brindar orientación en

el diseño correcto. Ayudan al equipo a centrarse en la historia que se está contando y a adoptar el enfoque más sencillo que funcione. Las pruebas unitarias verifican la comportamiento de partes tan pequeñas como un solo objeto o método [Meszaros, 2007]. Las pruebas de componentes ayudan a solidificar el diseño general de una parte desplegable del sistema probando la interacción entre clases o métodos.

El desarrollo de pruebas unitarias puede ser una herramienta de diseño esencial cuando se utiliza TDD. Cuando una programadora ágil comienza una tarea de codificación, escribe una prueba que captura la comportamiento de un pequeño fragmento de código y luego trabaja en el código hasta que pasa la prueba. Al construir el código en pequeños incrementos de prueba-código-prueba, el programador tiene una oportunidad para pensar en la funcionalidad que el cliente necesita. Como Si surgen preguntas, ella puede preguntarle al cliente. Puede emparejarse con un probador para ayudar a garantizar que todos los aspectos de ese fragmento de código y su comunicación con otras unidades, se prueban.

El término desarrollo basado en pruebas confunde a los profesionales que no entienden que se trata más de diseño que de pruebas. El código desarrollado primero como prueba es naturalmente diseñado para la capacidad de prueba. Todas las actividades del cuadrante 1 están dirigidas a producir software con la mayor calidad interna posible.

Cuando los equipos practican TDD, minimizan la cantidad de errores que deben corregirse. ser atrapado más tarde. La mayoría de los errores a nivel de unidad se evitan escribiendo la prueba antes del código. Pensar en el diseño escribiendo la prueba unitaria significa Es más probable que el sistema cumpla con los requisitos del cliente. Cuando el posdesarrollo El tiempo de prueba se dedica a encontrar y corregir errores que podrían haber sido detectado por las pruebas del programador, no hay tiempo para encontrar los problemas graves que podría afectar negativamente al negocio. Cuantos más errores se filren en nuestro proceso de codificación, más lenta será nuestra entrega y, al final, lo que importa es la calidad. que sufrirá. Por eso las pruebas del programador en el cuadrante 1 son tan críticas. Si bien cada equipo debe adoptar prácticas que funcionen para su situación, un Es poco probable que un equipo sin estas prácticas ágiles centrales se beneficie mucho de la metodología ágil. valores y principios.

Infraestructura de apoyo

El control sólido del código fuente, la gestión de la configuración y la integración continua son esenciales para obtener valor de las pruebas del programador que guían el desarrollo. Permiten que el equipo sepa siempre exactamente qué se está probando.

La integración continua nos brinda una forma de ejecutar pruebas cada vez que se crea un código nuevo. registrado. Cuando una prueba falla, sabemos quién registró el cambio que causó la falla y esa persona puede solucionar el problema rápidamente. Continuo

La integración ahorra tiempo y motiva a cada programador a ejecutar las pruebas antes de registrar el nuevo código. Un proceso continuo de integración y construcción. entrega un paquete de código desplegable para que lo probemos.

Los proyectos ágiles que carecen de estas prácticas ágiles básicas tienden a convertirse en "minicascadas". Los ciclos de desarrollo son más cortos, pero todavía se lanza código. "sobre la pared" para los evaluadores que se quedan sin tiempo para realizar la prueba porque el código es de mala calidad. El término cascada no es necesariamente despectivo. hemos trabajado en proyectos exitosos en "cascada" donde los programadores automatizan rigurosamente las pruebas unitarias, practican la integración continua y utilizan compilaciones automatizadas para ejecutar pruebas. Estos exitosos proyectos en "cascada" también involucran a clientes y probadores durante todo el ciclo de desarrollo. Cuando codificamos sin prácticas y herramientas apropiadas, independientemente de cómo llamemos al proceso, no vamos a lograrlo. para entregar código de alta calidad de manera oportuna.

¿POR QUÉ ESCRIBIR Y EJECUTAR ESTAS PRUEBAS?

No vamos a entrar en detalles aquí sobre cómo hacer TDD o las mejores formas de hacerlo. escribir pruebas unitarias y de componentes. Hay varios libros excelentes sobre esos temas. Nuestro objetivo es explicar por qué estas actividades son importantes para los evaluadores ágiles. Exploraremos algunas razones para utilizar pruebas tecnológicas que respalden al equipo.

Vamos a ir más rápido y hacer más

La velocidad nunca debería ser el objetivo final de un equipo de desarrollo ágil. Tratando de hacer las cosas rápido y cumplir plazos ajustados sin pensar en la calidad nos hace tomar atajos y volver a viejos y malos hábitos. Si tomamos atajos, acumular más deuda técnica y probablemente no cumplir con el plazo de todos modos. Afortunadamente, sin embargo, la velocidad es un efecto secundario a largo plazo de producir código con la mayor calidad interna posible. Notificación de compilaciones continuas que ejecutan pruebas unitarias el equipo de fallas a los pocos minutos del registro del problema, y el El error se puede encontrar y solucionar rápidamente. Una red de seguridad de unidad automatizada y Las pruebas de integración de código permiten a los programadores refactorizar con frecuencia. Este mantiene el código en un estándar razonable de mantenibilidad y entrega el mejor valor por el tiempo invertido. La deuda técnica se mantiene lo más baja posible.

Si ha trabajado como tester en un proyecto donde se descuidaron las pruebas unitarias, Sepa lo fácil que es dedicar todo su tiempo a encontrar defectos a nivel de unidad. Tú Puede encontrar tantos errores mientras prueba el "camino feliz" que nunca encontró Es hora de probar escenarios más complejos y casos extremos. La fecha límite de publicación es retrasado a medida que el ciclo de "buscar y arreglar" se prolonga, o las pruebas simplemente se detienen y se impone un producto con errores a clientes desprevenidos.

Nuestros años en equipos ágiles han sido utópicos en contraste con este escenario.

Impulsar las prácticas de codificación con pruebas significa que los programadores probablemente

Entendí razonablemente bien los requisitos de la historia. Han hablado extensamente con los clientes y evaluadores para aclarar los comportamientos deseados. Todo

las partes entienden los cambios que se están realizando. Cuando el equipo haya completado todas las tarjetas de tareas para codificar una historia, o una pequeña porción comprobable de una,

la característica ha sido bien cubierta por pruebas de unidades y componentes. Generalmente el

Los programadores se han asegurado de que al menos un camino a través de la historia funcione. para terminar.

Esto significa que nosotros, como evaluadores, perdemos poco tiempo buscando errores de bajo nivel. Eran probable que intentemos escenarios en los que los programadores no habían pensado y gastemos nuestro tiempo en funcionalidades empresariales de nivel superior. El código bien diseñado suele ser sólido y comprobable. Si encontramos un defecto, se lo mostramos al programador, quien escribe una prueba unitaria para reproducir el error y luego solucionarlo rápidamente. En realidad tenemos tiempo centrarse en las pruebas exploratorias y otros tipos de pruebas en profundidad para dar la Codifique un buen entrenamiento y aprenda más sobre cómo debería funcionar. A menudo, el Los únicos "errores" que encontramos son requisitos que todos en nuestro equipo omitieron o incomprendido. Incluso éstos se encuentran rápidamente si el cliente está involucrado y Tiene demostraciones periódicas y oportunidades de prueba. Después de que un equipo de desarrollo domina TDD, el enfoque de mejora pasa de la prevención de errores a calcular encontrar mejores formas de obtener y capturar requisitos antes de codificar.

Desarrollo basado en pruebas versus desarrollo basado en pruebas

Gerard Meszaros [Meszaros 2007, págs. 813–814] ofrece la siguiente descripción de en qué se diferencia el desarrollo basado en pruebas del desarrollo basado en pruebas:

"A diferencia del desarrollo basado en pruebas, el desarrollo basado en pruebas simplemente dice que las pruebas se escriben antes que el código de producción; no implica que el código de producción esté diseñado para funcionar una prueba a la vez (diseño emergente). El desarrollo de prueba primero se puede aplicar a nivel de prueba unitaria o de prueba del cliente, dependiendo de qué pruebas hayamos elegido automatizar".

Erik Bos [2008] observa que el desarrollo basado en la prueba implica tanto programación y diseño de prueba primero, pero hay una diferencia sutil:

"Con el diseño de prueba primero, el diseño sigue a las pruebas, mientras que se puede realizar una programación de prueba primero de un diseño que primero se escribe en una pizarra. En proyectos más grandes, tendemos a diseñar más a través de discusiones en la pizarra; El equipo analiza la arquitectura alrededor de una pizarra y codifica primero la prueba basándose en este diseño. En proyectos más pequeños, practicamos el diseño basado en pruebas".

Hay varias filosofías diferentes sobre cuándo escribir pruebas y para qué que propósito. Depende de cada equipo acordar el enfoque que le ayude lograr sus objetivos de calidad, aunque existe un acuerdo común en la comunidad ágil de que TDD definitivamente ayuda a un equipo a lograr software de mejor calidad. Esta es una forma importante en que las pruebas del programador apoyan al equipo. vamos Mira algunas formas más.

Facilitar el trabajo de los evaluadores

Las prácticas centrales relacionadas con las pruebas de programador generan muchas actividades de prueba. más fácil de lograr. Los programadores trabajan en sus propios entornos sandbox, donde Puede probar código nuevo sin afectar el trabajo de nadie más. no se registran código hasta que haya pasado un conjunto de pruebas de regresión en su entorno de pruebas.

El equipo piensa en los entornos de prueba y qué utilizar para los datos de prueba. Unidad Las pruebas generalmente funcionan con objetos falsos o simulados en lugar de bases de datos reales para velocidad, pero los programadores aún necesitan realizar pruebas con datos realistas. Los probadores pueden Ayúdelos a identificar buenos datos de prueba. Si las pruebas unitarias representan datos de la vida real, Se encontrarán menos problemas más adelante.

La historia de Lisa

He aquí un pequeño ejemplo. Cuando mi equipo actual adoptó por primera vez el desarrollo ágil, no teníamos pruebas automatizadas. No teníamos forma de producir un paquete de código desplegable y no teníamos entornos de prueba ni bases de datos de prueba rudimentarios. Tampoco tenía ningún medio para producir una construcción yo mismo. Decidimos comenzar a escribir código de prueba primero y nos comprometimos a automatizar las pruebas en todos los niveles cuando fuera apropiado, pero primero necesitábamos algo de infraestructura.

Nuestra primera prioridad fue implementar un proceso de construcción continuo, que se realizó en un par de días. Cada compilación envió un correo electrónico con una lista de archivos registrados y comentarios sobre las actualizaciones. Ahora podía elegir qué compilación implementar y probar. La siguiente prioridad era proporcionar entornos de prueba independientes para que las pruebas realizadas por una persona no interfirieran con otras pruebas. El nuevo experto en bases de datos creó nuevos esquemas para satisfacer las necesidades de prueba y una base de datos "semilla" de datos canónicos similares a los de producción. Estos esquemas podrían actualizarse rápidamente según demanda con un conjunto de datos limpio. Cada miembro del equipo, incluyéndome a mí, obtuvo un entorno de prueba único e independiente.

Incluso antes de que el equipo dominara TDD, la infraestructura adoptada ya estaba implementada para respaldar la ejecución de pruebas. Esta infraestructura permitió al equipo comenzar a realizar pruebas de manera mucho más efectiva. Otro aspecto de intentar automatizar las pruebas fue lidiar con una aplicación heredada que era difícil de probar. Las decisiones que se tomaron para habilitar TDD también ayudaron con las pruebas de cara al cliente. Decidimos comenzar a reescribir el sistema en una nueva arquitectura que facilitara las pruebas y la automatización de pruebas, no solo a nivel de unidad sino en todos los niveles.

—Lisa

Escribir pruebas y escribir código con esas pruebas en mente significa que los programadores siempre estamos haciendo conscientemente que el código sea comprobable. Todas estas buenas cualidades relacionadas con la infraestructura se extienden a las pruebas de cara al negocio y a las pruebas que critican la producto. Todo el equipo piensa continuamente en formas de mejorar el diseño. y facilitar las pruebas.

Diseñar teniendo en cuenta las pruebas

Una ventaja de impulsar el desarrollo con pruebas es que el código se escribe con la intención expresa de hacer pasar las pruebas. El equipo tiene que pensar, ¿verdad? desde el principio, sobre cómo ejecutará y automatizará las pruebas para cada historia que codifica. El desarrollo basado en pruebas significa que los programadores escribirán cada prueba antes de escribir el código para aprobarlo.

Escribir “código comprobable” es un concepto simple, pero no es una tarea fácil, especialmente si está trabajando en código antiguo que no tiene pruebas automatizadas y no está diseñado para ser comprobable. Los sistemas heredados suelen tener lógica empresarial, E/S, base de datos, y capas de interfaz de usuario entrelazadas. No existe una manera fácil de conectarse para automatizar una prueba debajo de la GUI o a nivel de unidad.

Un enfoque común en el diseño de una arquitectura comprobable es separar los diferentes capas que realizan diferentes funciones en la aplicación. Idealmente, querrá acceder a cada capa directamente con un dispositivo de prueba y probar algoritmos con diferentes entradas. Para hacer esto, aíslle la lógica de negocios en sus propia capa, utilizando objetos falsos en lugar de intentar acceder a otras aplicaciones o la base de datos real. Si la capa de presentación se puede separar de la lógica empresarial subyacente y el acceso a la base de datos, puede probar rápidamente la validación de las entradas. sin probar la lógica subyacente.

Arquitecturas en capas y capacidad de prueba

El equipo de Lisa adoptó el enfoque de la “aplicación estranguladora” para crear un sistema comprobable donde las pruebas pudieran usarse para impulsar la codificación. Mike Thomas, el arquitecto principal del equipo, explica cómo su nueva arquitectura en capas permitió un diseño comprobable.

Una arquitectura en capas divide una base de código en porciones horizontales que contienen funcionalidades similares, a menudo relacionadas con una tecnología. Los sectores del nivel más alto son los más específicos y dependen de los sectores inferiores, que son más generales. Por ejemplo, muchas bases de código en capas tienen porciones como las siguientes: interfaz de usuario, lógica empresarial y acceso a datos.

Las capas horizontales son sólo una forma de organizar una base de código: otra son las porciones orientadas al dominio (como la nómina o el ingreso de pedidos), que generalmente se consideran "verticales". Estos enfoques de capas se pueden combinar, por supuesto, y todos se pueden utilizar para mejorar la capacidad de prueba.

La estratificación tiene ventajas para las pruebas, pero sólo si el mecanismo para "conectar" los cortes proporciona flexibilidad. Si una base de código tiene porciones estrechamente acopladas a través de mecanismos tales como dependencias de clases concretas directas y métodos estáticos, es difícil aislar una unidad para realizar pruebas, a pesar de las capas. Esto convierte la mayoría de las pruebas automatizadas en pruebas de integración, que pueden ser complicadas y ejecutarse lentamente. En muchos casos, las pruebas sólo se pueden realizar ejecutando todo el sistema.

Compare esto con una base de código donde las capas están separadas por interfaces. Cada segmento depende únicamente de las interfaces definidas en el segmento debajo de él en lugar de clases específicas. Las dependencias en dichas interfaces son fáciles de satisfacer con dobles de prueba en el momento de la prueba: simulacros, stubs, etc. De este modo, las pruebas unitarias se simplifican porque cada unidad puede aislarse verdaderamente. Por ejemplo, la interfaz de usuario se puede probar con objetos simulados de la capa empresarial, y la capa empresarial se puede probar con objetos simulados de acceso a datos, evitando el acceso a la base de datos en vivo.

El enfoque en capas ha permitido al equipo de Lisa tener éxito en la automatización de pruebas en todos los niveles e impulsar el desarrollo con pruebas tanto orientadas a la tecnología como a los negocios.

Consulte la bibliografía para obtener más información sobre el patrón de puertos y adaptadores Alastair Cockburn.

Otro ejemplo de un enfoque de diseño comprobable es el de Alistair Cockburn. Patrón de puertos y adaptadores [Cockburn, 2005]. La intención de este patrón es "crear su aplicación para que funcione sin una interfaz de usuario ni una base de datos para que pueda ejecutarla". pruebas de regresión automatizadas contra la aplicación, funcionan cuando la base de datos deja de estar disponible y vincula las aplicaciones sin la participación del usuario". Los puertos aceptan eventos externos y un adaptador de tecnología específica lo convierte en un mensaje que la aplicación puede entender. Sucesivamente, la aplicación envía la salida a través de un puerto a un adaptador, que crea las señales que necesitan los usuarios humanos o automatizados receptores. Las aplicaciones diseñadas con este patrón pueden controlarse mediante scripts de prueba automatizados tan fácilmente como por usuarios reales.

Es más obvio cómo codificar primero la prueba en un proyecto totalmente nuevo. Los sistemas heredados, que no están cubiertos por pruebas unitarias automatizadas, presentan un gran desafío. Es difícil escribir pruebas unitarias para código que no está diseñado para la capacidad de prueba, y es difícil cambiar el código que no está protegido con pruebas unitarias. Muchos equipos tienen

La bibliografía tiene enlaces a más artículos sobre "rescue" y "extraño" gler" aborda el código heredado.

Seguimos las técnicas de "rescate de código heredado" explicadas por Michael Feathers. en Trabajar eficazmente con código heredado [Feathers, 2005]. Otros equipos, como como el de Lisa, pretenden "estrangular" su código heredado. Esta estrategia surge de Martín. La "aplicación estranguladora" de Fowler [Fowler, 2004]. Las nuevas historias se codificaron primero como prueba en una nueva arquitectura mientras el antiguo sistema aún se mantenía. Encima Con el tiempo, gran parte del sistema se ha convertido a la nueva arquitectura, con el objetivo de acabar con el antiguo sistema.

Las pruebas ágiles en un entorno tipo mainframe heredado presentan particularidades desafíos, uno de los cuales es la falta de disponibilidad de publicaciones y información sobre cómo hacerlo con éxito. COBOL, mainframes y sus Los de este tipo todavía se utilizan ampliamente. Deje que los principios y valores ágiles guíen a su equipo busca formas de habilitar pruebas automatizadas en su aplicación. tu podrías hay que adaptar algunas técnicas; por ejemplo, tal vez no pueda escribir código primero como prueba, pero puede probarlo poco después de escribir el código. Cuando sea el problema que debe resolver el equipo, y no sólo el problema de los evaluadores, encontrará una manera de escribir pruebas.

Prueba de sistemas heredados

John Voris, desarrollador de Crown Cork and Seal, trabaja en el lenguaje RPG, primo de COBOL, que se ejecuta en el sistema operativo anteriormente conocido como AS 400 y ahora conocido como System i. A John se le asignó la tarea de fusionar código nuevo con una base de código de proveedor. Aplicó los principios de las prácticas de codificación Agile, Lean y recomendadas por IBM para crear un enfoque que llama "ADEPT" para "Pantallas AS400 para pruebas y prototipos externos". Si bien no está codificando primero la prueba, está probando "Minutos después". Así es como resumió su enfoque:

Para obtener más información sobre el desarrollo de Presenter First, consulte la bibliografía.

- Escribir módulos pequeños de propósito único (no programas monolíticos) y refactorizar programas existentes en módulos. Utilice un enfoque de desarrollo de Presenter First (similar al patrón Model View Presenter o Model View Controller).
- Definir interfaces de parámetros para el arnés de prueba según los formatos y campos de pantalla. El único inconveniente aquí es que los números se definen como decimales zonificados en lugar de hexadecimales empaquetados, pero esto se compensa con la ganancia de productividad.
- "Minutos después" de codificar cada módulo de producción, cree un programa de prueba utilizando el formato de pantalla para realizar pruebas a través de la interfaz de usuario. La interfaz UI para la prueba se crea antes del programa de producción, porque la interfaz de prueba UI es la interfaz a la que se hace referencia para el módulo de producción. El ímpetu para ejecutar una prueba es grande para el programador, porque la mayor parte de la codificación para la prueba ya está hecha.

Para obtener más

información sobre RPGUnit,
consulte www.RPGUnit.org.

- Utilice conjuntos de datos de prueba estándar, que son datos de prueba canónicos e inmutables, para conducir las pruebas.
- Este enfoque, en el que los programas de prueba se generan casi automáticamente, se presta a la automatización con una herramienta de grabación/reproducción que capturaría entradas y salidas de datos, con pruebas ejecutadas en una compilación continua, utilizando RPGUnit.

Su equipo puede encontrar un enfoque para diseñar para la capacidad de prueba que funcione para tú. El secreto es el compromiso de todo el equipo con las pruebas y la calidad. Cuando un equipo trabaja constantemente para escribir pruebas y hacerlas pasar, encuentra una manera para hacerlo. Los equipos deben tomarse el tiempo para considerar cómo pueden crear una arquitectura que haga que las pruebas automatizadas sean fáciles de crear, económicas de mantener y duraderas. No tenga miedo de revisar la arquitectura si está automatizada.

Las pruebas no devuelven suficiente valor por la inversión realizada en ellas.

Comentarios oportunos

El mayor valor de las pruebas unitarias está en la velocidad de su retroalimentación. En nuestra opinión, un proceso continuo de integración y construcción que ejecute las pruebas unitarias debería terminar en diez minutos. Si cada programador verifica el código en varios veces al día, un proceso de compilación y prueba más largo hará que los cambios comiencen a acumularse. Como evaluador, puede resultar frustrante tener que esperar mucho tiempo para obtener nuevas funcionalidad o una corrección de errores. Si hay un error de compilación o una falla en la prueba unitaria, el retraso empeora aún más, ¡especialmente si ya casi es hora de irse a casa!

Un proceso de compilación y prueba que ejecute pruebas por encima del nivel unitario, como pruebas de API funcionales o pruebas de GUI, llevará más tiempo. Tener al menos una construcción proceso que se ejecuta rápidamente y un segundo que ejecuta las pruebas más lentas. Allí Debe haber al menos una "compilación" diaria que ejecute todas las pruebas funcionales más lentas. Sin embargo, incluso eso puede resultar difícil de manejar. Cuando una prueba falla y el problema es solucionado, ¿cuánto tiempo llevará estar seguro de que la compilación se realiza nuevamente?

Si su proceso de compilación y prueba lleva demasiado tiempo, solicite a su equipo que analice la causa de la desaceleración y tomar medidas para acelerar la construcción. Aquí están algunos ejemplos.

El acceso a la base de datos suele consumir mucho tiempo, así que considere utilizar objetos falsos, siempre que sea posible, para reemplazar la base de datos, especialmente a nivel de unidad.

Traslade las pruebas de integración y acceso a bases de datos de mayor duración al proceso secundario de compilación y prueba.

Vea si las pruebas se pueden ejecutar en paralelo para que finalicen más rápido.

Ejecute las pruebas mínimas necesarias para realizar pruebas de regresión en su sistema.
Distribuya tareas entre varias máquinas de construcción.
Actualice el hardware y el software que ejecutan la compilación.
Encuentre el área que requiere más tiempo y tome medidas graduales para acelerarla.

La historia de Lisa

Al principio de la evolución ágil de mi equipo actual, teníamos pocas pruebas unitarias, por lo que incluimos algunas pruebas de humo de GUI en nuestra compilación continua, que comenzaba en cada registro del sistema de control de código fuente. Cuando tuvimos suficientes pruebas unitarias para sentirnos bien al saber cuándo se había roto el código, movimos las pruebas de GUI y las pruebas funcionales de FitNesse a un proceso de compilación y prueba separado que se ejecutaba por la noche, en la misma máquina que nuestra compilación continua.

Nuestra construcción continua comenzó tomando menos de 10 minutos, pero pronto tardó más de 15 minutos en completarse. Escribimos tarjetas de tareas para diagnosticar y solucionar el problema. Las pruebas unitarias que los programadores habían escrito desde el principio no eran bien diseñado, porque nadie estaba seguro de cuál era la mejor manera de escribir pruebas unitarias. Se presupuestó tiempo para refactorizar las pruebas unitarias, utilizar objetos de acceso a datos simulados en lugar de la base de datos real y rediseñar las pruebas para aumentar la velocidad. Esto llevó la construcción a unos ocho minutos. Cada vez que comenzó a aumentar, abordamos el problema refactorizando, eliminando pruebas innecesarias, actualizando el hardware y eligiendo software diferente que ayudó a que la compilación se ejecutara más rápido.

A medida que nuestras pruebas funcionales cubrieron más código, la compilación nocturna falló con más frecuencia. Debido a que la compilación nocturna se ejecutó en la misma máquina que la compilación continua, la única forma de verificar que la compilación era "verde" nuevamente era detener la compilación en curso, lo que eliminó nuestra rápida retroalimentación. Esto empezó a hacer perder el tiempo a todos. Compramos y configuramos otra máquina de construcción para la construcción más larga, que ahora también funciona de forma continua. Esto fue mucho menos costoso que dedicar tanto tiempo a mantener dos compilaciones ejecutándose en la misma máquina, y ahora también recibimos comentarios rápidos de nuestras pruebas funcionales.

—Lisa

Vaya, múltiples procesos continuos de compilación y prueba que brindan retroalimentación constante; a muchos evaluadores les parece un sueño. Se detectarán errores de regresión temprano, cuando es más barato arreglarlos. Esta es una gran razón para escribir pruebas orientadas a la tecnología. ¿Podemos dejarnos llevar por ellos? Miremos el línea entre las pruebas orientadas a la tecnología y las pruebas orientadas a los negocios.

¿DÓNDE SE TERMINAN LAS PRUEBAS DE TECNOLOGÍA ?

A menudo escuchamos a la gente preocuparse de que las pruebas cara al cliente se superpongan tanto con las pruebas tecnológicas en las que el equipo perderá el tiempo. Lo sabemos

Las pruebas orientadas al negocio pueden cubrir un poco el mismo terreno que las pruebas de integración de código o unitarias, pero tienen propósitos tan diferentes que el desperdicio no es una preocupación.

Por ejemplo, tenemos una historia para calcular un calendario de amortización de préstamos y mostrárselo a un usuario que esté en el proceso de solicitar un préstamo. Una prueba unitaria para Esta historia probablemente pondría a prueba argumentos ilegales, como un pago anual. frecuencia si el negocio no lo permite. Podría haber una prueba unitaria para calcular la fecha prevista de inicio de pago del préstamo dada alguna definición de monto, tasa de interés, fecha de inicio y frecuencia. Las pruebas a nivel unitario podrían cubrir diferentes combinaciones de frecuencia de pago, monto, fecha de interés, plazo e inicio fecha para acreditar que el cálculo de la amortización es correcto. Ellos podría cubrir escenarios como los años bisiestos. Cuando pasan estas pruebas, el programador se siente seguro sobre el código.

Cada prueba unitaria es independiente y prueba una dimensión a la vez. Esto significa que cuando una prueba unitaria falla, el programador puede identificar el problema rápidamente y resolver el problema con la misma rapidez. Las pruebas empresariales rara vez cubren una sola dimensión, porque se abordan desde un punto de vista empresarial.

Las pruebas de cara al negocio para esta historia definirían más detalles para el reglas de negocio, la presentación en la interfaz de usuario y el manejo de errores. Verificarían que los detalles de pago, como el capital y el interés aplicado, se muestren correctamente en la interfaz de usuario. Probarían validaciones para cada campo en la interfaz de usuario y especificar el manejo de errores para situaciones como como saldo insuficiente o inelegibilidad. Podrían probar un escenario en el que un administrador procesa dos pagos de préstamos el mismo día, lo que podría ser más difícil de simular a nivel de unidad.

Capítulo 13, "¿Por qué Queremos automatizar las pruebas y lo que nos detiene", habla más sobre el ROI de los diferentes tipos de pruebas.

Las pruebas empresariales cubren escenarios de usuario más complejos y verifican que el usuario final tendrá una buena experiencia. Empuje las pruebas a niveles más bajos siempre que posible; si identifica un caso de prueba que se puede automatizar a nivel de unidad, eso casi siempre supone un mejor retorno de la inversión.

Si están involucradas varias áreas o capas de la aplicación, es posible que no sea posible automatizar a nivel de unidad. Tanto el nivel tecnológico como el empresarial pueden tener pruebas alrededor de la fecha del primer pago del préstamo, pero lo comprueban por diferentes motivos. La prueba unitaria comprobaría el cálculo de la fecha, y la prueba de cara al negocio verificaría que se muestra correctamente en el informe de préstamo del prestatario.

Aprender a escribir pruebas del cuadrante 1 es difícil. Muchos equipos que hacen la transición al desarrollo ágil comienzan sin pruebas unitarias automatizadas, ni siquiera un

proceso continuo de integración y construcción. En la siguiente sección, sugerimos acciones que los evaluadores ágiles pueden tomar si sus equipos no abordan las pruebas del Cuadrante 1.

¿Y SI EL EQUIPO NO HACE ESTAS PRUEBAS?

Muchas organizaciones han decidido probar el desarrollo ágil, o al menos han declarado esa intención, sin entender cómo hacer una transición exitosa.

Cuando desempeñamos el rol de tester, ¿qué podemos hacer para ayudar al equipo de desarrollo? implementar TDD, integración continua y otras prácticas que son clave para ¿Desarrollo exitoso?

Nuestra experiencia a lo largo de los años ha sido que si no somos programadores, no necesariamente tenemos mucha credibilidad cuando instamos a los programadores a adoptar prácticas como TDD.

Si pudiéramos sentarnos y mostrarles cómo codificar primero la prueba, eso sería convincente, pero muchos de nosotros, los evaluadores, no lo sabemos. tener ese tipo de experiencia. También hemos descubierto que evangelizar no funciona. No es tan difícil convencer a alguien conceptualmente de que TDD es una buena idea. Es mucho más complicado ayudarlos a conseguir tracción codificando primero la prueba.

¿Qué pueden hacer los evaluadores?

Si eres un evaluador en un equipo llamado "ágil" que ni siquiera está automatizando una unidad pruebas o producir compilaciones continuas o, como mínimo, realizar compilaciones en un diariamente, te frustrarás bastante rápido. No te rindas; mantener Lluvia de ideas sobre una manera de impulsar una transición positiva. Intente aprovechar el tiempo social u otra actividad relajante para tomarse un momento agradable y ver qué novedades hay. ideas que puede generar para que todos los miembros del equipo se unan.

Una trampa que se debe evitar es que los evaluadores escriban las pruebas unitarias. Debido a que TDD es más bien una actividad de diseño, es esencial que la persona que escribe el código también escriba las pruebas, antes de escribir el código. Los programadores también necesitan la inmediata retroalimentación que brindan las pruebas unitarias automatizadas. Las pruebas unitarias escritas por otra persona después de escribir el código aún pueden proteger contra defectos de regresión, pero no no tendrá los beneficios más valiosos de las pruebas escritas por el programador.

La historia de Lisa

Siempre que he querido efectuar un cambio, he recurrido a los patrones en Valiente Cambiar por Mary Lynn Manns y Linda Rising [2004]. Después de trabajar en dos equipos de XP, me uní a un equipo que profesaba el deseo de volverse ágil pero que no avanzaba hacia prácticas de desarrollo sólidas. Encontré varios patrones en Valiente Cambiar para intentar mover al equipo hacia prácticas ágiles.

"Pedir ayuda" fue un patrón que me ayudó. Este patrón dice, en parte: "Desde la tarea de introducir una nueva idea en una organización es un trabajo grande, busque personas y recursos que le ayuden en sus esfuerzos" [Manns y Rising, 2004]. Cuando quise que mi equipo comenzara a usar FitNesse, identifiqué al programador que más simpatizaba con mi causa y le pedí que trabajara conmigo para escribir pruebas de FitNesse para la historia en la que estaba trabajando. Les contó a los demás programadores los beneficios que obtuvo de las pruebas de FitNesse, lo que los animó a probarlas también. La mayoría de la gente quiere ayudar, y ágil se trata de que el equipo trabaje en conjunto, por lo que no hay razón para hacerlo solo.

La "bolsa marrón" es otro patrón de cambio que mis equipos han aprovechado. Por ejemplo, mi equipo actual celebró varias sesiones en las que escribieron pruebas unitarias juntos. "Gurú de tu lado" es un patrón productivo en el que solicitas la ayuda de un miembro del equipo muy respetado que podría entender lo que estás tratando de lograr. Un equipo anterior en el que estuve no estaba motivado para escribir pruebas unitarias. El programador más experimentado del equipo estuvo de acuerdo conmigo en que el desarrollo basado en pruebas era una buena idea y fue un ejemplo para el resto del equipo.

Creemos que descubrirá que siempre hay alguien en un equipo ágil que simpatiza con su causa. Consiga el apoyo de esa persona, especialmente si el equipo la percibe como un gurú de alto nivel.

—Lisa

Como evaluador en un equipo ágil, hay muchas cosas que puedes hacer para actuar como agente de cambio, pero su impacto potencial es limitado. En algunos casos, un fuerte apoyo administrativo es la clave para impulsar al equipo a participar en las actividades del Cuadrante 1.

¿Qué pueden hacer los gerentes?

Si administra un equipo de desarrollo, puede hacer mucho para fomentar el desarrollo basado en pruebas y la automatización de pruebas unitarias. Trabajar con el propietario del producto hacer de la calidad su objetivo y comunicar los criterios de calidad al equipo. Anime a los programadores a tomarse el tiempo para hacer su mejor trabajo en lugar de preocuparse por cumplir una fecha límite. Si una fecha de entrega está en peligro, presione para reducir el alcance, no la calidad. Su trabajo es explicar a los gerentes de negocios cómo hacer de la calidad una prioridad garantizará que obtengan un valor comercial óptimo.

Dé tiempo al equipo para aprender y brinde capacitación práctica y experta. Traer un coach experimentado en desarrollo ágil o contratar a alguien con experiencia en utilizando estas prácticas quién puede transferir esas habilidades al resto del equipo. Dedique tiempo a una refactorización importante, a una lluvia de ideas sobre el mejor enfoque para escribir pruebas de integración de código y unidades, y a evaluar, instalar y actualizar herramientas. Los administradores de pruebas deberían trabajar con el desarrollo.

gerentes para fomentar prácticas que mejoren la capacidad de prueba y permitan a los evaluadores escribir pruebas ejecutables. Los administradores de pruebas también pueden asegurarse de que los evaluadores tengan tiempo para Aprenda a utilizar las herramientas y marcos de automatización que el equipo decida. para implementar.

Es un problema de equipo

Más información sobre retrospectivas y mejoras de procesos en el Capítulo 19, "Conclusión de la iteración".

Si bien puedes encontrar formas de ser un agente de cambio eficaz, lo mejor que puedes hacer Se trata de involucrar a todo el equipo en la resolución de los problemas. Si aún no lo estás haciendo retrospectivas después de cada iteración, propongo probar esta práctica o alguna otro tipo de mejora de procesos. En la retrospectiva, plantee cuestiones que sean dificultando la entrega exitosa. Por ejemplo, "No estamos terminando las tareas de prueba antes del final de la iteración" es un problema que todo el equipo debe abordar. Si Una razón para no terminar es la gran cantidad de errores a nivel de unidad, sugiere experimentar con TDD, pero permitir a los programadores proponer sus propias formas para enfrentar el problema. Anime al equipo a probar un nuevo enfoque durante algunos iteraciones y ver cómo funciona.

Las pruebas tecnológicas que respaldan el proceso de desarrollo del equipo son una base importante para todas las pruebas que deben realizarse. si el equipo no está haciendo un trabajo adecuado con las pruebas en este cuadrante, los otros tipos de Las pruebas serán mucho más difíciles. Esto no significa que no puedas obtener valor de los otros cuadrantes por sí solos, solo significa que será más difícil hacerlo entonces porque el código del equipo carecerá de calidad interna y todo tardará más extenso.

Las pruebas tecnológicas no se pueden realizar sin las herramientas y la infraestructura adecuadas. En la siguiente sección, veremos ejemplos de los tipos de herramientas que un equipo debe ser eficaz con las pruebas del cuadrante 1.

KIT DE HERRAMIENTAS

No existe una herramienta mágica que asegure el éxito. Sin embargo, las herramientas pueden ayudar La buena gente hace su mejor trabajo. Es fundamental crear la infraestructura adecuada para respaldar las pruebas tecnológicas. Hay una gran selección de excelentes herramientas disponibles y mejoran todo el tiempo. Tu equipo debe encontrar las herramientas que funcionan mejor para su situación.

Control de código fuente

El control del código fuente también se conoce con otros nombres, como control de versiones o control de revisión. Ciertamente no es nuevo ni exclusivo del desarrollo ágil, pero

ningún equipo de desarrollo de software puede tener éxito sin él. Por eso lo estamos discutiendo aquí.

Sin control del código fuente, nunca estarás seguro de lo que estás

pruebas. ¿El programador cambió solo el módulo que dijo que cambió, o

¿Olvidó los cambios que hizo en otros módulos? No se pueden deshacer cambios no deseados o erróneos sin algún tipo de sistema de control de versiones.

El control del código fuente evita que diferentes programadores caminen entre sí los cambios de otros a los mismos módulos. Sin versiones, no puedes estar seguro qué código lanzar a producción.

Patrones de gestión de configuración de software: trabajo en equipo eficaz, práctico

Integrations [2003], de Stephen Berczuk y Brad Appleton, es un buen recurso

utilizar para aprender cómo y por qué utilizar el control del código fuente. Control de código fuente es esencial para cualquier estilo de desarrollo de software.

Utilice también el control del código fuente para scripts de prueba automatizados. es importante atar las pruebas automatizadas con la versión de código correspondiente en la que probaron caso de que necesite volver a ejecutar las pruebas con esa versión en el futuro. cuando etiquetas o etiquetar una compilación, asegúrese de etiquetar o etiquetar el código de prueba también, incluso si no es así. ser lanzado a producción.

Los equipos pueden organizar su jerarquía de código para proporcionar un repositorio para el código de producción, las pruebas unitarias correspondientes y los scripts de prueba de nivel superior. Haciendo esto podría requerir cierta lluvia de ideas y experimentación para obtener el estructura correcta.

Hay muchas opciones estupendas para elegir. Sistemas de código abierto como CVS y Subversion (SVN) son fáciles de implementar, se integran con un proceso de construcción continuo e IDE y son robustos. Las herramientas de proveedores como IBM Rational ClearCase y Perforce podrían agregar características que compensen la aumento de gastos generales que a menudo conllevan.

El control del código fuente está estrechamente integrado con los entornos de desarrollo.

Veamos algunos IDE utilizados por equipos ágiles.

IDE

Un buen IDE (entorno de desarrollo integrado) puede resultar útil para los programadores y evaluadores de un equipo ágil. El IDE se integra con la fuente.

Sistema de control de código para ayudar a prevenir problemas con versiones y cambios.

caminando unos sobre otros. Los editores dentro de un IDE son específicos del lenguaje de programación y señalan errores incluso mientras se escribe el código. Más importante,

Los IDE brindan soporte para la refactorización.

Los programadores que utilizan un IDE tienden a tener fuertes preferencias personales. Sin embargo, a veces una organización decreta que todos los programadores deben utilizar un IDE específico. Esto podría deberse a la licencia o podría tener la intención de fomentar la programación en pares abiertos. Es más fácil emparejarse con otro programador si la otra persona usa el mismo IDE, pero generalmente no es esencial para el mismo que se va a utilizar. La mayoría de las herramientas funcionan de manera similar, por lo que no es difícil cambiarlas de un IDE a otro para satisfacer nuevas necesidades o aprovechar nuevas características. Algunos incondicionales todavía prefieren utilizar tecnología probada y verdadera como vi, vim o emacs con archivos make en lugar de un IDE.

Los IDE de código abierto como Eclipse y NetBeans son ampliamente utilizados por los ágiles. equipos, junto con sistemas propietarios como Visual Studio e IntelliJ IDEA. Los IDE tienen complementos para admitir diferentes lenguajes y herramientas. Ellos funcionan tan bien con scripts de prueba como con código de producción.

La historia de Lisa

En mi equipo actual, algunos programadores usaban IntelliJ IDEA, mientras que otros usaban Eclipse. En casos excepcionales, las diferencias ambientales causaron problemas, como que las pruebas pasaran en el IDE pero no la compilación completa, o que los registros a través del IDE causaran estragos en el sistema de control del código fuente. Sin embargo, en general, el uso de diferentes IDE no causó problemas. Curiosamente, con el tiempo la mayoría de los usuarios de Eclipse cambiaron. El emparejamiento con los usuarios de IntelliJ les llevó a preferirlo.

Utilizo Eclipse para trabajar con los scripts de prueba automatizados, así como para investigar problemas con el código de producción. El complemento Ruby nos ayuda con nuestros scripts Ruby y Watir, y el editor XML nos ayuda con nuestros scripts Canoo WebTest. Podemos ejecutar pruebas unitarias y compilaciones a través del IDE. Los programadores del equipo me ayudaron a configurar y comenzar a usar Eclipse, y eso me ahorró una gran cantidad de tiempo. Mantener las pruebas automatizadas es mucho más fácil y la vista de "sincronización" del IDE me ayuda a recordar que debo registrar todos los módulos que he cambiado.

Las herramientas de prueba están comenzando a aparecer con sus propios IDE o complementos para funcionar con IDE existentes como Eclipse. Aproveche estas poderosas herramientas que ahorran tiempo y promueven la calidad.

—Lisa

Probadores que no están automatizando pruebas a través de un IDE, pero que quieren poder para ver fragmentos de código modificados, puede utilizar herramientas como FishEye que permiten los evaluadores obtengan acceso al código a través de la compilación automatizada.

Al momento de escribir este artículo, los IDE han agregado soporte para lenguajes dinámicos como Ruby, Groovy y Python. Los programadores que utilizan lenguajes dinámicos pueden preferir herramientas más livianas, pero aún necesitan buenas herramientas que admitan buenas Prácticas de codificación, como TDD y refactorización.

Independientemente del entorno de desarrollo y las herramientas que se utilicen, ágil

Los equipos necesitan un marco que integre los cambios de código de diferentes programadores, ejecute las pruebas unitarias para verificar que no se hayan producido errores de regresión y proporcione el código en un formato desplegable.

Herramientas de construcción

Su equipo necesita alguna forma de construir el software y crear un jar desplegable, war u otro tipo de archivo. Esto se puede hacer con herramientas basadas en shell como hacer, pero esas herramientas tienen limitaciones, como las plataformas donde trabajar. Los equipos ágiles que conocemos utilizan herramientas como ant, Nant y Maven para construir sus proyectos. Estas herramientas no sólo gestionan la construcción sino que también proporcionan formas sencillas de informar y documentar los resultados de la construcción, y se integran fácilmente con construir herramientas de automatización y prueba. También se integran con IDE.

Construir herramientas de automatización

La integración continua es una práctica central para los equipos ágiles. Necesitas una manera de no solo construir el proyecto sino también ejecutar pruebas automatizadas en cada compilación para hacer Seguro que no se rompió nada. Una compilación totalmente automatizada y reproducible que se ejecuta muchas veces al día es un factor clave de éxito para los equipos ágiles. Construcción automatizada Las herramientas proporcionan funciones como notificación por correo electrónico de los resultados de la compilación y se integran con herramientas de control de código fuente y de compilación.

Las herramientas comúnmente utilizadas al momento de escribir este libro incluyen el código abierto herramientas CruiseControl, CruiseControl.net, CruiseControl.rb y Hudson.

Otras herramientas de código abierto y patentadas disponibles en el momento de la publicación son AnthillPro, Bamboo, BuildBeat, CI Factory, Team City y Pulse, solo por nombrar algunos.

Sin un proceso de compilación automatizado, será difícil implementar código tanto para probar como para publicar. Gestión de edificios y automatización de edificios

Las herramientas son fáciles de implementar y absolutamente necesarias para una metodología ágil exitosa. proyectos. Asegúrese de iniciar su proceso de construcción temprano, incluso antes de empezar a codificar. Experimente con diferentes herramientas cuando descubra que necesita más características que las que proporciona su proceso actual.

Herramientas de prueba unitaria

Las herramientas de prueba unitaria son específicas del idioma en el que estás codificando. "xUnidad"

Las herramientas son comúnmente utilizadas por equipos ágiles, y hay una variedad para muchos lenguajes diferentes, incluidos JUnit para Java, NUnit para .NET, Test::Unit para Perl. y Ruby, y PyUnit para Python.



Consulte el Capítulo 9.

"Kit de herramientas para

Orientado a los negocios

Pruebas que apoyan al equipo", para

obtener más información

sobre el desarrollo impulsado por el

comportamiento.

herramientas de mento.



Consulte la bibliografía para

encontrar enlaces y libros que

ayuden a su equipo a buscar

las herramientas de prueba unitaria

adecuadas.

El desarrollo basado en el comportamiento es otra variante del desarrollo basado en pruebas, detallar el comportamiento esperado para realizar pruebas con herramientas como RSpec y easyb.

El código GUI también puede y debe desarrollarse primero como prueba. Algunas herramientas para pruebas unitarias de cliente rico son TestNG, Abbot y SWTBot.

Herramientas como EasyMock y Ruby/Mock ayudan a implementar objetos simulados y resguardos de prueba, una parte integral de las pruebas unitarias bien diseñadas.

Las herramientas que utilizan los programadores para escribir pruebas orientadas a la tecnología también se pueden utilizar para pruebas de cara al negocio. Si son adecuados para ese propósito en su

El proyecto depende de las necesidades de su equipo y de sus clientes.

RESUMEN

En este capítulo, explicamos el propósito de las pruebas tecnológicas que respaldan al equipo y hablamos sobre qué necesitan los equipos para usarlas de manera efectiva.

Las pruebas tecnológicas que respaldan la programación permiten al equipo producir código de la más alta calidad posible; Forman la base para todos los demás tipos de pruebas.

Los beneficios de las pruebas de este cuadrante incluyen ir más rápido y hacer más, pero la velocidad y la cantidad nunca deben ser el objetivo final.

Los programadores escriben pruebas orientadas a la tecnología que respaldan al equipo y brindan un gran valor a los evaluadores al mejorar la calidad interna y la capacidad de prueba del sistema.

Los equipos que no implementen las prácticas básicas relacionadas con el desarrollo ágil probablemente tendrán dificultades.

Los sistemas heredados suelen presentar los mayores obstáculos para el desarrollo basado en pruebas, pero estos problemas se pueden superar con enfoques incrementales.

Si su equipo aún no realiza estas pruebas, puede ayudarlos a comenzar involucrando a otros miembros del equipo y obteniendo apoyo de la gerencia.

Puede haber cierta superposición entre las pruebas orientadas a la tecnología y las pruebas orientadas a los negocios que respaldan al equipo. Sin embargo, cuando tenga que elegir, lleve las pruebas al nivel más bajo para maximizar el retorno de la inversión.

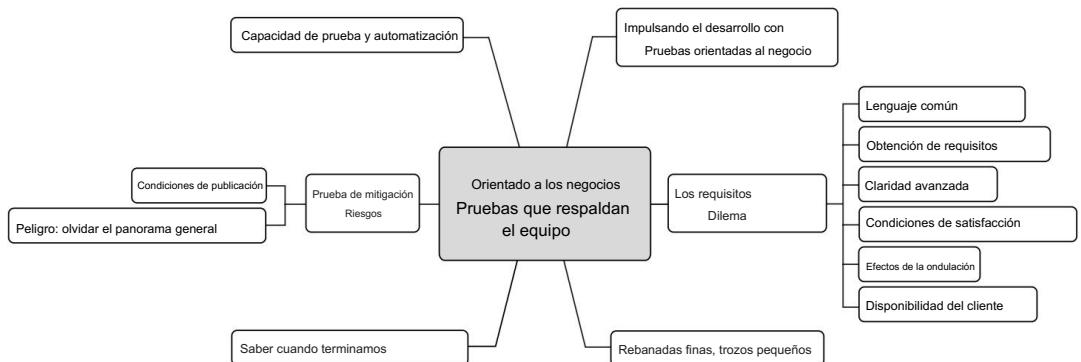
Los equipos deben configurar procesos continuos de integración, construcción y prueba para brindar comentarios lo más rápido posible.

Los equipos ágiles requieren herramientas para tareas como el control del código fuente, la automatización de pruebas, los IDE y la gestión de compilaciones para facilitar las pruebas tecnológicas que respalden al equipo.

Esta página se dejó en blanco intencionalmente.

Capítulo 8

PRUEBAS EMPRESARIALES QUE APOYAR AL EQUIPO



En el último capítulo hablamos de las pruebas de programador, esas pruebas de bajo nivel que ayudan a los programadores a asegurarse de haber escrito el código correctamente. ¿Cómo saben qué es lo correcto para construir? En metodologías graduales y cerradas, tratamos de resolver esto reuniendo los requisitos por adelantado y dándoles el mayor detalle posible. En proyectos que utilizan prácticas ágiles, ponemos toda nuestra fe en story cards y pruebas que los clientes entienden para ayudar a codificar lo correcto. Estas pruebas “comprendibles” son el tema de este capítulo.

IMPULSANDO EL DESARROLLO CON PRUEBAS DE CARA AL NEGOCIO

Vaya, estamos comenzando una iteración sin más información que la que cabe. Una ficha, algo parecido a lo que se muestra en la Figura 8-1.

Esa no es mucha información y no está destinada a serlo. Las historias son una breve descripción de la funcionalidad deseada y una ayuda para planificar y priorizar. trabajar. En un proyecto tradicional en cascada, el equipo de desarrollo podría estar

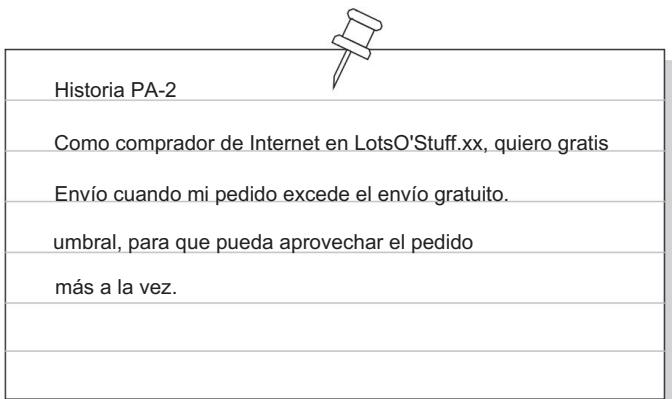


Figura 8-1 Historia para establecer una conversación

recibió un documento de requisitos detallado que incluye todos los detalles del conjunto de características. En un proyecto ágil, el equipo del cliente y el equipo de desarrollo hacen huelga entablar una conversación basada en la historia. El equipo necesita requisitos de algunos amables, y los necesitan a un nivel que les permita empezar a escribir, trabajar código casi de inmediato. Para hacer esto, necesitamos ejemplos que se conviertan en pruebas que confirmará lo que el cliente realmente quiere.

Estas pruebas orientadas al negocio abordan los requisitos comerciales. Estas pruebas ayudan proporcionar el panorama general y suficientes detalles para guiar la codificación. Orientado a los negocios Las pruebas expresan requisitos basados en ejemplos y utilizan un lenguaje y formato que tanto el cliente como los equipos de desarrollo puedan entender. Ejemplos forman la base para aprender el comportamiento deseado de cada característica, y utilizamos esos ejemplos como base para nuestras pruebas de historias en los Cuadrantes 2 (ver Figura 8-2).

Las pruebas orientadas a los negocios también se denominan "de cara al cliente", "historia", "cliente" o "de cara al cliente". y pruebas de "aceptación". El término "prueba de aceptación" es particularmente confuso, porque hace que algunas personas piensen sólo en "pruebas de aceptación del usuario". En el En el contexto del desarrollo ágil, las pruebas de aceptación generalmente se refieren a las pruebas orientadas al negocio, pero el término también podría incluir las pruebas orientadas a la tecnología de Cuadrante 4, como los criterios del cliente para el rendimiento o la seguridad del sistema. En este capítulo, analizamos solo las pruebas comerciales que respaldan al equipo al guiar el desarrollo y brindar retroalimentación rápida.

Como explicamos en los dos capítulos anteriores, el orden en el que presentamos Estos cuatro cuadrantes no están relacionados con el orden en el que podríamos realizar

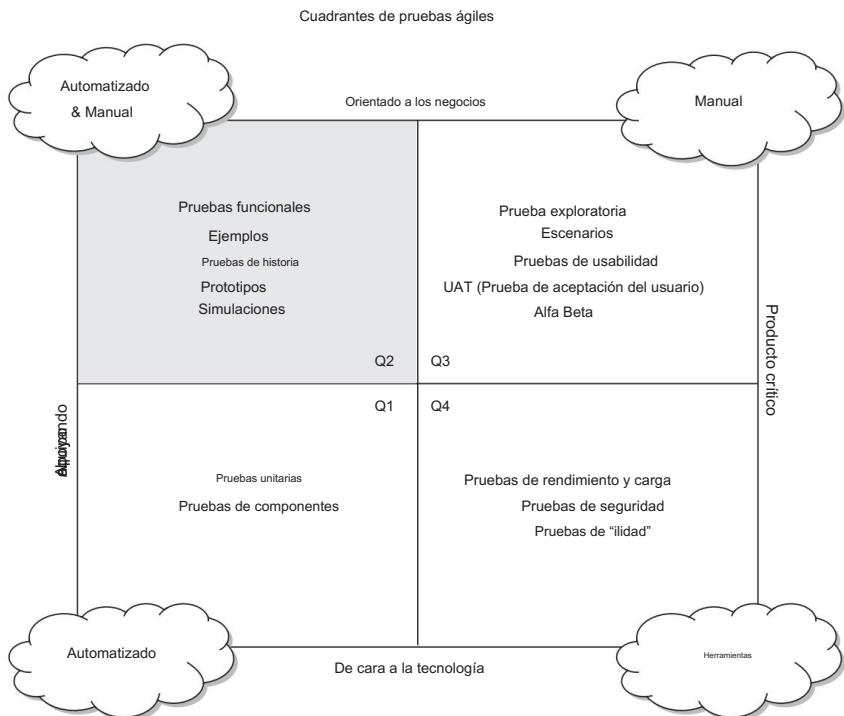


Figura 8-2 Los cuadrantes de pruebas ágiles, destacando el cuadrante 2

Parte V, "Una iteración en la vida", examina el orden en el que realizamos pruebas de los diferentes cuadrantes.

actividades de cada cuadrante. Las pruebas orientadas a los negocios en el Cuadrante 2 son escritos para cada historia antes de comenzar a codificar, porque ayudan al equipo comprender qué código escribir. Al igual que las pruebas del cuadrante 1, estas pruebas impulsan desarrollo, pero a un nivel superior. Las actividades del cuadrante 1 garantizan la calidad interna, maximizan la productividad del equipo y minimizan la deuda técnica. Cuadrante 2. Las pruebas definen y verifican la calidad externa y nos ayudan a saber cuándo hemos terminado.

Las pruebas del cliente para impulsar la codificación generalmente están escritas en un formato ejecutable y automatizadas, de modo que los miembros del equipo puedan ejecutar las pruebas con tanta frecuencia como quieran. Me gusta para ver si la funcionalidad funciona como se desea. Estas pruebas, o algunas Un subconjunto de ellos pasará a formar parte de un conjunto de regresión automatizada para que el desarrollo futuro no cambie involuntariamente el comportamiento del sistema.

Mientras discutimos las historias y ejemplos de comportamiento deseado, también debemos definir Requisitos no funcionales como rendimiento, seguridad y usabilidad. Bien

También tome nota de los escenarios para las pruebas exploratorias manuales. hablaremos de Estos otros tipos de actividades de prueba se describen en los capítulos sobre los cuadrantes 3 y 4.

Escuchamos muchas preguntas relacionadas con cómo los equipos ágiles obtienen los requisitos. Cómo ¿Sabemos qué debe hacer el código que escribimos? ¿Cómo obtenemos suficiente ¿Información para comenzar a codificar? ¿Cómo conseguimos que los clientes hablen con uno? expresar y presentar sus necesidades con claridad? ¿Por dónde empezamos en cada historia? Cómo ¿Conseguimos que los clientes nos den ejemplos? ¿Cómo los usamos para escribir? pruebas de historia?

Este capítulo explica nuestra estrategia para crear pruebas orientadas al negocio que apoyen al equipo a medida que desarrolla cada historia. Empecemos hablando más sobre requisitos.

EL DICIEMBRE DE LOS REQUISITOS

Casi todos los equipos de desarrollo que hemos conocido, ágiles o no, tienen dificultades requisitos. Los equipos de proyectos tradicionales en cascada podrían invertir meses en recopilación de requisitos sólo para que estén equivocados o salgan rápidamente de fecha. Es posible que los equipos en modo caos no tengan ningún requisito, y los programadores hacen sus mejores conjeturas sobre cómo debería funcionar una característica.

El desarrollo ágil acepta el cambio, pero ¿qué sucede cuando los requisitos? ¿Cambiar durante una iteración? No queremos una larga recopilación de requisitos. período antes de comenzar a codificar, pero ¿cómo podemos estar seguros de que nosotros (y nuestros clientes) realmente entendemos los detalles de cada historia?

En el desarrollo ágil, las nuevas funciones suelen comenzar como historias o grupos. de historias, escritas por el equipo del cliente. Escribir una historia no se trata de imaginar detalles de implementación, aunque las discusiones de alto nivel pueden tener un impacto en las dependencias y en la cantidad de historias que se crean. Es útil si algunos Los miembros del equipo técnico pueden participar en sesiones de redacción de cuentos para que que puedan contribuir a las historias de funcionalidad y ayudar a garantizar que Las historias técnicas se incluyen como parte del trabajo pendiente. Programadores y probadores También puede ayudar a los clientes a dividir las historias en tamaños apropiados, sugerir alternativas que podrían ser más prácticas de implementar y discutir las dependencias entre historias.

Las historias por sí solas no brindan muchos detalles sobre la funcionalidad deseada. Generalmente son sólo una oración que expresa quién quiere la función, cuál es el es la característica y por qué la quieren. "Como comprador de Internet, necesito una forma de eliminar artículos de mi carrito de compras para no tener que comprar artículos no deseados" deja un

mucho a la imaginación. Las historias sólo pretenden ser un punto de partida para un diálogo continuo entre los expertos empresariales y el equipo de desarrollo. Si el equipo Los miembros entienden qué problema el cliente está tratando de resolver, pueden sugerir alternativas que podrían ser más sencillas de utilizar e implementar.

En este diálogo entre clientes y desarrolladores, los equipos ágiles amplían historias hasta que tengan suficiente información para escribir el código apropiado. Probadores ayudan a obtener ejemplos y contexto para cada historia, y ayudan a los clientes a escribir pruebas de historia. Estas pruebas guían a los programadores mientras escriben el código y ayudan al equipo a saber cuándo ha cumplido las condiciones de satisfacción de los clientes. Si su equipo tiene casos de uso, pueden ayudar a complementar el ejemplo o la prueba de entrenamiento para aclarar la funcionalidad necesaria (consulte la Figura 8-3).

En el desarrollo ágil, aceptamos que nunca entenderemos todos los requisitos de una historia de antemano. Después del código que hace las pruebas de la historia. Una vez completado el pase, todavía necesitamos hacer más pruebas para comprender mejor el requisito y cómo deberían funcionar las características.

Una vez que los clientes tengan la oportunidad de ver lo que ofrece el equipo, es posible que tienen ideas diferentes sobre cómo quieren que funcione. A menudo los clientes tienen una idea vaga de lo que quieren y les resulta difícil definir exactamente qué es eso. El equipo trabaja con el cliente o el proxy del cliente durante una iteración y podría ofrecer sólo un núcleo de solución. El equipo sigue perfeccionando la funcionalidad a lo largo de múltiples iteraciones hasta que ha definido y entregado la función.

Ser capaz de iterar es una de las razones por las que el desarrollo ágil aboga por lanzamientos pequeños. y desarrollando un pequeño trozo a la vez. Si nuestro cliente no está satisfecho con el comportamiento del código que entregamos en esta iteración, podemos rectificarlo rápidamente que en la próxima, si lo consideran importante. Los cambios de requisitos son bonitos. mucho inevitable.

Debemos aprender todo lo que podamos sobre los deseos y necesidades de nuestros clientes. Si nuestros usuarios finales trabajan en nuestra ubicación, es factible viajar a la de ellos, nosotros Deberíamos sentarnos con ellos, trabajar junto a ellos y poder hacer su trabajo si poder. No sólo entenderemos mejor sus requisitos sino que también podremos incluso identificar requisitos que no pensaron en establecer.

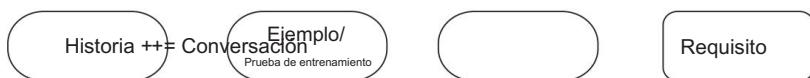


Figura 8-3 La composición de un requisito

Las pruebas deben incluir más que los requisitos declarados por los clientes. Nosotros necesitamos para probar las condiciones posteriores, el impacto en el sistema en su conjunto y la integración con otros sistemas. Identificamos riesgos y los mitigamos con pruebas según sea necesario. Todos estos factores guían nuestra codificación.

Lenguaje común

También podemos utilizar nuestras pruebas para proporcionar un lenguaje común que se entienda. tanto por el equipo de desarrollo como por los expertos empresariales. Como Brian Marick [2004] señala que un lenguaje compartido ayuda a los empresarios a visualizar el características que desean. Ayuda a los programadores a crear código bien diseñado que sea fácil de ampliar. Se pueden encontrar ejemplos de la vida real de conductas deseadas y no deseadas. expresados de manera que sean comprendidos tanto por el sector empresarial como por el técnico. lados. Se puede acceder a imágenes, diagramas de flujo, hojas de cálculo y prototipos. personas con diferentes orígenes y puntos de vista. Podemos utilizar estas herramientas para encontrar ejemplos y luego convierta fácilmente esos ejemplos en pruebas. las pruebas necesitan estar escritos de una manera que sea comprensible para un usuario empresarial que los lea aún ejecutable por el equipo técnico.

Más información

sobre Fit en el Capítulo 9, "Kit de herramientas para pruebas empresariales que respaldan al equipo".

Las pruebas de cara al negocio también ayudan a definir el alcance, para que todos sepan qué es parte de la historia y lo que no lo es. Muchos de los marcos de prueba ahora permiten equipos para crear un idioma de dominio y definir pruebas utilizando ese idioma. Adaptar (Funcional para Marco Integrado) es uno de esos.

El cliente perfecto

Andy Pols nos permitió reimprimir esta historia desde su blog [Pols, 2008]. En él, muestra cómo su cliente exigió una prueba, la escribió y se dio cuenta de que la historia estaba fuera de alcance.

En un proyecto reciente, nuestro cliente se entusiasmó tanto con nuestras pruebas de ajuste que se molestó mucho cuando implementé una historia sin una prueba de ajuste.

Se negó a permitir que el sistema funcionara hasta que tuviéramos la prueba de ajuste en su lugar.

La historia en cuestión era muy técnica e implicaba enviar un mensaje XML particular a un sistema externo. Simplemente no pudimos determinar cómo sería una prueba de ajuste para este tipo de requisito. Colocar el mensaje XML esperado, con todos sus detalles sangrientos, en la prueba de ajuste no habría sido útil porque se trata de un artefacto técnico y no tiene ningún interés para el negocio. No pudimos decidir qué hacer. El cliente no estaba presente para discutir esto, así que seguí adelante e implementé la historia (¡muy traviesa!).

Lo que el cliente quería era estar seguro de que enviábamos la información correcta del producto en el mensaje XML. Para resolver el problema, sugerí que hicieramos una prueba de ajuste que mostrara cómo los atributos del producto se asignan al mensaje XML usando Xpath, aunque todavía pensaba que esto era demasiado técnico para un usuario empresarial.

Le dimos al cliente un par de enlaces para explicar qué era XPath para que pudiera explorar si era una buena solución para él. Para mi sorpresa, quedó encantado con XPath (ahora sé a quién acudir cuando tengo un problema con XPath) y llenó el test de Fit.

Lo interesante para mí es que tan pronto como supo cómo era el mensaje y cómo estaba estructurado, se dio cuenta de que en realidad no respaldaba el negocio: estábamos enviando información que estaba fuera del alcance de nuestro trabajo y que debería haber sido enviada. sido alimentado por otro sistema. También se mostró escéptico sobre la velocidad a la que el equipo externo podría agregar nuevos productos debido a la naturaleza compleja del XML.

A la mayoría de las personas ágiles les contamos esta historia para pensar que tenemos el "perfecto ¡cliente!"

Incluso si sus clientes no son perfectos, involucrados en la redacción de pruebas para clientes les brinda la oportunidad de identificar funcionalidades que están fuera del alcance de la historia. Intentamos escribir pruebas para los clientes que los clientes puedan leer y comprender. A veces ponemos el listón demasiado bajo. Colabore con sus clientes para encontrar una herramienta y un formato para redactar pruebas que funcionen tanto para el cliente como para los equipos de desarrollo.

Está bien decir que nuestros clientes nos brindarán los ejemplos que necesitamos tener para que entendamos el valor que debe tener cada historia. entregar. ¿Pero qué pasa si no saben cómo explicar lo que quieren? En el En la siguiente sección, sugeriremos formas de ayudar a los clientes a definir sus condiciones de satisfacción.

Obtención de requisitos

Si alguna vez ha sido un cliente que solicita una función de software en particular, Sepa lo difícil que es articular exactamente lo que quiere. A menudo, no sabes exactamente lo que quieres hasta que puedes verlo, sentirlo, tocarlo y usarlo. Nosotros Tenemos muchas maneras de ayudar a nuestros clientes a obtener claridad sobre lo que quieren.

Hacer preguntas

Empiece por hacer preguntas. Los evaluadores pueden ser especialmente buenos preguntando una variedad de preguntas porque son conscientes del panorama general, del aspecto empresarial

y aspectos técnicos de la historia, y siempre están pensando en la experiencia del usuario final. Los tipos de preguntas generales que se pueden hacer son:

- ¿Esta historia está resolviendo un problema?
- Si es así, ¿cuál es el problema que estamos tratando de resolver?
- ¿Podríamos implementar una solución que no resuelva el problema?
- ¿Cómo aportará la historia valor al negocio?
- ¿Quiénes son los usuarios finales de la función?
- ¿Qué valor obtendrán de ello?
- ¿Qué harán los usuarios justo antes y después de utilizar esa función?
- ¿Cómo sabemos que hemos terminado con esta historia?

Una pregunta que a Lisa le gusta hacer es: "¿Qué es lo peor que podría pasar?"

Los peores escenarios tienden a generar ideas. También nos ayudan a considerar el riesgo. y centrar nuestras pruebas en áreas críticas. Otra buena pregunta es: "¿Cuál es el ¿Lo mejor que podría pasar?" Esta pregunta suele generar nuestra felicidad. prueba de ruta, pero también podría revelar algunas suposiciones ocultas.

Ejemplos de uso

Lo más importante es pedirle al cliente que le dé ejemplos de cómo funciona la función. Debería trabajar. Digamos que la historia trata sobre eliminar artículos de un carrito de compras en línea. Pídale al cliente que haga un dibujo en una pizarra de cómo eso La función de eliminación podría verse. ¿Quieren funciones adicionales, como un paso de confirmación o la posibilidad de guardar el elemento en caso de que quieran recuperarlo más tarde? ¿Qué esperarían ver si no se pudiera realizar la eliminación?

Los ejemplos pueden formar la base de nuestras pruebas. Nuestro desafío es capturar ejemplos, que podrían expresarse en el lenguaje del dominio empresarial, como pruebas que realmente se puede ejecutar. Algunos clientes se sienten cómodos expresando ejemplos usando una herramienta de prueba como Fit o FitNesse siempre que puedan escribirlos en el idioma de su dominio.

Exploraremos la diferencia entre un ejemplo y una prueba con una historia simple. (ver Figura 8-4). La gente suele confundirse entre estos dos términos.

Un ejemplo sería algo como esto:

Hay 5 elementos en una página. Quiero seleccionar el artículo 1 por \$20.25 y poner en el carrito de compras. Hago clic en la página siguiente, que tiene 5 elementos más. Selecciono un segundo artículo en esa página por \$5.38 y lo pongo en mi carrito de compras. carro. Cuando digo que terminé de comprar, se mostrará tanto el artículo del

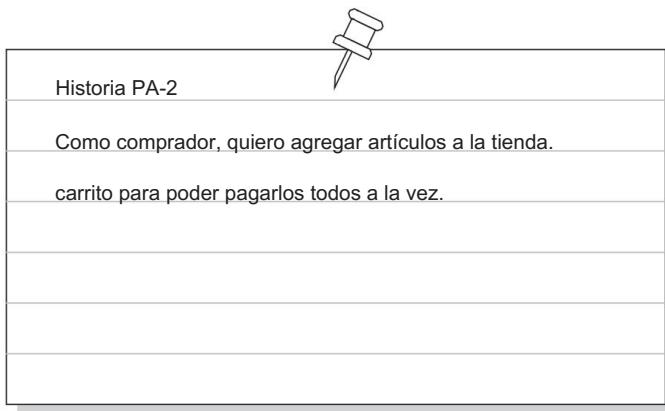


Figura 8-4 Historia que se utilizará como base para ejemplos y pruebas

primera página y el artículo de la segunda página en mi carrito de compras, con el total de \$25.63

La prueba podría ser bastante diferente. Usaremos un formato de tipo Ajustar en la Tabla 8-1 para mostrarle cómo se podría representar la prueba.

La prueba captura el ejemplo en un formato ejecutable. Puede que no use exactamente las mismas entradas, pero encapsula el escenario de usuario de muestra. Más casos de prueba se puede escribir para probar condiciones de contorno, casos extremos y otros escenarios.

Múltiples puntos de vista

Cada ejemplo o prueba tiene un punto de vista. Diferentes personas escribirán diferentes pruebas o ejemplos desde sus perspectivas únicas. Nos gustaría capturar como tantos puntos de vista diferentes como podamos, así que piense en sus usuarios.

Tabla 8-1 Prueba para la historia PA-2

	Entradas		Resultados previstos	
—	Artículo	Precio	Coste total	# de artículos
001	Artículo A	20.25	20.25	1
002	Artículo D	0,01	20.26	2
003	Artículo F	100,99	121,25	3

Cumplir correctamente los requisitos es un área en la que los miembros del equipo que desempeñan muchos roles diferentes pueden intervenir para ayudar. Analistas de negocios, expertos en la materia, los programadores y varios miembros del equipo del cliente tienen algo que aportar. Piense en otras partes interesadas, como su producción. equipo de apoyo. Tienen una perspectiva única.

A menudo nos olvidamos de requisitos no funcionales como "¿Cuánto tiempo dura?". ¿El sistema necesita estar activo? ¿Qué pasa si falla? Si tenemos middleware que pasa mensajes, ¿esperamos que los mensajes sean lo suficientemente grandes como para ¿Podría ser necesario considerar la pérdida durante la transmisión? ¿O serán una constante? ¿tamaño? ¿Qué pasa si no hay tráfico durante horas? ¿El sistema necesita ¿advertir a alguien? Las pruebas para este tipo de requisitos generalmente caen dentro de cuadrantes 3 y 4, pero aún necesitamos escribir pruebas para asegurarnos de que se realicen.

Todos los ejemplos que los clientes dan al equipo se acumulan rápidamente. ¿Realmente tenemos que convertir todo esto en pruebas ejecutables? No mientras tengamos la clientes allí para decirnos si el código funciona de la manera que desean. Con técnicas como la creación de prototipos en papel, los diseños se pueden probar antes de que se escriba una línea de código. está escrito.

Pruebas del Mago de Oz

Gerard Meszaros, ScrumMaster certificado (practicante) y Agile Coach, compartió su historia sobre las pruebas del Mago de Oz en proyectos ágiles. Describe un buen ejemplo de cómo los artefactos que generamos para generar requisitos pueden ayudar a comunicar significado de forma inequívoca.

Pensamos que estábamos listos para lanzar nuestro software. Lo habíamos estado construyendo una iteración a la vez bajo la guía de un cliente en el sitio que había priorizado la funcionalidad en función de lo que necesitaba para iniciar las pruebas de integración con sus socios comerciales. Aplazamos conscientemente la funcionalidad de generación de informes y mantenimiento de datos maestros para iteraciones posteriores para garantizar que teníamos lista la funcionalidad necesaria para las pruebas de integración. Las pruebas de integración fueron bien, y solo se registraron algunos defectos (todos relacionados con funcionalidades faltantes o mal entendidas). Mientras tanto, en las últimas iteraciones implementamos el mantenimiento de los datos maestros en paralelo con las pruebas de integración. Cuando realizamos las pruebas de aceptación con los usuarios empresariales, nos llevamos una dura sorpresa: ¡odiaban la funcionalidad de mantenimiento y generación de informes! Registraron tantos defectos y "mejoras imprescindibles" que tuvimos que retrasar el lanzamiento un mes. ¡Demasiado para idear un plan que nos permitiera realizar entregas anticipadas!

Mientras reimplementábamos el mantenimiento de datos maestros, asistí a la conferencia Agile 2005 y tomé un tutorial de Jeff Patton. Uno de los ejercicios fue crear prototipos en papel de la interfaz de usuario para una aplicación de muestra. Luego "probamos" los prototipos en papel con miembros de otros grupos como nuestros usuarios y descubrimos cuán defectuosos eran nuestros diseños de interfaz de usuario. ¡Deja Vu! El tutorial se parecía a mi realidad.

A mi regreso al proyecto en casa, llevé aparte al director de proyecto al que estaba asesorando en desarrollo ágil y le sugerí que se hicieran prototipos en papel y pruebas del "Mago de Oz" (la referencia del Mago de Oz es a un ser humano que actúa como una computadora). (una especie de "hombre detrás de la cortina") podría haber evitado nuestro revés de un mes. Después de una breve discusión, decidimos probarlo en nuestra funcionalidad de la versión 2. Nos quedamos hasta tarde un par de noches y diseñamos la interfaz de usuario utilizando capturas de pantalla de la funcionalidad R1 superpuestas con la funcionalidad R2 dibujada a mano. Hacía mucho tiempo que ninguno de nosotros usaba tijeras y barras de pegamento, ¡y fue divertido!

Para las pruebas del Mago de Oz con usuarios, les pedimos a nuestros clientes en el sitio que encontraran algunos usuarios reales con quienes realizar las pruebas. También propusieron algunas tareas de muestra realistas para que los usuarios intentaran ejecutarlas. Colocamos los datos de muestra en hojas de cálculo de Excel e imprimimos varias combinaciones de cuadriculas de datos para usar en las pruebas. Algunos futuros usuarios vinieron a la ciudad para asistir a una conferencia. Secuestramos pares de ellos durante una hora cada uno e hicimos nuestras pruebas.

Actué como el "mago", desempeñando el papel de la computadora ("es un procesador 286, así que no esperes que los tiempos de respuesta sean muy buenos"). El cliente in situ presentó el problema y los programadores actuaron como observadores, registrando los pasos en falso de los usuarios como "posibles defectos". Después de unas pocas horas, teníamos enormes cantidades de datos valiosos sobre qué partes de nuestro diseño de interfaz de usuario funcionaban bien y qué partes necesitaban repensarse. ¡Y hubo poca discusión sobre cuál era cuál! Repetimos las pruebas de usabilidad con otros usuarios cuando tuvimos versiones alfa de la aplicación disponibles y obtuvimos más información valiosa. Nuestro cliente empresarial encontró el ejercicio tan valioso que en un proyecto posterior el equipo empresarial se dedicó a crear prototipos en papel y pruebas del Mago de Oz sin que el equipo de desarrollo se lo pidiera. Esto podría haber estado influenciado en cierta medida por el primer correo electrónico que recibimos de un usuario real 30 minutos después de su lanzamiento: "¡¡¡Me encanta esta aplicación!!!"

Desarrollar interfaces de usuario probando primero puede parecer un esfuerzo intimidante. La técnica del Mago de Oz se puede realizar antes de escribir una sola línea de código. El equipo puede probar la interacción del usuario con el sistema y recopilar mucha información para comprender el comportamiento deseado del sistema. Es una excelente manera de facilitar la comunicación entre el cliente y los equipos de desarrollo.

Colaboración estrecha y constante entre el equipo del cliente y el desarrollador.

El equipo es clave para obtener ejemplos en los que basar las pruebas de los clientes que impulsen codificación. La comunicación es un valor ágil central y hablamos más de ello en la siguiente sección.

Comunicarse con los clientes

En un mundo ideal, nuestros clientes están disponibles para nosotros todo el día, todos los días. En realidad, muchos equipos tienen acceso limitado a sus expertos comerciales y, en muchos casos,

En algunos casos, los clientes se encuentran en una ubicación o zona horaria diferente. Hacer cualquier cosa puedes tener conversaciones cara a cara. Cuando no puedes, llamadas en conferencia,

Las conversaciones telefónicas, los correos electrónicos, los mensajes instantáneos, las cámaras y otras herramientas de comunicación tendrán que sustituirlos. Afortunadamente, cada vez hay más herramientas disponibles para facilitar la comunicación remota. Hemos oido hablar de equipos, como

El equipo de Erika Boyer en iLevel by Weyerhaeuser, que utiliza cámaras web que pueden ser controlado por la gente en lugares remotos. Acércate lo más que puedas para dirigir la conversación.

La historia de Lisa

Trabajé en un equipo donde los programadores estaban distribuidos en tres zonas horarias y los clientes en una diferente. Enviamos diferentes programadores, evaluadores y analistas al sitio del cliente para cada iteración, de modo que cada miembro del equipo tuviera "tiempo cara a cara" con los clientes al menos cada tercera iteración. Esto generó confianza entre los equipos de desarrolladores y clientes. El resto del tiempo utilizamos llamadas telefónicas, conferencias telefónicas abiertas y mensajes instantáneos para hacer preguntas. Con continuos ajustes basados en discusiones retrospectivas, logramos satisfacer e incluso deleitar a los clientes.

—Lisa

Incluso cuando los clientes están disponibles y las líneas de comunicación son amplias abierta, es necesario gestionar la comunicación. Queremos hablar con cada miembro del equipo de atención al cliente, pero todos tienen puntos de vista diferentes. Si conseguimos varias diferentes versiones de cómo debería funcionar una funcionalidad, no saber qué codificar. Consideraremos formas de lograr que los clientes se pongan de acuerdo sobre las condiciones de satisfacción para cada historia.

Claridad avanzada

Si su equipo de clientes está formado por personas de diferentes partes de la organización, puede haber opiniones contradictorias entre ellos sobre lo que se pretende exactamente con una historia en particular. En la empresa de Lisa, el desarrollo empresarial quiere

características que generan ingresos, operaciones quiere características que reduzcan llamadas de soporte telefónico y finanzas quiere funciones que agilicen la contabilidad, gestión de efectivo y presentación de informes. Es sorprendente cuántas interpretaciones únicas de la misma historia pueden surgir de personas que tienen puntos de vista diferentes.

La historia de Lisa

Aunque teníamos un propietario de producto cuando implementamos Scrum por primera vez, todavía recibíamos diferentes directivas de diferentes clientes. La gerencia decidió nombrar a un vicepresidente con amplio conocimiento de dominio y operaciones como nuevo propietario del producto. Está a cargo de lograr que todas las partes interesadas se pongan de acuerdo desde el principio sobre las implicaciones de cada historia. Él y el resto del equipo del cliente se reúnen periódicamente para discutir temas e historias futuras y acordar prioridades y condiciones de satisfacción. A esto lo llama "claridad avanzada".

—Lisa

Un Product Owner es un rol en Scrum. Él es responsable no sólo de lograr avanzar en la claridad, sino también actuar como "representante del cliente" al priorizar historias. Sin embargo, hay una desventaja. Cuando canalizas las necesidades de Muchos puntos de vista diferentes a través de una persona, algo se puede perder. Idealmente, el equipo de desarrollo debería sentarse junto con el equipo del cliente y aprender cómo hacer el trabajo del cliente. Si entendemos las necesidades del cliente suficientemente bien para realizar sus tareas diarias, tenemos muchas más posibilidades de producir software que respalde adecuadamente esas tareas.

La historia de Janet

Nuestro equipo no implementó el rol de propietario del producto al principio y utilizó a los expertos en el dominio del equipo para determinar la priorización y la claridad. Funcionó bien, pero lograr el consenso requirió muchas reuniones porque cada persona tuvo experiencias diferentes. El producto era mejor, pero había compensaciones. Debido a las numerosas reuniones, los expertos en el dominio no siempre estaban disponibles para responder las preguntas de los programadores, por lo que la codificación fue más lenta de lo previsto.

Había cuatro equipos de proyecto separados trabajando en el mismo producto, pero cada uno se centraba en funciones diferentes. Después de varias retrospectivas y muchas sesiones de resolución de problemas, cada equipo de proyecto nombró un propietario de producto. El número de reuniones se redujo significativamente porque la mayoría de las decisiones comerciales las tomaban los expertos en el dominio de su proyecto particular. Se llevaron a cabo reuniones para todos los expertos del dominio si había alguna diferencia de opinión, y el propietario del producto facilitó el logro de consenso sobre un tema. Las decisiones se tomaron mucho más rápido, los expertos en el dominio estuvieron más disponibles para responder las preguntas del equipo y pudieron mantenerse al día con las pruebas de aceptación.

—Janet

Independientemente de cómo su equipo elija reunir diferentes puntos de vista, es importante que solo se presente "una voz del cliente" al equipo.

Dijimos que los propietarios de productos brindan condiciones de satisfacción. miremos más de cerca lo que queremos decir.

Condiciones de satisfacción

Existen condiciones de satisfacción tanto para toda la liberación como para cada característica o historia. Las pruebas de aceptación ayudan a definir la aceptación de la historia. Su equipo de desarrollo no puede entregar con éxito lo que la empresa quiere a menos que se acuerden desde el principio las condiciones de satisfacción para una historia. El equipo del cliente Necesita "hablar con una sola voz". Si obtiene requisitos diferentes de diferentes partes interesadas, es posible que tengas que retroceder y posponer la historia hasta dispone de una lista firme de condiciones de satisfacción empresarial. Pídale al representante del cliente que proporcione una cantidad mínima de información sobre cada historia para que puedas comenzar cada iteración con una conversación productiva.

La mejor manera de comprender los requisitos del equipo del cliente es hablar con los clientes cara a cara. Porque todo el mundo lucha con los "requisitos" Existen herramientas para ayudar al equipo del cliente a trabajar en cada historia. Las condiciones de satisfacción deben incluir no sólo las características que ofrece la historia sino también los impactos en el sistema más amplio.

El propietario del producto de Lisa utiliza un formato de lista de verificación para resolver problemas como:

Condiciones de satisfacción empresarial

Impacto en funciones existentes como el sitio web, documentos, facturas, formularios o informes.

Consideraciones legales

El impacto en los procesos programados regularmente

Referencias a maquetas de historias de UI

Texto de ayuda, o quién lo proporcionará

Casos de prueba

Migración de datos (según corresponda)

Comunicación interna que debe suceder

Comunicación externa a socios comerciales y proveedores.

El Capítulo 9, "Kit de herramientas para pruebas empresariales que respaldan al equipo", incluye listas de verificación de ejemplo, así como otras herramientas para expresando requisitos.

El propietario del producto utiliza una plantilla para colocar esta información en la página del equipo. wiki para que pueda usarse cuando los miembros del equipo aprendan sobre las historias y comiencen pruebas de escritura.

Estas condiciones se basan en suposiciones y decisiones clave tomadas por el equipo de clientes para una historia. Generalmente surgen de conversaciones con el cliente sobre los criterios de aceptación de alto nivel para cada historia. Discutir las condiciones de satisfacción ayuda a identificar suposiciones riesgosas y aumenta la confianza del equipo para escribir y estimar correctamente todas las tareas que se necesitan para completar la historia.

Efectos de la ondulación

En el desarrollo ágil, nos centramos en una historia a la vez. Cada historia suele ser una pequeño componente de la aplicación general, pero podría tener un gran efecto dominó. Una nueva historia cae como una piedrecita en el agua de la aplicación, y nosotros Es posible que no piense en dónde podrían chocar las ondas resultantes. Es fácil de perder la noción del panorama general cuando nos centramos en una pequeña cantidad de historias en cada iteración.

Al equipo de Lisa le resulta útil hacer una lista de todas las partes del sistema que podría verse afectado por una historia. El equipo puede comprobar cada “punto de prueba” para ver qué requisitos y casos de prueba que pueda generar. Una pequeña e inocente historia. podría tener un impacto de amplio alcance, y cada parte de la aplicación que Los toques pueden presentar otro nivel de complejidad. Necesitas estar al tanto de todo los impactos potenciales de cualquier cambio de código. Hacer una lista es un buen punto de partida. En los primeros días de la iteración, el equipo puede investigar y analizar las zonas afectadas. áreas más a fondo y vea si se necesitan más tarjetas de tareas para cubrirlas todas.

La historia de Janet

—
Capítulo 16, “Golpe the Ground Run-ning”
y el Capítulo 17,
“Iteración Kickoff”, brinda
ejemplos de cuándo y
cómo los equipos
pueden planificar
pruebas con clientes y
explorar el impacto más
amplio de cada
historia.

En un proyecto en el que estuve, utilizamos una hoja de cálculo simple que enumeraba todas las funciones de alto nivel de la aplicación bajo prueba. Durante la planificación del lanzamiento y al comienzo de cada nueva iteración, revisamos la lista y pensamos en cómo la funcionalidad nueva o cambiante afectaría esas áreas. Ese se convirtió en el punto de partida para determinar qué nivel de pruebas era necesario realizar en cada área funcional. Este análisis de impacto se sumó a las pruebas de la historia real y permitió a nuestro equipo ver el panorama general y el impacto de los cambios en el resto del sistema.

—Janet

Las historias que parecen pequeñas pero que impactan áreas inesperadas del sistema pueden volver a morderte. Si su equipo se olvida de considerar todas las dependencias y Si el nuevo código se cruza con la funcionalidad existente, su historia podría tomar mucho más de lo planeado para terminar. Asegúrese de que sus pruebas de historia incluyan consecuencias menos obvias de la implementación de la nueva funcionalidad.

Tómese el tiempo para identificar el valor central que proporciona cada historia y descubra una enfoque incremental para su desarrollo. Planifique pequeños incrementos de escritura pruebas, escribir código y probar el código un poco más. De esta manera, tu Cuadrante 2 Las pruebas garantizan que entregará el valor mínimo según lo planeado.

REBANADAS FINAS , PEQUEÑOS TROZOS

Escribir historias es un asunto complicado. Cuando el equipo de desarrollo estima nuevas historias, es posible que algunas le parezcan demasiado grandes, por lo que le preguntará al cliente equipo para regresar y dividirlas en historias más pequeñas. Las historias pueden ser demasiado pequeñas también, y podría ser necesario combinarlos con otros o simplemente tratarlos como tareas. El desarrollo ágil, incluidas las pruebas, ocupa una pequeña parte del funcionalidad a la vez.

Cuando su equipo se embarque en un nuevo proyecto o tema, pregúntele al propietario del producto. Llevar todas las historias relacionadas a una sesión de lluvia de ideas antes de la primera iteración para ese tema. Haga que el propietario del producto y otras partes interesadas expliquen las historias. Es posible que descubra que es necesario subdividir algunas historias o que es necesario escribir historias adicionales para llenar los vacíos.

Después de comprender qué valor debe ofrecer cada historia y cómo encaja En el contexto del sistema, puedes dividir las historias en partes pequeñas y manejables. Puede escribir pruebas de clientes para definir esos pequeños incrementos, teniendo en cuenta el impacto en la aplicación más amplia.

Un enfoque incremental inteligente para redactar pruebas de clientes que guíen el desarrollo es comenzar con la "porción delgada" que sigue un camino feliz desde un extremo. al otro. La identificación de un corte fino, también llamado "hilo de acero" o "bala trazadora", se puede realizar a nivel temático, donde se utiliza para verificar la arquitectura general. Este hilo de acero conecta todos los componentes entre sí, y después Es sólido, se pueden agregar más funciones.

Consulte el Capítulo 10,
"Presentación empresarial
Pruebas que critican el
producto", para obtener
más información sobre
las pruebas exploratorias.

Descubrimos que esta estrategia también funciona a nivel de historia. Cuanto antes puedas construir el camino de un extremo a otro, antes podrá realizar pruebas significativas, obtener comentarios, comenzar automatizar pruebas e iniciar pruebas exploratorias. Comience con una rebanada fina de funcionalidad más simplificada que se puede probar. Esto puede considerarse como el camino crítico. Para una interfaz de usuario, esto podría comenzar simplemente navegando de una página a la siguiente. Podemos mostrárselo al cliente y ver si el flujo tiene sentido. Podríamos escribir una prueba GUI automatizada simple. Para el historia del umbral de envío gratuito al comienzo de este capítulo, podríamos comenzar verificando la lógica utilizada para resumir el total del pedido y determinar si



Consulte la Parte IV.

"Automatización", para obtener

más información sobre la

automatización de pruebas de regresión. Una vez que la porción fina esté funcionando, podemos escribir pruebas de clientes para la siguiente porción.

o capa de funcionalidad, y escribir el código que hace que esas pruebas pasen. Ahora

También recibiremos comentarios sobre este pequeño incremento. Tal vez agreguemos la interfaz de usuario a mostrar la página de pago que muestra que el pedido califica para envío gratuito,

o agregue la capa para conservar las actualizaciones de la base de datos. Podemos agregar pruebas automatizadas que escribimos para la primera pasada. Es un proceso de "escribir pruebas, escribir codificar, ejecutar pruebas, aprender". Si haces esto, sabrás que todo el código de tu

El equipo produce, satisface al cliente y trabaja correctamente en cada etapa.

La historia de Lisa

Mi equipo descubrió que tenemos que concentrarnos en lograr una rebanada delgada y simple y agregarla en pequeños incrementos. Antes de hacer esto, tendíamos a quedarnos estancados en una parte de la historia. Por ejemplo, si tuviéramos un flujo de interfaz de usuario que incluyera cuatro pantallas, involucrarnos tanto en el primero que tal vez no lleguemos al último, y no había un camino que funcionara de principio a fin. Al comenzar con un camino feliz de extremo a extremo y agregar funcionalidad paso a paso, podemos estar seguros de entregar el valor mínimo necesario.

Aquí hay un ejemplo de nuestro proceso. La idea era añadir un nuevo paso condicional al proceso de establecimiento del plan de jubilación de una empresa. Este paso permite a los usuarios seleccionar carteras de fondos mutuos, pero no todos los usuarios tienen acceso a esta función. La funcionalidad de establecimiento del plan de jubilación está escrita en un código antiguo y mal diseñado. Planeamos escribir la nueva página en la nueva arquitectura, pero vincular el código nuevo y el antiguo es complicado y propenso a errores. Dividimos la historia en porciones que pueden parecer pequeñas pero que nos permitieron gestionar el riesgo y minimizar el tiempo necesario para codificar y probar la historia. La Figura 8-5 muestra un diagrama de los pasos incrementales planificados para esta historia.

La porción delgada n.º 1 es insertar una página nueva y vacía basada en una propiedad. Si bien no es mucho para nuestros clientes, nos permite probar el puente entre el código antiguo y el nuevo, y luego verificar que la navegación de establecimiento del plan aún funciona correctamente.

La porción 2 introduce cierta lógica empresarial: si no hay carteras de fondos mutuos disponibles para la empresa, salte al paso de selección de fondos, que aún no vamos a cambiar. Si hay carteras de fondos disponibles, muéstrelas en el nuevo paso 3. En el segmento 3, cambiamos el paso de selección de fondos, agregando lógica para mostrar los fondos que componen las carteras. El segmento 4 agrega elementos de navegación entre varios pasos del proceso de establecimiento.

Escribimos pruebas de clientes para definir cada porción. A medida que los programadores completaron cada uno, lo probamos manualmente y se lo mostramos a nuestros clientes. Cualquier problema encontrado se solucionó de inmediato. Escribimos una prueba de GUI automatizada para el segmento n.º 1 y la agregamos a medida que finalizaban los pasos restantes. La historia fue difícil debido a que el antiguo código heredado interactuaba con la nueva arquitectura, pero el enfoque paso a paso hizo que la implementación fuera fluida y ahorró tiempo.

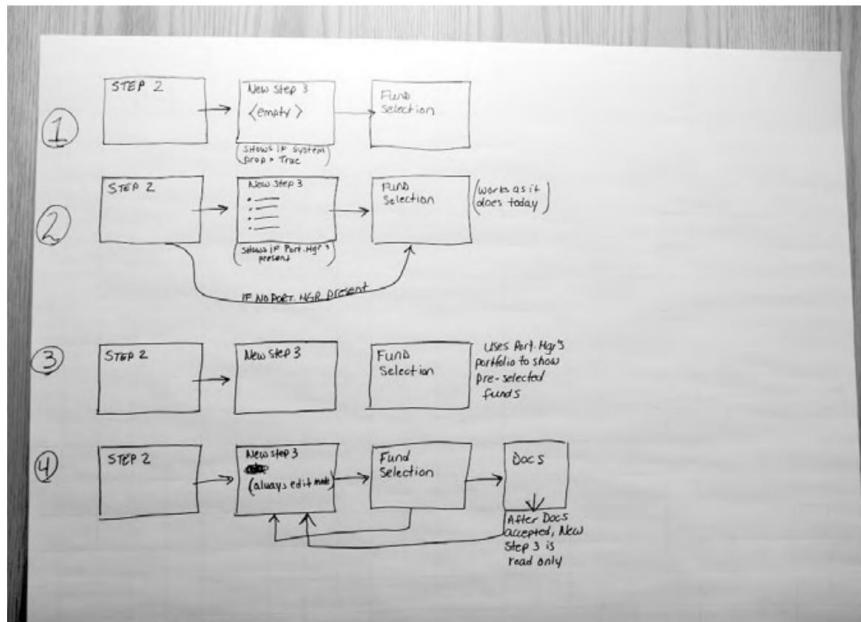


Figura 8-5 Pasos incrementales

Consulta la bibliografía de Gerard. El artículo de Meszaros. "Usar tipos Story para dividir XP hinchado Cuentos."

Cuando dibujamos diagramas como este para dividir las historias en partes, subimos fotografías de ellas a la wiki de nuestro equipo para que nuestro miembro remoto del equipo también pueda verlas. Como cada paso está finalizado, lo marcamos para proporcionar información visual instantánea.

—Lisa

Si la tarea de escribir pruebas de clientes para una historia parece confusa o abrumadora, es posible que su equipo necesite dividir la historia en pasos más pequeños o trozos. Terminar las historias paso a paso ayuda a distribuir las pruebas, esfuerzo para que no sea empujado hasta el final de la iteración. También te da una mejor imagen de su progreso y le ayuda a saber cuándo ha terminado: un tema que exploraremos en la siguiente sección.

¿CÓMO SABEMOS QUE HEMOS TERMINADO?

Contamos con nuestras pruebas comerciales que respaldan al equipo: aquellas pruebas que han sido redactados para garantizar que se hayan cumplido las condiciones de satisfacción.

Comienzan con el camino feliz y demuestran que la historia cumple con lo previsto.

necesidad. Cubren varios escenarios de usuario y garantizan que otras partes del sistema no se ve afectado negativamente. Estas pruebas se han realizado y pasan (o al menos han identificado problemas que deben solucionarse).

¿Hemos terminado ahora? Podríamos serlo, pero aún no estamos seguros. La verdadera prueba es si el usuario del software puede realizar la acción que se suponía que debía realizar la historia proporcionar. Las actividades de los cuadrantes 3 y 4, como las pruebas exploratorias, las pruebas de capacidad de nosotros y las pruebas de desempeño, nos ayudarán a descubrirlo. Por ahora, nosotros Solo necesitamos hacer algunas pruebas con los clientes para asegurarnos de que hemos capturado todos los requisitos. Los usuarios comerciales o propietarios de productos son las personas adecuadas para determinar si se han cumplido todos los requisitos, para que sean los correctos personas para hacer la exploración en esta etapa.

Cuando todas las pruebas pasan y se han identificado los requisitos no cumplidos, se realizan con el propósito de apoyar a los programadores en su búsqueda de código que hace "lo correcto". No significa que hayamos terminado de realizar las pruebas. Bien Hablaremos mucho más sobre eso en los capítulos que siguen.

Otro objetivo de las pruebas de clientes es identificar áreas de alto riesgo y asegurarse el código está escrito para solidificarlos. La gestión de riesgos es una práctica esencial en cualquier metodología de desarrollo de software, y los evaluadores desempeñan un papel en la identificación y mitigación de riesgos.

LAS PRUEBAS MITIGAN EL RIESGO

Las pruebas de los clientes se escriben no solo para definir el comportamiento esperado del código. sino para gestionar el riesgo. Impulsar el desarrollo con pruebas no significa que identificaremos cada requisito por adelantado o que seremos capaces de predecir perfectamente cuándo hemos terminado. Nos da la oportunidad de identificar riesgos y mitigarlos con casos de prueba ejecutables. El análisis de riesgos no es una técnica nueva. Desarrollo ágil mitiga inherentemente algunos riesgos al priorizar el valor del negocio en pequeñas y medianas empresas. piezas entregables probadas y al involucrar al cliente en procesos incrementales aceptación. Sin embargo, aún debemos hacer una lluvia de ideas sobre eventos potenciales, la probabilidad de que ocurran y el impacto en la organización si suceden para que se pueda emplear la estrategia de mitigación adecuada.

La codificación de pruebas predefinidas no funciona bien si las pruebas son improbables. casos extremos. Si bien no queremos probar sólo el camino feliz, es un buen lugar. para comenzar. Una vez conocido el camino feliz, podemos definir los escenarios de mayor riesgo: casos que no sólo tienen un mal resultado sino que también tienen una buena posibilidad. de suceder.

Además de hacer preguntas al equipo del cliente como "¿Cuál es el peor ¿Qué podría pasar?", haga a los programadores preguntas como éstas: "¿Qué Cuáles son las condiciones de publicación de esta sección de código? ¿En qué se debe persistir? ¿la base de datos? ¿Qué comportamiento deberíamos buscar en el futuro? Especificar pruebas para cubrir resultados potencialmente riesgosos de una acción.

La historia de Lisa

Mi equipo considera los peores escenarios para ayudarnos a identificar las pruebas de los clientes. Por ejemplo, planeamos una historia para reescribir el primer paso de un asistente de creación de cuentas de varios pasos con un par de opciones nuevas. Nos hicimos preguntas como las siguientes: "Cuando el usuario envía esa primera página, ¿qué datos se insertan en la base de datos? ¿Se activan otras actualizaciones? ¿Necesitamos realizar una prueba de regresión en todo el proceso de configuración de la cuenta? ¿Qué pasa con las actividades que la cuenta de usuario podría realizar después de la configuración? Es posible que necesitemos probar todo el ciclo de vida de la cuenta. No tenemos tiempo para probar más de lo necesario, por lo que las decisiones sobre qué probar son fundamentales. Las pruebas adecuadas nos ayudan a mitigar el riesgo que conlleva el cambio.

—Lisa

Los programadores pueden identificar partes frágiles del código. ¿La historia involucra ¿Unir código heredado con una nueva arquitectura? ¿El código es modificado interactuar con otro sistema o depender de software de terceros? Por discutir impactos potenciales y áreas de riesgo con programadores y otros equipos miembros, podemos planificar actividades de prueba apropiadas.

Hay otro riesgo. Podríamos involucrarnos tanto en escribir casos de prueba detallados frente que el equipo pierde el bosque entre los árboles; es decir, podemos olvidarnos de lo grande imagen mientras nos concentraremos en detalles que podrían resultar irrelevantes.

Peligro: olvidar el panorama general

Es fácil caer en el hábito de probar sólo historias individuales o basar las pruebas en lo que el programador le dice sobre el código. Si encuentra problemas de integración entre historias al final del lanzamiento o si faltan muchos requisitos una vez "terminada" la historia, tome medidas para mitigar este peligro.

Considere siempre cómo cada historia individual impacta otras partes del sistema. Utilice datos de prueba realistas, utilice ejemplos concretos como base de sus pruebas y tenga muchas discusiones en la pizarra (o su equivalente virtual) para asegurarse de que todos comprendan la historia. Asegúrese de que los programadores no comiencen a codificar antes de que se escriban las pruebas y utilice pruebas exploratorias para encontrar brechas entre las historias.

Recuerde el objetivo final y el panorama general.

Como equipo ágil, trabajamos en iteraciones cortas, por lo que es importante programar el tiempo dedicado a escribir pruebas antes de comenzar. Después de completar cada iteración, tome Ha llegado el momento de evaluar si habría sido útil disponer de más detalles desde el principio. Eran ¿Hay suficientes pruebas para mantener al equipo en el buen camino? ¿Hubo mucho tiempo perdido? ¿Porque se malinterpretó la historia? El equipo de Lisa ha encontrado que lo mejor es escribir pruebas de historias de alto nivel antes de codificar, para escribir casos de prueba detallados una vez codificada comienza y luego realizar pruebas exploratorias en el código a medida que se entrega en orden para brindarle al equipo más información y ayudar a realizar los ajustes necesarios.

Janet trabajó en un proyecto que requería cálculos muy intensivos. El tiempo dedicado a crear ejemplos y pruebas detallados antes de comenzar la codificación, para garantizar que los cálculos se realizaron correctamente, fue un tiempo bien invertido. Comprender el dominio y el impacto de cada historia es fundamental para evaluar el riesgo y elegir la estrategia de mitigación correcta.

Si bien las pruebas empresariales pueden ayudar a mitigar los riesgos, se recomiendan otros tipos de pruebas. también crítico. Por ejemplo, muchos de los problemas más graves suelen descubrirse durante las pruebas exploratorias manuales. Rendimiento, seguridad, estabilidad y La usabilidad también son fuentes de riesgo. Las pruebas para mitigar estos otros riesgos se analizan en los capítulos sobre los cuadrantes 3 y 4.

Experimente y encuentre formas en que su equipo pueda equilibrar el uso de detalles iniciales y mantenerse enfocado en el panorama general. La belleza de las iteraciones cortas y ágiles es que tenga oportunidades frecuentes para evaluar cómo está funcionando su proceso para que puedas realizar mejoras continuas.

PRUEBAS Y AUTOMATIZACIÓN

Cuando los programadores de un equipo ágil se preparan para realizar un desarrollo basado en pruebas, utilizan las pruebas comerciales de la historia para saber qué codificar. Trabajar a partir de pruebas significa que todos piensan en la mejor manera Diseñar el código para facilitar las pruebas. Las pruebas comerciales del Cuadrante 2 se expresan como pruebas automatizadas. Deben entenderse claramente, fácil de ejecutar y proporciona retroalimentación rápida; de lo contrario, no se acostumbrarán.

Es posible escribir scripts de prueba manuales para que los programadores los ejecuten antes de verificar el código para asegurarse de que cumplen con las condiciones del cliente, pero no es realista esperar que lleguen a tanto. problemas por mucho tiempo. Cuando es necesario entregar un valor comercial significativo cada dos semanas o cada 30 días, la información tiene que ser directa y automática. Los equipos ágiles sin experiencia podrían aceptar la necesidad de impulsar la codificación con pruebas automatizadas en el nivel de prueba del desarrollador más fácilmente que en la prueba del cliente.

nivel. Sin embargo, sin las pruebas de los clientes, los programadores tienen mucho Es más difícil saber qué pruebas unitarias escribir.

La Parte IV,
“Automatización de
pruebas”, lo guiará
a medida que desarrolle
una estrategia de automatización con deuda técnica.

Cada equipo ágil debe encontrar un proceso de redacción y automatización de pruebas comerciales que impulsen el desarrollo. Los equipos que automatizan sólo pruebas orientadas a la tecnología descubren que pueden tener código libre de errores que no hace lo que el cliente quiere. Los equipos que no automaticen ninguna prueba se anclarán en una estrategia de automatización con deuda técnica.

El cuadrante 2 contiene muchos tipos diferentes de pruebas y actividades. Nosotros necesitamos las herramientas adecuadas para facilitar la recopilación, discusión y comunicación de ejemplos y pruebas. Herramientas sencillas como papel o una pizarra funcionan bien para recopilar ejemplos si el equipo está ubicado en el mismo lugar. Herramientas más sofisticadas ayudan a los equipos a escribir pruebas comerciales que guíen el desarrollo en un formato ejecutable y automatizable. En el próximo capítulo, veremos los tipos de herramientas necesarias para obtener ejemplos y escribir, comunicar y ejecutar pruebas comerciales que apoyan al equipo.

RESUMEN

En este capítulo, analizamos formas de apoyar al equipo durante el proceso de codificación con pruebas comerciales.

En el desarrollo ágil, los ejemplos y las pruebas empresariales, en lugar de los documentos de requisitos tradicionales, le dicen al equipo qué código escribir.

Trabajar en pequeñas porciones de funcionalidad, en iteraciones cortas, brinda a los clientes la oportunidad de ver y utilizar la aplicación y ajustar sus requisitos según sea necesario.

Un área importante en la que contribuyen los evaluadores es ayudar a los clientes a expresar condiciones de satisfacción y crear ejemplos de comportamiento deseado y no deseado para cada historia.

Haga preguntas abiertas para ayudar al cliente a pensar en todas las funciones deseadas y evitar ocultar suposiciones importantes.

Ayude a los clientes a lograr un consenso sobre el comportamiento deseado para historias que se adapten a los distintos puntos de vista de las diferentes partes del negocio.

Ayudar a los clientes a desarrollar herramientas (por ejemplo, una lista de verificación de historias) para expresar información como las condiciones de satisfacción empresarial.

Los equipos de desarrollo y de cliente deben pensar en todas las partes de la aplicación a las que afecta una historia determinada, teniendo en cuenta la funcionalidad general del sistema.

Trabaje con su equipo para dividir los conjuntos de funciones en historias y rutas pequeñas y manejables dentro de las historias.

Siga un patrón de “escribir prueba, escribir código, ejecutar pruebas, aprender” paso a paso, basándose en cada paso por la funcionalidad.

Utilice pruebas y ejemplos para mitigar los riesgos de perder funcionalidad o perder de vista el panorama general.

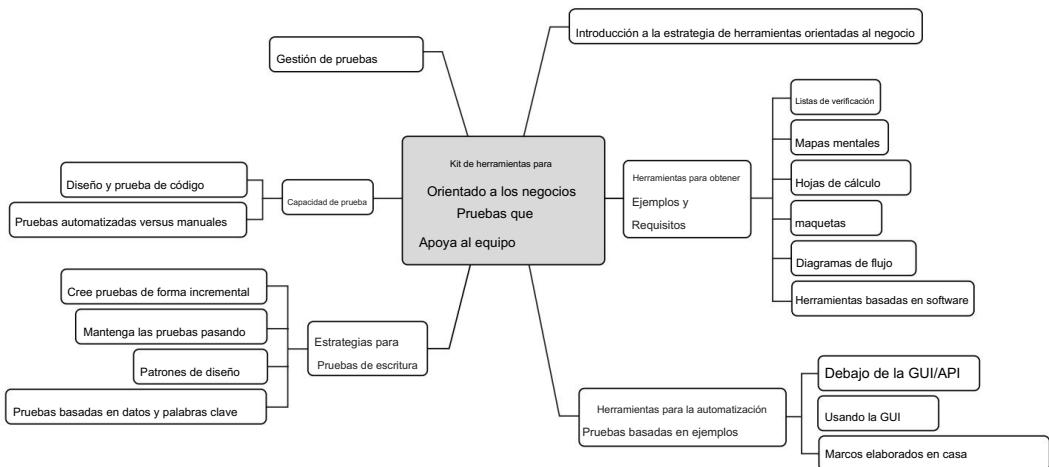
Impulsar la codificación con pruebas empresariales hace que el equipo de desarrollo sea constantemente consciente de la necesidad de implementar una aplicación comprobable.

Las pruebas empresariales que respaldan al equipo deben automatizarse para obtener comentarios rápidos y sencillos, de modo que los equipos puedan ofrecer valor en iteraciones cortas.

Esta página se dejó en blanco intencionalmente.

Capítulo 9

KIT DE HERRAMIENTAS PARA LOS NEGOCIOS PRUEBAS QUE APOYAN AL EQUIPO



En el capítulo anterior, hablamos sobre cómo abordar las pruebas funcionales o comerciales para apoyar al equipo en su esfuerzo por crear el software adecuado. En este capítulo, examinaremos algunas de las herramientas que puede utilizar para ayudar a su equipo a tener éxito con las pruebas del Cuadrante 2.

ESTRATEGIA DE HERRAMIENTA DE PRUEBA DE ORIENTACIÓN AL NEGOCIO

¿Cómo capturamos las pruebas empresariales que ayudan a los programadores a saber qué codificar? Las conversaciones cara a cara entre programadores y clientes suelen ser la mejor manera, pero incluso cuando los clientes forman parte de su equipo, no tienen todo el día para pasar el rato con los programadores y explicarles las funciones.

Si algún cliente o miembro del equipo de desarrolladores se encuentra en diferentes ubicaciones, es posible que no sea factible mantener conversaciones improvisadas en el pasillo. Además, seis meses

A partir de ahora, es posible que queramos una manera de recordar por qué codificamos una parte de la funcionalidad de cierta manera. Si algunos de los miembros de nuestro equipo se encuentran en diferentes ubicaciones, Definitivamente vamos a necesitar alguna forma de compartir información electrónicamente.

A medida que el desarrollo ágil ha ganado popularidad, tenemos cada vez más herramientas para ayudarnos a capturar ejemplos y usarlos para escribir pruebas ejecutables. Las herramientas disponibles están cambiando demasiado rápido para que podamos incluir un inventario de ellos en este libro, pero podemos ofrecer algunos ejemplos de herramientas y algunas estrategias para usar ellos para ayudar a proporcionar pruebas comerciales que respalden el desarrollo de nuevas historias por parte del equipo. Algunas de las herramientas que analizamos aquí no son nuevas ni específicas al desarrollo ágil, pero funcionan bien en un proyecto ágil.

Para obtener más información sobre un enfoque general para la automatización de pruebas, consulte el Capítulo 14, "Una automatización ágil de pruebas".

Estrategia."

Su estrategia para seleccionar las herramientas que necesita debe basarse en la experiencia de su equipo, conjunto de habilidades, la tecnología que utiliza su aplicación, las prioridades de automatización de su equipo, las limitaciones de tiempo y presupuesto, y otras preocupaciones exclusivas de su situación. Su selección de una herramienta o herramientas no debe basarse en las últimas y La mejor herramienta ofrecida por un vendedor. Es posible que necesites muchas herramientas diferentes para resolver diferentes problemas.

Alentamos a los clientes a que se preparen con anticipación y estén listos para explique ejemplos para cada historia durante la planificación de la iteración. Los probadores están en buen estado. Posición para ayudar a los clientes a descubrir cómo proporcionar la cantidad adecuada de detalles, al comienzo de la iteración. Es difícil lograr el equilibrio adecuado.

La historia de Lisa

Poco después de que nuestro equipo decidiera utilizar FitNesse para especificar y automatizar pruebas comerciales, nuestro propietario del producto y yo intentamos hacer un buen uso de la nueva herramienta. Se avecinaba una epopeya extremadamente compleja. Pasamos muchas horas escribiendo casos de prueba detallados para reglas comerciales altamente complejas semanas antes de la iteración donde comenzó la primera historia de la epopeya. Nos sentimos bien al comenzar a desarrollar la nueva funcionalidad.

Cuando empezaron a trabajar en estas historias, los programadores se quejaron de que no podían obtener una visión general de estas pruebas detalladas. Las pruebas también se diseñaron de una manera que era incompatible con el diseño del código real. Terminé pasando horas refactorizándolos. No fue una completa pérdida de tiempo, porque al menos entendí bien las historias y teníamos varios casos de prueba que podríamos usar eventualmente, pero no era el enfoque correcto para nuestro equipo. La prueba y el error nos han demostrado que las pruebas de alto nivel combinadas con algunos ejemplos de comportamiento deseado y no deseado son la mejor manera para que los programadores sepan qué empezar a codificar.

—Lisa

Experimente con diferentes niveles de detalle inicial en casos de prueba para descubrir lo que funciona mejor para su equipo. Cualquiera que sea el nivel de detalle que busque, necesitará alguna forma de ayudar a los clientes a encontrar y expresar ejemplos del comportamiento deseado del sistema. En la siguiente sección, analizamos los tipos de herramientas que pueden hacer eso.

HERRAMIENTAS PARA OBTENER EJEMPLOS Y REQUISITOS

Como señalamos en el capítulo 8, las historias son sólo un punto de partida para una conversación prolongada sobre el comportamiento deseado. Tener historias del tamaño correcto donde la característica, el usuario y el propósito estén claramente establecidos nos da una idea. comenzar. No son muy detallados porque, como señala Mike Cohn [2004], es Es mejor posponer la recopilación de detalles hasta que la historia se incluya en una iteración. Recopilar detalles para una historia que tal vez nunca se incluya es un desperdicio de recursos. Nos gusta el patrón “rol, función, valor comercial” para las historias de usuarios que Mike Cohn describe en User Stories Applied, como en:

Como (rol), quiero (función) para que (valor comercial).

Este formato no funciona para todos, por lo que te animamos a experimentar. y vea qué funciona mejor en su situación. Independientemente de cómo se lean sus historias de usuario, necesita alguna forma de desarrollar esas historias con ejemplos y pruebas de cara al negocio que guían el desarrollo.

Una historia simple puede tener un impacto de amplio alcance, no sólo en la aplicación, sino en toda la organización, sus clientes, sus asociados, proveedores o socios. Si cambiamos una API, debemos notificar a cualquier cliente o proveedor que podría estar usándolo. Si planeamos un cambio en la interfaz de usuario, queremos, o incluso podríamos estar obligados contractualmente, a avisar con cierta antelación a los usuarios. Cuentos puede afectar inquietudes legales o impactar los informes externos. Nuevas características a menudo Significa documentación nueva o actualizada. Por supuesto, la funcionalidad modificada es Es probable que afecte a otras partes del sistema.

El equipo de desarrollo de software, incluidos los evaluadores, debe ayudar al cliente a capturar y comunicar todos los requisitos relacionados con cada historia. o tema. Desarrollar nuevas funciones, sólo para que se les impida publicarlas. por razones legales o porque un socio comercial no fue informado a tiempo, es un frustrante pérdida de tiempo (¡pregúntale a Lisa!). El desarrollo Lean nos enseña a Evitemos desperdicios mientras desarrollamos software.

¿Qué herramientas pueden ayudarnos a ilustrar el comportamiento deseado con ejemplos, realizar una lluvia de ideas? implementaciones potenciales y efectos dominó, y crear requisitos que podamos convertirse en pruebas? Algunos ejemplos son:

- Listas de verificación
- Mapas mentales
- Hojas de cálculo
- maquetas
- Diagramas de flujo
- Herramientas basadas en software

La lista incluye una serie de herramientas simples que no son exclusivas de las pruebas ágiles. pero eso no debe descuidarse. En el desarrollo ágil, las soluciones simples son normalmente lo mejor. Veámoslos con más detalle.

Listas de verificación

Las listas de verificación son una forma para que los propietarios de productos se aseguren de evaluar correctamente y comunicar todos los aspectos de una historia. El propietario del producto de Lisa's.

El equipo, Steve Perkins, ideó su propia "lista de verificación de historias" para asegurarse de que y las partes interesadas piensan en todo lo afectado por la historia. Creó una plantilla en la wiki del equipo para este propósito. La lista de verificación específica

Condiciones de satisfacción: lo que la empresa necesita de la historia. También incluye impactos en funciones existentes como el sitio web, documentos, formularios administrativos, estados de cuenta y otros componentes del sistema y

la operación diaria del negocio. La lista de verificación garantiza que el equipo no omitir requisitos como migración de datos, notificaciones, consideraciones legales, y comunicaciones a proveedores y socios comerciales porque olvidaron para considerarlos. La Figura 9-1 muestra una lista de verificación de historias de muestra.

Mapas mentales

Los mapas mentales son una forma sencilla pero eficaz de buscar ideas que puedan No se te ocurrirá en una simple sesión de lluvia de ideas. Los mapas mentales son diagramas. creado para representar conceptos, palabras o ideas vinculadas a un concepto clave central. Usamos mapas mentales para organizar este libro.

Realmente no importa si compras una herramienta como la que usamos nosotros. o dibujar en una pizarra o en una hoja de papel grande. El efecto es el mismo. Mente Los mapas le permiten generar ideas y trabajar de una manera coherente con la forma en que piensas sobre los problemas.

As a plan sponsor I can have an updated summary statement so that I can:

SATISFACTION CONDITIONS

- 1. separate (smart) dividends/interest detail on summary and activity;**
- 2. separate (smart) concessions detail on summary and activity;**
- 3. add balance by fund to investment performance page;**
- 4. update font, point size, style and coloring (see attached prototype)**

IMPACT

New Product Website	
Legal contracts	
Documents - Invoices	
Documents - Plan	
Administrative Forms	
Reports - Existing or new	
Account Statements	Revised, smart to not show new details prior to 01/01/2008;

U/I MOCKUPS PREPARED, REVIEW AND FINALIZED

Screens	see attached docs
Help Text Written	

ADDITIONAL ISSUES

Data migration	
Impact on Processing	
Vendor APIs	
Impact if incomplete	
Audit tracking	

TEST CASE OUTLINES

- 1. test smartness of divs/int and concessions;**
- 2. further systematic testing;**
- 3. test statements with period dates prior to 01/01/2008**

COMMUNICATION (INTERNAL/EXTERNAL)

Pieces Written	Notify partners, plan sponsors
Delivery method	

Figura 9-1 Lista de verificación de historias de muestra

¿Qué tal un ejemplo? Estamos discutiendo la historia que se muestra en la Figura 9-2.

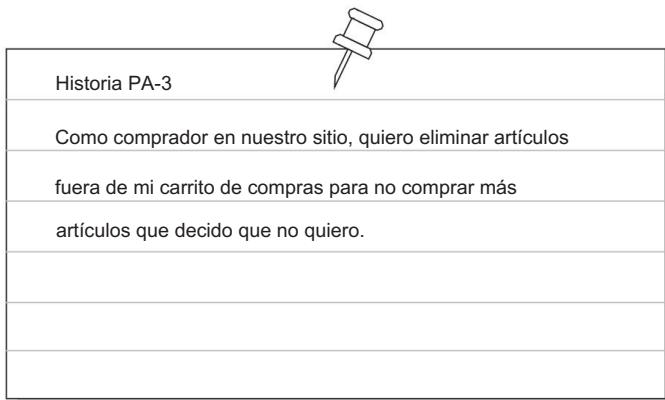


Figura 9-2 Historia de eliminación del carrito de compras

Nos reunimos alrededor de la pizarra y comenzamos a hacer preguntas. ¿Dónde debería el elemento eliminado ir? ¿Deberían guardarse para su aprobación posterior o deberían simplemente desaparecer? ¿Cómo debería verse la pantalla después de eliminar un elemento? Figura 9-3 muestra un ejemplo del tipo de mapa mental que podríamos dibujar en una pizarra.

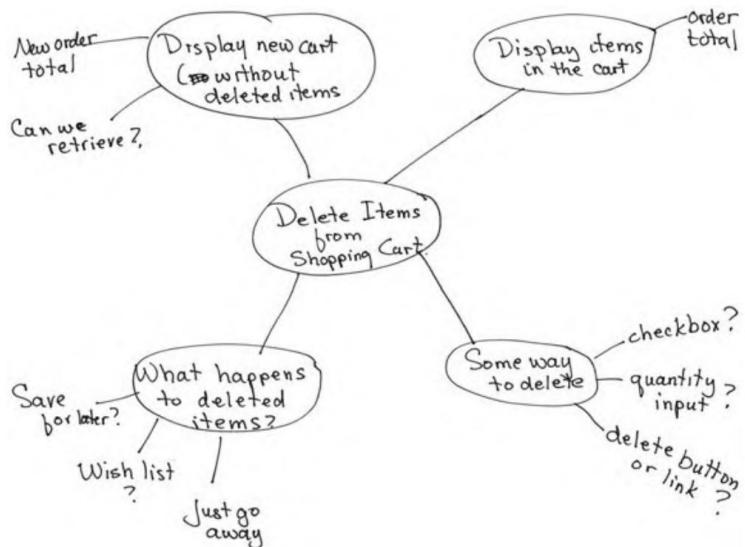


Figura 9-3 Ejemplo de mapa mental para la historia de eliminación del carrito de compras

Hojas de cálculo

Cuando sea posible, las herramientas para especificar pruebas orientadas al negocio deben encajar bien con el dominio de tu negocio. Por ejemplo, las hojas de cálculo son ampliamente utilizadas por las empresas de servicios financieros, por lo que para un proyecto en el área de servicios financieros es necesario

Tiene sentido utilizar hojas de cálculo para definir ejemplos de la funcionalidad que una historia debería entregar.

Los clientes pueden escribir algunos casos de prueba de alto nivel para ayudar a completar una historia antes, hasta el inicio de la iteración, posiblemente utilizando algún tipo de lista de verificación. Algunos equipos de clientes simplemente escriben un par de pruebas, tal vez un camino feliz y uno negativo. prueba, en el reverso de cada tarjeta de historia. Algunos escriben ejemplos más detallados en hojas de cálculo o cualquier formato con el que se sientan cómodos trabajando.

Steve Perkins, el propietario del producto del equipo de Lisa, a menudo ilustra complejos cálculos y algoritmos en hojas de cálculo, que el equipo puede convertir en pruebas más tarde. La Figura 9-4 muestra una de sus hojas de trabajo, que realiza cálculos sobre los valores de entrada para producir los valores en las columnas ADR y ACR. Este formato es fácil de ingresar en un marco de prueba automatizado (consulte

Figura 9-8 para el ejemplo de FitNesse correspondiente).

Mire las herramientas que ya utilizan sus expertos comerciales y vea si pueden adaptarse para documentar ejemplos del comportamiento deseado de las funciones para ayudar al equipo de desarrollo a comprender mejor la historia.

Janet ha trabajado con varios equipos que han utilizado hojas de cálculo como entrada para sus pruebas de ajuste. Esto permite a los clientes trabajar en una herramienta que les resulta familiar, pero no desperdicias esfuerzo en traducirlos a una herramienta de automatización.

		Testing Year	Testing	Testing				
		Eligible	Year	Year	Catch-up			
EE	HCE	Compensation	Deferral	Match	Deferral	ADR	ACR	
E-1001	Y	102,500.00	16,000.00	16,000.00	3,000.00	12.68	15.61	
E-1002	Y	102,500.00	13,000.00	13,000.00	-	12.68	12.68	
E-1003	Y	30,000.00	7,500.00	7,500.00	-	25.00	25.00	
E-1004	Y	30,000.00	3,000.00	3,000.00	-	10.00	10.00	
E-1005	Y	40,000.00	8,000.00	8,000.00	-	20.00	20.00	
E-1006	Y	150,000.00	13,000.00	13,000.00	-	8.67	8.67	
E-1007	Y	100,000.00	-	-	-	-	-	
						12.72	13.14	

Figura 9-4 Ejemplo de hoja de cálculo del propietario del producto

maquetas

Véase el capítulo 8 para Gerard Meszaros.
descripción del uso de prototipos de papel y pruebas del Mago de Oz.

Las maquetas pueden adoptar muchas formas. Los prototipos en papel son una forma sencilla pero eficaz. forma de probar cómo funcionarán las pantallas juntas. Dibujar en una pizarra puede lograr el mismo objetivo, pero no se puede compartir. Las capturas de pantalla de aplicaciones existentes pueden formar la base de una discusión sobre cómo agregar una nueva. característica y dónde encajará en la interfaz de usuario. Es posible que haya utilizado herramientas como éstas en otras metodologías de desarrollo. La gran diferencia en el desarrollo ágil es que creamos y discutimos las maquetas justo cuando estamos a punto de comenzar a escribir el código, en lugar de semanas o meses de antelación. Podemos estar seguros de que la maqueta representa lo que los clientes quieren en este momento.

La historia de Lisa

Usamos enfoques simples para crear maquetas para que no nos sintamos tentados a invertir tiempo codificando antes de terminar de trabajar en la maqueta. A menudo, dibujamos una interfaz de usuario o un flujo de trabajo en la pizarra y luego le tomamos fotos para cargarlas en la wiki de nuestro equipo para que nuestro miembro remoto del equipo también pueda verlo. En otras ocasiones, un cliente o el propietario de nuestro producto dibuja la maqueta en papel o modifica una página de interfaz de usuario o un informe existente para mostrar lo que se debe agregar y cambiar. Las maquetas en papel se escanean y se publican en la wiki.

Una imagen vale más que mil palabras, incluso en el desarrollo ágil de software. Las maquetas muestran los deseos del cliente con mayor claridad que una narrativa. Proporcionan un buen punto focal para discutir el comportamiento deseado del código.

—Lisa

La Figura 9-5 muestra un ejemplo de una maqueta que el equipo de Lisa utilizó para simular. un nuevo informe, simplemente marcando un informe existente que sea similar.

No es necesario que las maquetas sean sofisticadas o bonitas, ni que requieran mucho tiempo para crearlas. Deben ser comprensibles tanto para el cliente como para el desarrollador. equipos.

Diagramas de flujo Las

herramientas de diagramación simples son útiles, ya sea que el equipo esté ubicado en el mismo lugar o no. A menudo es una buena idea capturar de una forma más permanente un flujo de trabajo o un árbol de decisiones elaborado durante una discusión. Los diagramas de flujo pueden convertirse en base de un escenario de usuario que podría ayudarle a unir dos o tres historias de usuario. Veamos nuevamente la historia del pedido de envío que presentamos en Capítulo 8 (ver Figura 9-6).

~~Default Investment Employees~~

~~Auto Enrolled Employees~~

The following employees have not enrolled in the plan and are subject to the plan's auto-enrollment features. Use this information to update your payroll records for these employees. Please see the Plan Highlights for information on the auto-enrollment schedule.

Name	SSN	Entry Date	Year of Participation	Current Deferral Rate	Next Change Date	Next Deferral Rate	Normal Retirement Year
[REDACTED]	XXX-XX-8021	05-07-2008	0	0%	05-07-2008	5%	QDIA Fund
[REDACTED]	XXX-XX-4955	05-16-2008	0	0%	05-16-2008	5%	
[REDACTED]	XXX-XX-3175	08-30-2007	1	6%	08-30-2008	5%	
[REDACTED]	XXX-XX-3577	11-22-2007	1	6%	11-22-2008	6%	

Copyright © 2000-2008, Prime k

Sort by QDIA
 and then alphabetic with that
 part
 -if Balanceed the just by all same
 -if Target then tells ER which Fund sheet to give
 to each ee (Fund sheet to EE's)

Figura 9-5 Modelo de informe de muestra



Historia PA-1
Como comprador de Internet en LotsO'Stuff.xx,
Quiero envío gratis cuando mi pedido supere el límite gratuito.
umbral de envío, para que pueda aprovechar
de pedir más a la vez.

Figura 9-6 Historia de los gastos de envío

La Figura 9-7 muestra un diagrama de flujo muy simple de un proceso de decisión sobre si el pedido de un cliente es elegible para envío gratuito en función de un monto umbral de pedido. Debido a que analizamos esta historia con nuestro cliente, descubrimos que el pedido del cliente no solo debe exceder un monto umbral en dólares, sino que también debe enviarse a una sola dirección y debe pesar menos que el umbral de peso de envío. Si se cumplen todas estas condiciones, el pedido del cliente se enviará gratis; de lo contrario, el cliente deberá seleccionar en la página “elegir opciones de envío”.

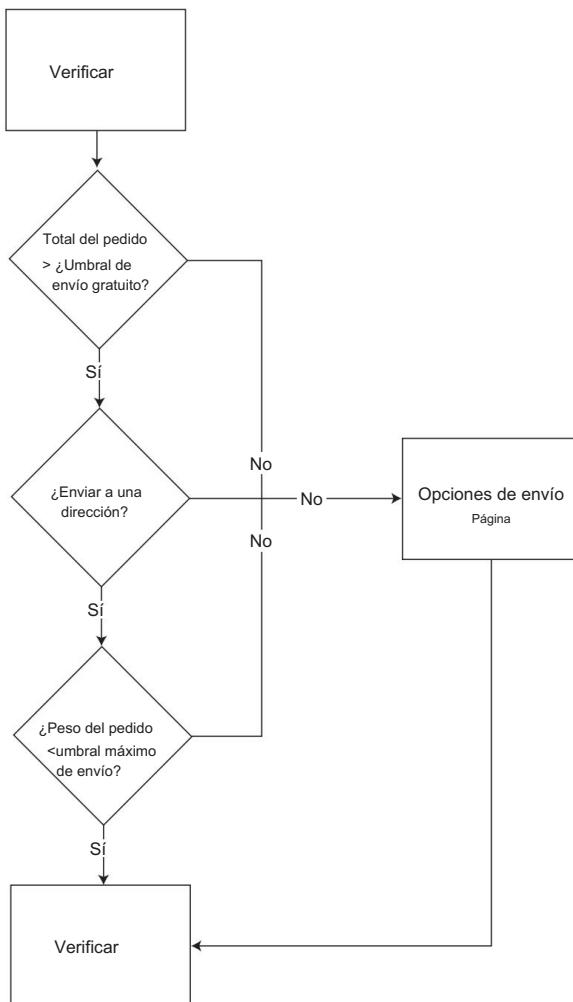


Figura 9-7 Diagrama de flujo para calificar para la opción de envío gratuito

Los elementos visuales como diagramas de flujo y mapas mentales son buenas maneras de describir una descripción general de la funcionalidad de una historia, especialmente si están dibujadas por un grupo de clientes, programadores y evaluadores. En el desarrollo ágil, creamos estos diagramas ya que estamos a punto de comenzar a escribir pruebas y código. De estos, el equipo puede comenzar inmediatamente a profundizar en los requisitos detallados.

Herramientas basadas en software

Si estamos en una ubicación diferente a la de nuestros clientes, necesitamos herramientas que nos ayuden conversar con ellos. Los equipos distribuidos nos dicen que compartir escritorio es la herramienta número uno que les ayuda a trabajar en ubicaciones separadas. ventanas NetMeeting y VNC son ejemplos de herramientas que permiten que dos miembros del equipo en diferentes ubicaciones realicen pruebas en pares. Herramientas de videoconferencia como WebEx y Skype Permitir la colaboración y demostraciones entre equipos remotos y clientes. En línea pizarras blancas como Scriblink y herramientas de pizarra interactiva como Mimeo Facilitar debates en pizarras distribuidas.

Cada vez hay más herramientas disponibles para uso directo de propietarios de productos y expertos empresariales, y muchos equipos desarrollan las suyas propias. Herramientas tales como Fit (Marco para Pruebas Integradas) y FitNesse fueron diseñados para facilitar la colaboración y comunicación entre el cliente y los equipos de desarrollo. Estamos escuchando acerca de más equipos donde los clientes realmente escribir las pruebas en una herramienta como esas.

Notas de un equipo distribuido

Pierre Veragen y Erika Boyer de iLevel de Weyerhaeuser nos dijeron que cada iteración comienza cuando todos los miembros del equipo escriben pruebas de aceptación. Eso es cómo inician su planificación de iteraciones. Lo más interesante es el hecho de que los propietarios de sus productos, que son ingenieros mecánicos, escriben ellos mismos las pruebas de FitNesse. Pierre explica que una ventaja de una herramienta como FitNesse es la capacidad de utilizar su propio lenguaje de dominio en las pruebas de FitNesse. No importa lo que terminen eligiendo como interfaz de usuario. Pueden probar todos sus cálculos complejos en las pruebas.

Con este proceso, se pueden escribir pruebas antes de escribir el código de prueba o el sistema bajo prueba. Es un verdadero desarrollo basado en pruebas. Pueden seguir cambios de comportamiento y correcciones de errores.

Algunos equipos construyen sus propios marcos que permiten a los clientes, empresas analistas y evaluadores para documentar ejemplos que se pueden convertir directamente en pruebas ejecutables. A menudo se basan en herramientas de código abierto como xUnit, Fit, Selenio y Watir. Nos gusta este enfoque porque ahorra tiempo y

recursos. Cuando entrega código listo para producción en iteraciones cortas, necesita un proceso optimizado.

Las herramientas de foros en línea son una buena alternativa a las conversaciones por correo electrónico para continuar discusiones sobre características o inquietudes técnicas, especialmente para equipos que No os sentéis todos juntos. Los correos electrónicos a menudo se pierden o se pierden, las personas deben recordar elegir "Responder a todos" y puede resultar difícil reunir los detalles del correo electrónico. discusión más adelante. El equipo de Lisa utiliza un foro en línea para obtener opiniones sobre diferentes herramientas, proponer diferentes comportamientos para las funciones y llevar a cabo debates filosóficos, como por ejemplo si se debe realizar un seguimiento de los defectos.

Encontrar las herramientas electrónicas adecuadas es particularmente vital para los equipos distribuidos. La mensajería instantánea, el teléfono, VoIP y Skype nos ayudan a comunicarnos, pero Carecen del componente visual. Algunos equipos globales piden a sus miembros que reunirse en horarios no estándar para poder tener conversaciones en tiempo real, pero Los marcos para la comunicación escrita y visual siguen siendo críticos.

Los wikis son una herramienta común utilizada para mejorar la comunicación y registrar discusiones y decisiones. Los wikis permiten a los usuarios editar el contenido de una página web en un sitio web. navegador. Los usuarios pueden agregar hipervínculos y crear fácilmente nuevas páginas. Puede cargar maquetas, muestras e imágenes de dibujos de pizarra y crear hacerlos fácilmente visibles en las páginas Wiki. La organización jerárquica puede conseguir Es complicado de mantener, pero hay muchos software wiki de proveedores y de código abierto. Hay paquetes disponibles que facilitan la administración de la gestión de su base de conocimientos y del intercambio de información. Si su base de conocimientos wiki ha crecido hasta el punto donde es difícil encontrar algo, contrate a un redactor técnico para transformarlo en documentación organizada y utilizable.

Las herramientas comerciales y de código abierto brindan formas de permitir que los equipos colaboren en requisitos y casos de prueba en línea. No podemos enfatizar lo suficiente la necesidad de usted. identificar herramientas que podrían ser útiles, experimentar con ellas durante algunas iteraciones y decidir qué tan bien funcionan para usted. Las necesidades de tu equipo cambiarán con el tiempo, así que esté siempre abierto a probar nuevas técnicas y marcos.

Estas herramientas ayudan a crear la conversación sobre la historia. Con estas técnicas y toda la conversación en tiempo real y el intercambio visual que podamos gestionar, podemos definir el producto adecuado desde el principio.

HERRAMIENTAS PARA LA AUTOMATIZACIÓN DE PRUEBAS BASADAS EN EJEMPLOS

¿Qué pasa con las herramientas de prueba? Nos gusta la colaboración inherente a herramientas como En forma y en forma. Sin embargo, en nuestra opinión, cualquier herramienta que consiga probadores y profesionales

gramáticos, programadores y clientes, y evaluadores y clientes hablando es uno genial. Conocemos equipos donde los clientes realmente escriben pruebas en Fit, Fit-Nesse, Expect u otras herramientas. Esto funciona cuando la herramienta se ha configurado en un manera que sea clara para todos los que escriben pruebas, con el lenguaje de dominio fácil entender y los accesorios apropiados proporcionados.

Herramientas para probar por debajo del nivel de GUI y API

Hay una multitud de herramientas de código abierto que le permiten probar debajo del GUI o en la capa API. Enumeramos solo algunos, pero su equipo deberá determine la herramienta adecuada para usted.

Herramientas de prueba a nivel unitario

Algunos equipos utilizan las herramientas xUnit, como JUnit o NUnit, para la gestión empresarial. pruebas, así como pruebas tecnológicas. Si los evaluadores y los clientes se sienten cómodos con estas herramientas y proporcionan todas las pruebas funcionales detrás de la GUI necesarias, están bien. Para que estas herramientas sean más amigables para el cliente, los equipos podrían crear un marco además de las herramientas a nivel de unidad que

Los evaluadores y los clientes pueden utilizar para especificar pruebas.

Janet ha trabajado en un par de aplicaciones como esa. uno era un mensaje sistema de manejo que se estaba implementando en una organización. Los programadores utilizaron JUnit para todas las pruebas de integración y componentes. Ellos construyeron un marco de prueba de carga que podría hacer uso de las pruebas JUnit, por lo que no se necesitaban otras herramientas de prueba. La interfaz gráfica de usuario era tan pequeña que Janet pudo pruébalo manualmente. En este caso no tenía sentido automatizar las pruebas de la GUI.

Las herramientas de desarrollo basado en el comportamiento (BDD) también son adecuadas para este propósito. porque utilizan un lenguaje más natural para especificar las pruebas. El desarrollo basado en el comportamiento es una variación del desarrollo basado en pruebas, iniciado por Dan North [2006], y desarrollado por muchos otros. Está relacionado con el dominio impulsado diseño, centrándose en el dominio más que en la tecnología, e impulsando el diseño con un modelo. En lugar de la palabra "probar" o "afirmar", BDD utiliza la palabra palabra "debería". Al pensar en términos de comportamiento, es natural escribir especificaciones antes que el código. Las especificaciones de prueba utilizan un lenguaje específico de dominio para Proporcionar pruebas que los clientes puedan leer pero que también puedan automatizarse fácilmente.

Algunas de las muchas herramientas BDD disponibles al momento de escribir este artículo incluyen easyb y JBehave para la plataforma Java, NBehave y NSpec para .NET y RSpec para Rubí. Estas herramientas, al igual que las herramientas XUnit, están destinadas a que las utilicen programadores para codificación de guía, pero también se pueden usar para expresar pruebas comerciales que impulsan desarrollo, involucrando más estrechamente a los clientes en el proceso de desarrollo.

Desarrollo impulsado por el comportamiento

Andrew Glover, presidente de Stelligent Incorporated y autor de libros que incluyen y detrás de una de las [Introducción a las Pruebas y el Desarrollo de Software](#), Java Pruebas Patrones, explica el pensamiento-

afirmarEquals(42.50, orden.precio(), 0.0). Sin examinar el contexto en el que aparece esta afirmación, este código resulta algo incomprensible. Ahora imagine que ni siquiera lee el código; es decir, que es una parte interesada que solicita (en realidad) nuevas funciones. La declaración de código anterior ~~también~~ podría ser farsi (suponiendo que en realidad no puedes leer farsi!).

order.price().debería ser 42,50. Si bien el contexto en el que aparece esta declaración aún está ausente, esta línea de código es un poco más coherente. De hecho, se lee como una oración normal (y esta vez no se requiere conocimiento de farsi!). Las partes interesadas, en este caso, podrían entender este código si decidieran leerlo; Además de eso, resulta que esta línea de código coincide esencialmente con lo que solicitaron en primer lugar. Esta línea de código también describe el comportamiento de una manera más literal: el código utiliza una frase cotidiana normal como debería ser, que es claramente diferente a la frase afirmada previamente escrita .

Ambas líneas de código de los párrafos anteriores transmiten el mismo significado y, de hecho, validan el mismo requisito; sin embargo, la última se acerca muchísimo a aprovechar el lenguaje del cliente. Este es un punto fundamental de la noción de desarrollo impulsado por el comportamiento, que se esfuerza por validar de manera más apropiada un sistema de software pensando en términos del término “debería” en lugar de probar. De hecho, al centrarse en el comportamiento y modelarlo estrechamente según lo que piden las partes interesadas, el desarrollo impulsado por el comportamiento converge en la idea de documentación ejecutable. De hecho, al aprovechar el lenguaje de una parte interesada, hay un menor desajuste de impedancia entre lo que quiere y lo que finalmente recibe; Además, emplear el lenguaje de las partes interesadas facilita un nivel más profundo de colaboración entre todas las partes. Escuche cómo podría desarrollarse una conversación:

Parte interesada: Para el próximo lanzamiento de nuestra tienda en línea, nuestros clientes de nivel Gold deberían recibir un descuento cuando realicen una compra.

Desarrollador: ¿Qué tipo de descuento? ¿Qué criterios deben cumplir para recibarlo?

Interesado: Cuando tenga al menos \$50 dólares en su carrito de compras.

Desarrollador: ¿El descuento aumenta según el monto o es fijo independientemente del valor del carrito de compras?

Parte interesada: Buena pregunta: el descuento se fija en el 15% independientemente del precio. Entonces, dado un cliente de nivel Gold, cuando el carrito de compras sume \$50 o más, debería recibir un descuento del 15% sobre el precio total.

La última declaración de la parte interesada es clave: observe cómo se ha especificado el requisito y los medios para validarla. De hecho, la parte interesada esencialmente ha narrado un escenario específico en una historia más amplia relacionada con los descuentos.

Ante este escenario, un desarrollador puede aceptar los comentarios de las partes interesadas: palabra por palabra y ejecutarlas. Por ejemplo, un marco de desarrollo basado en el comportamiento, denominado easyb, facilita la validación del sistema a través de un lenguaje de dominio específico que admite historias y escenarios.

Por ejemplo:

```
escenario "Cliente de nivel Gold con $50 en el carrito de compras", {  
    dado      "un cliente de nivel Oro"  
    "cuando \"su carrito de compras suma $50 o más"  
    entonces "deberían recibir un 15% de descuento sobre el precio total"  
}
```

Por supuesto, este escenario particular en realidad no hace nada (aparte de capturar los requisitos de las partes interesadas, ¡lo cual sigue siendo bastante importante!); en consecuencia, se considera pendiente. Este estado por sí solo transmite información valiosa: las partes interesadas pueden, en primer lugar, ver sus palabras como un medio para validar sus solicitudes y, en segundo lugar, evaluar si se ha cumplido su requisito. Una vez implementado este escenario, puede, por supuesto, adoptar otros dos estados: éxito o fracaso, los cuales sirven para transmitir más información sobre el estado a las partes interesadas.

Ahora, con un escenario colaborativo definido, el desarrollo puede proceder a la implementación; lo bueno en este caso es que pueden implementar directamente el comportamiento deseado en línea con los requisitos, como este:

```
escenario "Cliente de nivel Gold con $50 en el carrito de compras", {  
    dado "un cliente de nivel Gold", {  
        cliente = nuevo Cliente Dorado()  
    }  
    cuando "su carrito de compras suma $50 o más", {  
        cliente.carritodecompras << nuevo artículo("widget", 50.00)  
    }  
    entonces "deberían recibir un descuento del 15 % sobre el precio total", customer.orderPrice.shouldBe  
        42,50  
    }  
}
```

¡Este escenario ahora es ejecutable dentro del contexto de la aplicación que sirve para validar! El escenario también aprovecha las palabras exactas del cliente; Es más, independientemente de la capacidad del cliente para leer el código, el código en sí aprovecha el lenguaje natural: `customer.orderPrice.shouldBe 42,50`.

Al aprovechar el idioma del cliente, éste tiene la capacidad de colaborar para facilitar la validación del sistema que desea crear.

Además, cuando el desarrollo aprovecha el lenguaje de las partes interesadas, existe un vínculo directo entre lo que las partes interesadas piden y lo que reciben.

Y ni siquiera necesitas entender farsi para ver el beneficio de eso.

Dos de las preguntas más comunes que nos hacen los nuevos equipos ágiles son: "¿Qué pasa con la documentación?" y "¿Cómo puede la automatización de pruebas mantenerse al día con el desarrollo en iteraciones de dos semanas?" Herramientas como easyb responden a esa pregunta con documentación ejecutable utilizando un lenguaje específico de dominio que todos los equipos de clientes y desarrolladores entienden.

El objetivo de las pruebas empresariales que respaldan al equipo es promover la comunicación y la colaboración entre clientes y desarrolladores, y permitir que los equipos entreguen valor real en cada iteración. Algunos equipos hacen esto mejor con herramientas de nivel unitario, y otros se adaptan mejor a herramientas de prueba de nivel funcional.

Herramientas de prueba funcional de capa API

Antes de que Lisa se uniera a su primer equipo ágil, probar "detrás de la GUI" era un concepto que sonaba bien, pero nunca había tenido la oportunidad de probarlo. Adaptar, y FitNesse, que se basa en Fit, son herramientas de prueba funcionales que crecieron de la necesidad de que el equipo del cliente pueda escribir y comprender el

Pruebas de cara al negocio que impulsan el desarrollo. Con estas herramientas, los equipos pueden probar Lógica empresarial sin involucrar la capa de presentación.

En forma y en forma. Fit (Marco para Pruebas Integradas) es un código abierto marco de prueba que promueve la colaboración, lo que lo convierte en una buena herramienta para ayudar a refinar los requisitos. Fit, invención de Ward Cunningham, ha contado con una ilustre lista de desarrolladores contribuyentes. Fit permite a los clientes, probadores y programadores utilicen ejemplos para especificar lo que esperan del sistema para hacer. Cuando se ejecutan las pruebas, Fit compara automáticamente las expectativas de los clientes con los resultados reales.

Con Fit, los clientes pueden brindar orientación utilizando su experiencia en la materia para definir los ejemplos con los que los programadores pueden codificar. Los programadores participan escribiendo los dispositivos que realizan las comprobaciones reales los ejemplos. Estos aparatos utilizan los datos especificados en los ejemplos para ejecutarse. con el programa real.

Las pruebas de ajuste se automatizan mediante dispositivos que pasan las entradas de prueba a la producción. código y luego acepta los resultados, que luego compara con los resultados esperados. Los resultados de las pruebas están codificados por colores, por lo que es fácil detectar una falla o excepción.



Obtenga más información sobre Fit en fit.c2.com.



Obtenga más información sobre FitNesse en www.fitnesse.org.

Las pruebas de ajuste están escritas como tablas HTML, pero los equipos pueden personalizar Fit para que las pruebas se pueda escribir en hojas de cálculo o en cualquier forma que los clientes, evaluadores y los analistas consideran utilizable.

FitNesse es un servidor web, un wiki y una herramienta de prueba de software que se basa en Adaptar. Desarrollada originalmente por Robert C. "Uncle Bob" Martin y Micah Martin, es una herramienta de código abierto con una comunidad de desarrolladores activa. El principal La diferencia entre FitNesse y Fit es que las pruebas de FitNesse están escritas en wiki, marcado en lugar de tablas HTML, lo que a algunos usuarios les resulta más fácil. También admite la creación de pruebas en hojas de cálculo e importarlas a las pruebas.

La Figura 9-8 muestra parte de la prueba de FitNesse que se creó a partir del ejemplo de Figura 9-4. Se agregaron más entradas para ejecutar el código de producción, pero Los datos de prueba esenciales provienen de la hoja de cálculo. Los resultados de la prueba están codificados por colores, verde cuando pasan, rojo cuando fallan.

Otro beneficio de una herramienta tipo Fit o FitNesse es que promueve la colaboración entre diferentes miembros del equipo para elaborar las pruebas adecuadas.

Add Employees

Build Employees Fixture																
userId	dob	doh	doe	dot	directOwnerPct	lookbackTotalOwnerPct	lookbackAnnualComp	annualComp	deferral	eligibleComp	match	add!				
1001	01-01-1950	01-01-1993	01-01-1994	null	0	0	101500.00	102500.00	18000.00	102500.00	160000.00	true				
1002	01-01-1960	01-01-1993	01-01-1994	null	4	3	102500.00	102500.00	13000.00	102500.00	13000.00	true				
1003	01-01-1960	01-01-1993	01-01-1994	null	5.01	5.01	30000.00	30000.00	7500.00	30000.00	7500.00	true				
1004	01-01-1960	01-01-1993	01-01-1994	null	10	10	20000.00	30000.00	30000.00	30000.00	30000.00	true				
1005	01-01-1960	01-01-1993	01-01-1994	null	8	0	4UUUU.UU	4UUUU.UU	8UUUU.UU	4UUUU.UU	8UUUU.UU	true				
1006	01-01-1960	01-01-1993	01-01-1994	null	5.01	0	150000.00	150000.00	13000.00	150000.00	150000.00	true				
1007	01-01-1960	01-01-1993	01-01-1994	null	0	0	100000.00	100000.00	0	100000.00	0	true				
1008	01-01-1960	01-01-1993	01-01-1994	null	0	0	40000.00	50000.00	3000.00	50000.00	30000.00	true				

OPERATE ON INPUT BY RUNNING ADP TEST

Operate Adp Test Fixture	
operate!	
true	

MAKE ASSERTIONS ABOUT ADP TEST RESULTS

Check Employee Fixture				
userId	isHce?	isEligible?	adr?	act?
1001	true	true	12.682927	15.61
1002	true	true	12.682927	12.68
1003	true	true	25.00	25
1004	true	true	10.00	10
1005	true	true	2U.UU	2U
1006	true	true	8.000007	8.67
1007	true	true	0	0
1008	false	true	6	6

Figura 9-8 Prueba automatizada de FitNesse del ejemplo del cliente

para guiar el desarrollo. Los clientes, programadores, evaluadores y otros trabajan juntos para especificar y automatizar las pruebas.

Pruebas de servicios web. Los servicios web son solo otra forma de API que permite que otras aplicaciones accedan a su aplicación. Hablemos de algunos de las herramientas que puede utilizar para probar varias entradas en su sistema.

Verificar por distintos modos. CrossCheck es un ejemplo de una herramienta para probar servicios web. Usted proporciona el WSDL (lenguaje de descripción de servicios web); CrossCheck compila la página y luego le presenta un menú con pestañas que contiene cuadros de texto para que los complete. Tiene un modo Ejecutar donde puede agregar sus pruebas a una suite y luego ejecutar la suite. Ni Lisa ni Janet han probado esta herramienta, pero se señaló en el grupo de pruebas ágiles de Yahoo como una herramienta para probar servicios web si se ejecutan los mismos datos cada vez.

Ver el "Sistema Test" en el Capítulo 12, "Resumen de los cuadrantes de prueba", para ver cómo funciona Janet.
El equipo utilizó Ruby Test::Unit para probar los servicios web.

Prueba de Ruby::Unidad. Un proyecto en el que Janet estaba utilizando el marco de pruebas unitarias de Ruby, Test::Unit, para probar servicios web, con gran éxito. De hecho, el equipo pudo realizar pruebas tempranas para brindarles a los programadores retroalimentación inmediata, lo que ayudó con el diseño final.

jabón UI. Otra herramienta sugerida para probar servicios web es SoapUI. Tiene una curva de aprendizaje pronunciada, pero se puede utilizar para pruebas de rendimiento y carga. Debido a que puede recorrer filas en una hoja de cálculo de Excel o un archivo de texto, se puede utilizar para pruebas basadas en datos.

Las pruebas que funcionan en las capas debajo de la capa de presentación son muy adecuadas para escribir y automatizar pruebas de clientes que guían la codificación. Algunos practicantes no han obtenido el valor que esperaban del desarrollo basado en pruebas de historias. Brian Marick [2008] planteó la hipótesis de que una aplicación creada con desarrollo basado en pruebas de programadores, un diseño empresarial con muchos ejemplos que se basa en gran medida en discusiones de pizarra, un pequeño conjunto de pruebas de cordura automatizadas, y muchas pruebas exploratorias podrían ser menos costosas e igualmente efectivas. acercarse. Cualquiera que sea el enfoque que adopte, si está probando una aplicación con una interfaz de usuario, necesitará algo de automatización a nivel de GUI.

Herramientas para realizar pruebas a través de la GUI

Espera un minuto. ¿Cómo podemos utilizar las pruebas de GUI para impulsar el desarrollo, porque el ¿La GUI no estará lista hasta que se complete la historia? Suena contradictorio, pero las pruebas de GUI automatizadas son importantes para ayudarnos mientras desarrollamos nueva funcionalidad. Los marcos de prueba se pueden utilizar para especificar casos de prueba para una GUI herramienta antes de escribir el código. Además, puede automatizar las pruebas de GUI antes

Una justificación de la selección de herramientas

David Reed, un ingeniero de automatización de pruebas, y su equipo optaron por SoapUI Pro para automatizar las pruebas de sus servicios web. A continuación se presentan algunas de las razones que dio para elegir esta herramienta en particular.

- Tiene una versión de código abierto, por lo que puedes probarla gratis. Puedes aprenderlo, patear los neumáticos, ampliar cosas y conocer sus fortalezas y debilidades.
- Fue fácil determinar qué solicitudes realizar para qué servicio.
- Las afirmaciones proporcionadas para verificar los resultados de las solicitudes son excelentes y ampliables. Una realmente útil es verificar que la respuesta llegue en un período de tiempo aceptable, generando un error si no es así.
- La versión Pro elimina muchas molestias a la hora de diseñar consultas XPath para verificar los resultados. También agrega algunos toques agradables para recuperar datos de bases de datos.
- Es ampliable con Groovy, un lenguaje de programación basado en Java. (Ellos son trabajando en una aplicación Java, por lo que vale la pena tener herramientas compatibles con Java).
- Los desarrolladores pueden usarlo sin burlarse de él como una "herramienta de prueba".
- Se integra fácilmente con nuestro entorno de integración continua.
- Tiene una función para verificar la cobertura del código.
- El precio está bien.

antes de finalizar la codificación, ya sea mediante el uso de maquetas HTML o desarrollando un corte básico de extremo a extremo a través de todas las pantallas que simplemente navega pero que aún no proporciona todas las funciones. Incluso si no estás usando muchas pruebas de historias automatizadas para impulsar el desarrollo, exploración manual pruebas que nos ayudan a conocer la funcionalidad y proporcionan información inmediata La retroalimentación se vuelve bastante tediosa y lenta sin la ayuda de la automatización. Veamos los tipos de herramientas de prueba de GUI que ayudan a impulsar el desarrollo mediante pruebas orientadas al negocio.

Herramientas de grabación/reproducción

Las herramientas de grabación/reproducción son atractivas porque normalmente puedes aprender a grabar un guión y reproducirlo rápidamente, y puede crear muchos guiones en un poco tiempo. Sin embargo, tienen inconvenientes. Se registraron las primeras herramientas de prueba de GUI movimientos del mouse usando las coordenadas de pantalla XY. Scripts que utilizan esas herramientas También puede ser sensible a los cambios en la resolución de la pantalla, la profundidad del color e incluso donde se coloca la ventana en la pantalla.

La mayoría de las herramientas de prueba de GUI modernas utilizan objetos para reconocer los controles en una aplicación gráfica, como botones, menús y widgets de entrada de texto, para que puedan hacer referencia a ellos. a ellos simbólicamente en lugar de con coordenadas de pantalla sin formato. Esto hace la aplicación es mucho más comprobable, porque es más robusta y resiste cambios. Un botón puede moverse a una parte diferente de la pantalla, pero la prueba Todavía puedo encontrarlo según el nombre de su objeto.

Incluso con un reconocimiento de objetos mejorado, los guiones creados con grabación/reproducción suelen ser frágiles y costosos de mantener. La grabación puede ser una buena manera de Comience a crear un guión. Probadores o programadores que conocen las secuencias de comandos de la herramienta. El lenguaje puede refactorizar el guión grabado en un modelo orientado a objetos que es más fácil de usar y mantener. Históricamente, las herramientas de grabación/reproducción utilizaban lenguajes de programación patentados, que los programadores no están interesados en aprender. También es más difícil cambiar los patrones de diseño utilizados en las pruebas.

Algunas herramientas basadas en scripts, como las que hablaremos en las siguientes secciones, ofrecen una función de registro para ayudar a las personas a comenzar rápidamente a escribir la prueba. guion. Sin embargo, con esas herramientas, los guiones grabados no están destinados a reproducción directa; son solo un punto de partida para crear un sistema bien diseñado y conjunto de pruebas de fácil mantenimiento.

Muchos equipos ágiles prefieren herramientas y lenguajes de programación que les permitan crear sus lenguaje propio de dominio específico (DSL). Esto hace que las pruebas sean mucho más fáciles de entender e incluso escribir para los expertos en negocios. Veamos algunos de estos a continuación.

Herramientas de prueba ágiles de código abierto

Cada una de las herramientas de esta sección fue escrita originalmente por un equipo de desarrollo ágil que necesitaba una herramienta de prueba de GUI y no pudo encontrar ninguna de terceros. herramientas que funcionaron para su situación. Con estas herramientas, puedes escribir scripts que Utilice aplicaciones web como un usuario humano. Completan campos de texto, seleccionan de listas y haga clic en casillas de verificación y botones. Proporcionan una variedad de formas para verificar la navegación correcta y el contenido de las páginas, como la verificación específica de la herramienta pasos o XPath. Algunas de estas herramientas tienen una curva de aprendizaje más alta que las simples. herramientas de grabación/reproducción, pero la inversión adicional de tiempo generalmente vale la pena en scripts con un bajo coste total de propiedad.

Rubí con Watir. Watir (Pruebas de aplicaciones web en Ruby) es una sencilla herramienta abierta. fuente de la biblioteca Ruby para automatizar navegadores web que funciona con Internet Explorador en Windows. Hay diferentes versiones para otros navegadores, incluidos FireWatir para Firefox y SafariWatir para Safari.

La historia de Janet

Trabajé en un proyecto que desarrolló un marco de prueba de tres capas usando Ruby y Watir. La primera capa era un conjunto común de bibliotecas y la segunda capa era para acceder a las páginas y proporcionar navegación. La tercera y superior capa creó un lenguaje de dominio utilizando métodos de tipo accesorio que se correspondían con las necesidades comerciales. Esto permitió a los evaluadores manuales escribir pruebas automatizadas de alto nivel para flujos de trabajo antes de completar la codificación. Si un dispositivo no existía debido a una nueva funcionalidad, se podía crear la prueba y la palabra de acción para el dispositivo faltante se podía "ficticia" pulg. Tan pronto como se codificó el dispositivo, la prueba se pudo ejecutar como prueba de aceptación.

Un ejemplo muy sencillo del uso de Ruby con Watir incorpora la idea de DSL.

Se crearon métodos para simplificar las pruebas de modo que cualquiera de los evaluadores pudiera crear un script automatizado sin conocer Ruby o Watir.

El siguiente ejemplo muestra una prueba y luego dos de los métodos utilizados en la prueba.

```
def prueba_create_nuevo_usuario
    iniciar sesión 'administrador','admin'
    navegar_to_tab 'Administrar usuarios'
    click_button "Crear nuevo usuario"
    set_text_field "userFirstNameInput", "Rubi"
    set_text_field "usuarioApellidoInput", "RubyTester"
    click_button "Guardar cambios"
    verificar_texto "Cambios guardados"
fin

# métodos creados para facilitar la redacción de pruebas
def navegar_a_tab(nombre del elemento del menú)
    @browser.link(:texto,menuItemName).hacer clic
fin

def set_text_field(id, valor)
    @browser.text_field(:id,id).establecer valor
fin
```

Se podría agregar fácilmente un tercer nivel si se llamara a `create_new_user` más de una vez. Simplemente extraiga el código común que podría llamar la prueba:

```
create_new_user (Rubi, RubyTester)
```

Estas pruebas fueron muy adecuadas para guiar el desarrollo y proporcionar retroalimentación rápida. Hacer que las pruebas sean fáciles de escribir para los evaluadores y los clientes, manteniendo al mismo tiempo el marco de automatización diseñado para una mantenibilidad óptima, redujo el costo total de propiedad de las pruebas.

—Janet

Siempre hay inconvenientes en cualquier herramienta que utilice. Por ejemplo, existen limitaciones en el uso de objetos. A veces los programadores usan controles personalizados o un nuevo conjunto de herramientas que su herramienta tal vez no entienda.

La historia de Janet

Comencé un nuevo trabajo como gerente de control de calidad y, después de mucha deliberación, decidimos abandonar la herramienta de proveedor que el equipo había estado usando durante un par de años. No pudimos determinar qué pruebas se estaban realizando realmente ni cuál era la cobertura real. Decidimos empezar a automatizar las pruebas utilizando Ruby y Watir. La automatización fue bastante rápida al principio, pero luego las pruebas empezaron a fallar. Pasamos mucho tiempo cambiando las pruebas para reflejar nuevos nombres de objetos. Los desarrolladores simplemente estaban usando los nombres de objetos predeterminados de WebLogic, que cambiarían cada vez que se agregaba un nuevo objeto a la página. Los evaluadores acudieron a los desarrolladores para preguntarles si podían cambiar la forma en que codificaban. Fue necesario un poco de convencimiento, pero cuando los desarrolladores se dieron cuenta de los problemas que estaba causando su práctica, cambiaron sus hábitos. Con el tiempo, se cambiaron todos los valores predeterminados y a cada objeto se le asignó un nombre. Las pruebas se volvieron mucho más sólidas y pasamos mucho menos tiempo en modo de mantenimiento.

—Janet

La implementación de una nueva herramienta de automatización de pruebas generalmente requiere algo de experimentación para obtener un buen equilibrio entre código comprobable y scripts de prueba bien diseñados. Involucrar a todo el equipo hace que esto sea mucho más fácil. Watir es un ejemplo de La herramienta de prueba GUI que hemos descubierto es muy adecuada para proyectos ágiles. Veamos un Un par más, Selenium y Canoo WebTest.

Selenium. Selenium es otra herramienta de código abierto, en realidad un conjunto de herramientas, para probar aplicaciones web. Las pruebas pueden escribirse como tablas HTML o codificarse en un varios lenguajes de programación populares y se puede ejecutar directamente en la mayoría Navegadores web modernos. Un complemento de Firefox llamado "Selenium IDE" proporciona una manera para aprender la herramienta rápidamente. Se proporciona una grabadora para ayudar a crear las pruebas, incluida la escritura de afirmaciones. Las pruebas se pueden escribir en varios lenguajes de programación y secuencias de comandos comunes diferentes, incluidos Java, C# y Ruby.

Consulte el Capítulo 14, "Una automatización de pruebas ágil Strategy", para ver un ejemplo del uso de Selenium RC para crear un marco de automatización de pruebas de dominio específico.

Prueba web de Canoo. En los scripts de WebTest, las pruebas se especifican como "pasos" en XML archivos, simulando las acciones de un usuario a través de una interfaz de usuario web. He aquí un ejemplo de cómo un script WebTest podría invocar una página y verificar los resultados:

```
<setInputField descripción="establecer consulta" nombre="q" valor="Agile Tester"/>
<clickButton descripción="enviar consulta" etiqueta="Búsqueda de Google"/>
<verifyText descripción="verificar resultado" texto="Lisa Crispin" />
<verifyText descripción="verificar resultado" text="Janet Gregory" />
```

En lugar de ejecutar un navegador real, como lo hacen Selenium y Watir, WebTest simula el navegador deseado usando HtmlUnit. La ventaja de especificar pruebas en lugar de codificar scripts de prueba, es porque no hay lógica en ellos, usted No es necesario realizar la prueba.

La historia de Lisa

Mi equipo eligió WebTest para automatizar las pruebas de humo para nuestra aplicación heredada por varias razones. Debido a que los scripts están escritos en XML, los programadores del equipo se sintieron cómodos usando la herramienta. Utiliza Ant para ejecutar las pruebas, por lo que integrarlo en el proceso de compilación continuo fue sencillo. Es fácil de aprender y las pruebas se pueden diseñar de forma modular, por lo que son bastante fáciles de mantener. WebTest admite la prueba de archivos PDF, correos electrónicos y archivos de Excel, todos los cuales se utilizan ampliamente en nuestra aplicación.

Como estaba acostumbrado a poderosas herramientas de prueba comerciales, era escéptico sobre el concepto de especificar pruebas, en lugar de programarlas. Me sorprendió lo efectivas que eran las pruebas simples para detectar errores de regresión. Es posible poner lógica en las pruebas usando Groovy otros lenguajes de programación, pero solo hemos encontrado la necesidad en unos pocos casos.

Al escribir algunas pruebas por iteración, automaticé las pruebas de humo para todas las áreas críticas de nuestra aplicación en ocho meses. Estas pruebas sencillas encuentran errores de regresión con regularidad. Refactorizamos las pruebas con frecuencia, por lo que son relativamente fáciles de mantener. Nuestro retorno de la inversión en estas pruebas ha sido tremendo.

—Lisa

Selenium, WebTest y Watir son sólo tres ejemplos de los muchos programas abiertos herramientas fuente disponibles para pruebas de GUI en el momento en que escribimos este libro. Muchos equipos escriben sus propios marcos de automatización de pruebas. Veamos un ejemplo en la siguiente sección.

Herramientas de automatización de pruebas “caseras”

Bret Pettichord [2004] acuñó el término “hecho en casa” para las herramientas ágiles. Los equipos crean para satisfacer sus propias necesidades de prueba únicas. Esto permite aún más personalización que una herramienta de código abierto. El objetivo de estas herramientas suele ser proporcionar una manera para que los miembros no técnicos del equipo del cliente y los evaluadores escriban pruebas que son realmente ejecutables por la herramienta automatizada. Herramientas caseras se adaptan a las necesidades exactas del proyecto. Se pueden diseñar para minimizar el coste total de propiedad. A menudo se construyen sobre estructuras abiertas existentes. herramientas fuente.

Janet ha estado involucrada en algunos proyectos que han utilizado Ruby y Watir para crear un marco completo para pruebas funcionales. Estos marcos permitieron a los clientes especificar pruebas que luego se convirtieron en un conjunto de regresión funcional.

Ninguna herramienta de prueba garantiza el éxito. De hecho, la historia de la automatización de pruebas está plagada de intentos fallidos. Hacer que todo el equipo piense en las mejores herramientas. Usar es de gran ayuda, pero no importa qué herramienta utilice, necesita un enfoque inteligente para redactar exámenes. Discutiremos eso en la siguiente sección.

Pruebas funcionales PAS

La siguiente historia trata sobre un proyecto en el que trabajó Janet que tuvo éxito con la automatización de pruebas casera.

PAS es una aplicación de contabilidad de producción para la industria del petróleo y el gas. Utilizando lecturas brutas de medidores y acuerdos contractuales, debe calcular la propiedad de los distintos productos hasta un nivel muy preciso (es decir, los componentes del gas). Hay literalmente miles de interacciones entre las combinaciones disponibles al configurar el sistema y las salidas reales visibles para un usuario.

Dada la gran cantidad de interacciones, PAS ha empleado muchas estrategias complementarias para las pruebas.

Joseph King, uno de los programadores iniciales y entrenador ágil del equipo, nos cuenta la historia de cómo realizaron sus pruebas funcionales.

En el nivel más bajo, existen pruebas funcionales para desarrolladores que ejercitan funciones específicas a través de una API y verifican los resultados utilizando otra API de usuario de solo lectura. Actualmente hay más de 24.000 pruebas implementadas en JUnit que cada desarrollador debe ejecutar antes de poder "registrar" sus cambios en el código fuente.

El siguiente nivel es un conjunto de pruebas de GUI que prueban la clasificación de los datos del usuario hacia la API, particularmente en torno a la creación y actualización de "datos maestros". Actualmente hay más de 500 de estas pruebas implementadas usando Watij (una biblioteca de código abierto similar a Watir pero que usa Java) y JUnit que se ejecutan varias veces al día.

El último nivel de prueba es un conjunto de pruebas de integración creadas por los usuarios que corren con un arnés tipo Fit. Los usuarios identifican casos de prueba densos que reflejan casos del mundo real que cubren muchas de las funciones que trabajan juntas para producir resultados financieros y regulatorios. Estos casos de prueba luego se transcriben en plantillas de importación y luego se procesan utilizando un lenguaje de dominio que refleja la forma en que los clientes finales piensan sobre sus procesos.

Por ejemplo, después de que un cliente final ha creado la configuración de instalaciones y contratos que desea ejercer en su prueba, trabaja con un desarrollador para utilizar el lenguaje de dominio para procesar sus instalaciones en el orden correcto. Los usuarios finales también proporcionan un conjunto de resultados esperados que luego se verifican mediante una API de solo lectura. Estos resultados pueden contener miles de números, cualquiera de los cuales puede cambiar por razones aparentemente menores en un producto en evolución. Es un desafío constante distinguir entre lo que es un cambio comercial legítimo y lo que es un defecto.

Actualmente hay más de 400 pruebas de integración que se ejecutan dos veces al día y brindan retroalimentación a los clientes finales y desarrolladores.

Las pruebas exploratorias se realizan continuamente durante todo el ciclo de desarrollo y se aumentan al final de los lanzamientos.

Nuestro primer intento con PASFIT (que es lo que llamamos el marco de prueba funcional) fue una hoja de cálculo de entradas y salidas codificadas por colores. Nosotros

Luego generé código Java basado en el color de las celdas para crear los datos en PAS. Esto resultó difícil de mantener, en parte porque la aplicación estaba en constante cambio tanto a nivel de GUI como de base de datos.

Nuestra siguiente versión de PASFIT no evolucionó durante casi un año después del intento anterior. Después de tener un conjunto más estable de vistas de base de datos y GUI, pudimos crear un motor que usaba un lenguaje imperativo simple (es decir, un script) para realizar acciones con argumentos contra una GUI (por ejemplo, Ir a la página de equilibrio, Equilibrar Batería: Aceite, Agua). El guión evolucionó siguiendo el proceso de pensamiento de un contador de producción y se convirtió en un lenguaje de dominio específico. El motor fue escrito usando Ruby y Watir, y una instrucción del script era básicamente un método Ruby que se invocaba dinámicamente para que fuera fácil de actualizar. Después de ejecutar el script, el marco cargó una instantánea de las vistas que la prueba deseaba comparar e hizo una comparación simple fila por fila, celda por celda de lo que se iba a afirmar y lo que realmente sucedió. . Con el tiempo, esto se mejoró en la hoja de cálculo para utilizar tablas dinámicas para permitir a los usuarios centrarse únicamente en los resultados que deseaban afirmar para su prueba. En general, ha sido bastante exitoso, aunque los requisitos de nuestra aplicación significan que 300 pruebas tardan unas 12 horas en ejecutarse, lo cual es mucho tiempo.

Conseguir que la empresa se involucre más en el mantenimiento de las pruebas de regresión también ha sido difícil, pero cuando sucede es muy bueno. Actualmente, tenemos un stand-up donde los usuarios comerciales y los desarrolladores se reúnen durante 15 minutos para retomar cualquiera de las pruebas de escenarios que se están presentando ese día. Es bastante efectivo porque las personas a menudo saben, cuando llegan al stand-up, lo que podrían haber roto el día anterior. Es probable que las mejoras futuras incluyan la afirmación contra informes de usuarios reales en lugar de las vistas y la ejecución de una migración cada noche contra el script del escenario.

PASFIT logró un equilibrio entre permitir que los expertos empresariales escribieran pruebas en un DSL y automatizar esas pruebas con una aplicación altamente compleja. El éxito llegó con algo de prueba y error. Los equipos que escriben sus propios marcos de prueba necesitan tiempo para experimentar y encontrar la solución adecuada tanto para la empresa como para el equipo de desarrollo.

ESTRATEGIAS PARA PRUEBAS DE ESCRITURA

Las mejores herramientas del mundo no ayudarán si no las usas sabiamente. Herramientas de prueba puede hacer que sea muy fácil especificar pruebas, pero si estás especificando el Las pruebas adecuadas en el momento adecuado dependen de usted. El equipo de Lisa descubrió que demasiados detalles iniciales nublaban el panorama general hasta tal punto que los programadores No sabía qué codificar. Esto no será cierto para todos los equipos, y en algunos punto, necesitamos detalles. El último momento para proporcionarlos es cuando un programador toma una tarjeta de tarea de codificación y comienza a trabajar en una historia.

Escribir casos de prueba detallados que comuniquen el comportamiento deseado de manera efectiva requiere tanto arte como ciencia. Escenarios mal expresados y mal diseñados Los casos de prueba pueden crear más confusión de la que resuelven. Experimenta para que Puede encontrar el nivel de detalle adecuado y el diseño de prueba adecuado para cada historia. Veamos algunas estrategias que le ayudarán a utilizar herramientas con éxito para escribir útiles pruebas de cara al negocio.

Cree pruebas de forma incremental

Después de haber definido nuestras pruebas de aceptación de alto nivel para que el programador sabe qué empezar a codificar, podemos empezar a desarrollar el resto de la historia pruebas. Podemos trabajar estrechamente con el programador para asegurarnos de automatizar el mejor manera posible.

Cuando un programador comienza a trabajar en las tareas de programación de una historia, Comience a escribir pruebas detalladas. Para aquellos de nosotros que disfrutamos de las pruebas, es tentador Busque los "lores" más fuertes de inmediato, las áreas donde creemos que el código podría ser frágil. Resistir la tentación. Asegúrese de que el caso de uso más obvio esté funcionando primero. Escriba una prueba automatizada de ruta simple y feliz para mostrar que el código realiza la tarea más básica que debería. Después de que pase la prueba, puedes comenzar volviéndose más creativo. Escribir las pruebas orientadas al negocio es un proceso iterativo.

La historia de Lisa

Empiezo a escribir pruebas ejecutables orientadas al negocio que apoyan al equipo escribiendo una prueba de FitNesse simple basada en ejemplos que proporciona el propietario del producto. Le muestro esto al programador que trabaja en el código. Puede hacer sugerencias de cambios en ese momento o puede modificar él mismo la prueba según corresponda cuando lo necesite. listo para automatizarlo. Discutir la prueba a menudo lleva al programador a darse cuenta de que omitió o entendió mal un requisito. Es posible que necesitemos otra conversación a tres bandas con el cliente. El programador actualiza el código en consecuencia. También podemos mostrar la prueba al propietario del producto para asegurarnos de que capturamos el comportamiento correctamente.

Capítulo 18, "Codificación y pruebas", entra en más detalles sobre cómo los evaluadores y programadores trabajan juntos para probar y codificar.

Una vez superada la prueba simple, escribo más pruebas que cubren más reglas comerciales. Escribo algunas pruebas más complejas, las ejecuto y el programador actualiza el código o las pruebas según sea necesario. La historia se está completando para ofrecer todo el valor deseado.

—Lisa

Limite cada prueba a una regla o condición comercial. En algún momento puedes automatice o realice manualmente escenarios más complejos, pero comience cubriendo cada condición con una prueba simple. Si has seguido nuestras recomendaciones corte fino o patrón de hilo de acero, el primer conjunto de pruebas debería demostrar la primera

rebanada fina de extremo a extremo. A medida que pasen las pruebas automatizadas, agréguelas al conjunto de regresión que se ejecuta en un proceso de compilación frecuente.

Mantenga las pruebas superadas

Una vez superada una prueba, no debería fallar a menos que se hayan cambiado los requisitos. Si eso sucede, la prueba debe actualizarse antes de modificar el código. Por supuesto, Si se olvida una prueba como parte de un cambio de requisitos, esperamos que falle. Él Hizo su trabajo como detector de cambios. En este momento, es probable que sea necesario cambiar la prueba, para que pase.

Siempre que una prueba falla en un proceso continuo de integración y construcción, el La máxima prioridad del equipo (aparte de un problema crítico de producción) debe ser para que la compilación pase nuevamente. No comente la prueba fallida y corríjala más tarde; ese es el camino a la perdición. Pronto tendrás docenas de comentarios pruebas y mucha deuda técnica. Todos en el equipo deberían detener lo que lo que están haciendo y asegurarse de que la construcción vuelva a ser "verde". Determinar si un error se ha introducido, o si la prueba simplemente necesita actualizarse para adaptarse al comportamiento modificado intencionalmente. Solucione el problema, verifíquelo y haga

Asegúrese de que todas las pruebas pasen.

La historia de Lisa

Al principio de nuestros esfuerzos ágiles, mi equipo no arreglaba las pruebas fallidas con la suficiente rapidez. Escribí "¡Las pruebas no son temporales!" en la pizarra para recordarles a todos que una vez que se pasa una prueba, es necesario seguir aprobándola. Unos días después, las palabras "¡pero los probadores sí!" había sido agregado para vengarse de mí. Después de eso, mejoramos mucho en mantener nuestras construcciones "verdes".

—Lisa

Una prueba superada lleva a otra. Mantenga sus pruebas actualizadas y mantenibles con refactorización. Ampliarlos para cubrir otros casos de prueba. Las diversas combinaciones y escenarios podrían o no convertirse en parte del conjunto de regresión. después de que pasen. Queremos que nuestra suite de regresión se ejecute de manera oportuna y tener demasiadas pruebas para casos extremos lo ralentizaría.

Utilice patrones de diseño de prueba adecuados

Al diseñar pruebas, observe diferentes patrones y elija aquellos que trabajo para ti. Manténlos lo más simples que puedas. Antes de poder diseñar pruebas, tienes que identificar los que necesitas. Pierre Veragen acuñó el término prueba Patrones de génesis para anotar los patrones que le ayudan a pensar en las pruebas. Ejemplos y los casos de uso alimentan nuestros patrones de génesis de pruebas.

Construir/Operar/Verificar

El equipo de Lisa a menudo utiliza un patrón de construcción/operación/verificación: construir la entrada datos, en la memoria o realmente en la base de datos, dependiendo del propósito del prueba; invocar el código de producción para operar con esas entradas; y comprobar los resultados de esa operación. Algunos equipos llaman a esto configurar/ejecutar/validar. Por ejemplo, para probar la factura presentada a un nuevo titular de cuenta, configure las tarifas que se cargado, ingrese las propiedades de la cuenta que se relacionan con los montos de las tarifas, ejecute el código que calcula las tarifas y luego verifique qué tarifas fueron realmente cargado. Consulte la Figura 9-9 para ver un ejemplo de una prueba que configura un préstamo con un monto, tasa de interés, plazo, frecuencia de pago y fecha de inicio del servicio especificados. y luego verifica el calendario de amortización resultante. Los datos de prueba están integrados. memoria, lo que hace que la prueba sea rápida. Un dispositivo de "desmontaje" (no mostrado) elimina los datos de la prueba de la memoria para que no interfieran con pruebas posteriores.

Si es necesario probar la capa de acceso a datos de la aplicación, las pruebas se pueden ejecutar usando una base de datos real. Cada prueba puede insertar los datos de prueba que necesita, operar sobre ellos, comprobar los resultados y eliminar los datos. Probar con datos en una base de datos real puede ser una medios para automatizar una prueba contra código heredado cuyo acceso a datos y capas de lógica de negocios no se separan fácilmente.

BUILD THE DATA

Loan Fixture					
loanAmount	interestRate	term	frequency	serviceStartDate	amortize!
1000	6	1	Monthly	10-01-2005	true

CHECK THE RESULTS

Check Loan Amortization Fixture					
paymentNumber	paymentAmount	principalAmount	interestAmount	endingBalance	
1	86.07	81.07	5.00	918.93	
2	86.07	81.48	4.59	837.45	
3	86.07	81.88	4.19	755.57	
4	86.07	82.29	3.78	673.28	
5	86.07	82.70	3.37	590.58	
6	86.07	83.12	2.95	507.46	
7	86.07	83.53	2.54	423.93	
8	86.07	83.95	2.12	339.98	
9	86.07	84.37	1.70	255.61	
10	86.07	84.79	1.28	170.82	
11	86.07	85.22	0.85	85.60	
12	86.03	85.60	0.43	0.00	

Figura 9-9 Prueba de ejemplo con patrón de construcción/operación/verificación

Observe que la tabla "verificar" en el ejemplo usa un estilo declarativo, con cada fila forma un caso de prueba independiente, sin cambiar el estado del sistema. Cada fila de nuestro ejemplo prueba una línea en el calendario de amortización del préstamo. En la siguiente sección, veremos patrones que tienen un estilo procesal, con pasos que cambian o prueban el estado del sistema.

Patrones de eventos, actividades y basados en el tiempo A veces, un patrón de procedimiento basado en el tiempo refleja mejor el negocio. Por ejemplo, al probar un préstamo, queremos asegurarnos de que el interés y el principal se apliquen correctamente en cada pago. El monto del interés depende en la fecha en que se recibió el pago y la fecha del último pago procesado. Queremos una prueba que simule la obtención de un préstamo por un determinado dólar monto, tasa de interés y período de tiempo, y luego, con el tiempo, simula que el prestatario envía pagos, que se reciben y procesan. Figura 9-10 muestra un ejemplo simple de una prueba "DoFixture" de FitLibrary que solicita un préstamo, comprueba el importe del pago, contabiliza los pagos del prestatario, recibe el pago y los procesa, y luego verifica el interés, el capital y monto del saldo del préstamo. También verifica el estado de incumplimiento del préstamo.

Dependiendo del dominio, un enfoque basado en tiempo o eventos podría simular mejorar los procesos de negocio reales y ser más comprensibles para las empresas expertas que una prueba de tipo declarativa. Otros clientes pueden encontrar el estilo de tabla declarativa más sencillo de entender, porque oculta los detalles del procedimiento. Diferentes patrones funcionan mejor para diferentes situaciones, así que experimente con ellos.

1. Take out a loan
2. Check the calculated loan payment
3. Post the payment, then receive it
4. Settle and confirm the payment
5. Check the interest, principal, loan balance and default state

Loan Processing Fixture							
take loan in the amount of	1000	with interest rate	6.0	frequency	Monthly	and term	1 year with loan origination date 12-31-2005
check	periodic payment is	86.07					
post payment	1	of	86.07	on	01-30-2006		
receive payment	1	of	86.07	on	01-31-2006		
settle and confirm payment	1						
check	interest applied for	1	is	5.10			
check	principal applied for	1	is	80.97			
check	loan balance is	919.03					
as of	02-01-2006						
check	default state is	Not in Default					

Figura 9-10 Ejemplo de prueba basada en el tiempo

Aprendiendo más

Su equipo debe informarse sobre los patrones de prueba que ayudan a impulsar la programación. Encontrar el patrón correcto para cada tipo de prueba garantiza que la prueba se comunique con claridad, sea fácil de mantener y se ejecute en un período de tiempo óptimo.

Consulte la bibliografía para obtener recursos más valiosos sobre el diseño de pruebas, como Patrones de prueba xUnit de Gerard Meszaros : refactorización del código de prueba.

Reúna a programadores y evaluadores para intercambiar ideas sobre enfoques de prueba y para ayudar a decidir qué pruebas se pueden automatizar y cómo se debe diseñar el código para respaldar las pruebas. La lógica y los algoritmos empresariales deben ser accesibles. mediante dispositivos de prueba, sin tener que pasar por una interfaz de usuario o un proceso de programación por lotes. Esto permite el desarrollo basado en pruebas, que a su vez produce arquitectura comprobable.

Un enfoque común para automatizar pruebas es realizar pruebas con palabras clave o palabras de acción. Esto se puede utilizar con herramientas como Fit y FitNesse o Ruby. con Watir. Explicaremos esto a continuación.

Pruebas basadas en datos y palabras clave

Las pruebas basadas en datos son una herramienta que puede ayudar a reducir el mantenimiento de las pruebas y permitirle compartir la automatización de sus pruebas con los evaluadores manuales. Hay muchos momentos en los que desea ejecutar el mismo código de prueba una y otra vez, repitiendo solo los insumos y resultados esperados. Las hojas de cálculo o tablas, como las que admite Fit, son formas excelentes de especificar entradas. El dispositivo de prueba, el método, o script puede recorrer cada valor de datos uno a la vez, coincidiendo con lo esperado. resultados a resultados reales. Al utilizar pruebas basadas en datos, en realidad está utilizando ejemplos para mostrar lo que se supone que debe hacer la aplicación.

Las pruebas basadas en palabras clave son otra herramienta utilizada en las pruebas automatizadas, donde se utilizan palabras clave predefinidas para definir acciones. Estas acciones corresponden a un proceso relacionado con la solicitud. Es el primer paso para crear un lenguaje de prueba de dominio. Estas palabras clave (o palabras de acción) representan un lenguaje de especificación muy simple que los no programadores pueden utilizar para desarrollar aplicaciones automatizadas. pruebas. Aún necesita programadores o especialistas en automatización técnica para implementar los dispositivos sobre los que actúan las palabras de acción. Si estas palabras clave se extienden para emular el idioma del dominio, los clientes y los evaluadores no técnicos Puede especificar pruebas que se asignan al flujo de trabajo más fácilmente.

La hoja de cálculo de muestra en la Figura 9-11 muestra cómo una empresa utilizó la acción palabras para automatizar la configuración de su prueba. Las mismas palabras de acción se pueden utilizar para prueba. Las palabras Regístrate, Cerrar sesión y CCDeposit son palabras que pertenecen al dominio.

ID de secuencia de comandos	Inicio sesión	Sitio de Medio Ambiente		Justo	Corre electrónico			
8	EN	PUESTA EN ESCENA	Global	Inglés	EN			
<hr/>								
ID de prueba	Descripción Nombre de clase		Acción	Entrada 1	Entrada 2	Entrada 3	Entrada 4 Entrada 5	
# Registrar cliente								
123 cliente cdn	Miembro	Inscribirse	janet	gregorio	Calgary	Calle 123	T1T2A2	
123 Registro completo	Miembro	cerrar sesión	VERDADERO					
123 Cerrar sesión	Miembro	Cerrar sesión	VERDADERO					
<hr/>								
# Realizar depósito CC								
Configuración	Descripción Nombre de clase		Acción	Entrada 1	Entrada 2	Entrada 3	Entrada 4 Entrada 5	
234 Iniciar sesión	Miembro	Acceso	get.AcctId get.ID_greg		obtener.pwd_			
234 Enviar CC txn	Miembro	VISA CCDepósito		444433322 02		2008	25,86	
234 Miembro Cerrar sesión	Miembro	cerrar sesión						
FIN								

Figura 9-11 Ejemplo de hoja de cálculo de prueba con palabras de acción

específico. Sus usuarios podrían escribir pruebas fácilmente sin comprender el código subyacente.

Combinar técnicas de prueba basadas en datos y palabras clave puede resultar muy útil. poderoso. Fit y FitNesse utilizan palabras clave y datos para realizar pruebas. El otro Las herramientas que hemos descrito en este capítulo también pueden adaptarse a este enfoque.

Cualquier estrategia de prueba puede tener problemas si el código no está diseñado para ser fácilmente probado. Echemos un vistazo a las preocupaciones sobre la capacidad de prueba.

PRUEBAS

Pruebas empresariales creadas con patrones de diseño adecuados y escritas con antelación de cualquier codificación ayuda al equipo a lograr un diseño de código comprobable. los programadores

Comience analizando las pruebas de cara al negocio, tal vez junto con un evaluador, un analista o un cliente, de modo que la necesidad de ejecutar esas pruebas esté siempre en su mente.

mentes a medida que avanzan con su diseño basado en pruebas. Pueden construir de manera que el las pruebas proporcionan entradas y controlan las condiciones de tiempo de ejecución.

La historia de Janet

Me encontré con un problema cuando intentaba automatizar un flujo de trabajo de GUI con Ruby y Watir. No se reconoció la función emergente del calendario y el campo de datos era de solo lectura. Llevé mi problema a uno de los programadores. Nos emparejamos para que él pudiera ver el problema que estaba teniendo. Lo primero que hizo fue comprender la función del calendario. Pensó que sería demasiado difícil automatizar la prueba, por lo que sugirió otra alternativa. Creó un nuevo método que "engañosamente" el campo de entrada para que aceptara una fecha en el campo de texto. Sabíamos que el riesgo era que no hubiera automatización en el calendario, pero en aras de la simplicidad optamos por su opción.

No todo el código se puede probar mediante la automatización, pero trabaje con los programadores para encontrar Soluciones alternativas a sus problemas.

—Janet

Veamos técnicas que promueven el diseño de código comprobable.

Diseño de código y diseño de pruebas.

En el Capítulo 7, "Pruebas tecnológicas que respaldan al equipo", explicamos cómo el desarrollo basado en pruebas a nivel de unidad garantiza una arquitectura comprobable. Esto también se aplica a las pruebas empresariales. La arquitectura en capas de Lisa El equipo diseñado funciona igual de bien para pruebas funcionales. Se pueden hacer pruebas directamente contra la lógica de negocios sin involucrar la interfaz de usuario, y si apropiado, sin involucrar la capa de base de datos. Esto no significa que el No es necesario probar la capa de base de datos. Todavía hay que probarlo, sólo tal vez. en algún otro lugar.

También se debe considerar la capacidad de prueba al codificar la capa de presentación. GUI Las herramientas de prueba funcionan mejor con código bien diseñado y desarrollado con buenas prácticas.

La historia de Lisa

Cuando comencé a intentar automatizar las pruebas de GUI usando Canoo WebTest, descubrí que el HTML y JavaScript utilizados en el sistema no cumplían con los estándares y contenían muchos errores. WebTest y la herramienta en la que se basa, HtmlUnit, requerían HTML y Javascript estándar y correctos. La especificación de pruebas dependía de un buen HTML

prácticas como dar a cada elemento una identificación única. Los programadores comenzaron a escribir HTML y JavaScript (y más tarde, Ajax) con la herramienta de prueba en mente, lo que facilitó mucho la automatización de las pruebas. También comenzaron a validar su HTML y a asegurarse de que cumpliera con los estándares de la industria. Esto también redujo la posibilidad de que la aplicación tuviera problemas en diferentes navegadores y versiones de navegador.

—Lisa

La codificación y las pruebas son parte de un proceso en el desarrollo ágil. diseño de código y el diseño de pruebas son complementarios e interdependientes. Es el escenario del huevo y la gallina: no se pueden escribir pruebas sin un diseño de código comprobable, y No puedo escribir código sin pruebas bien diseñadas que comuniquen claramente los requisitos y sean compatibles con la arquitectura del sistema. Esta es la razón por la que nosotros Consideremos siempre codificar y probar juntos. Cuando estimamos historias, incluimos tiempo tanto para la codificación como para las pruebas, y cuando planificamos cada iteración y historia, asignamos tiempo para diseñar tanto las pruebas como el código. Si automatiza una prueba Si resulta difícil, evalúe el diseño del código. Si los programadores están escribiendo código que no coincide con las expectativas del cliente, el problema puede ser que las pruebas estén mal diseñadas.

Pruebas del cuadrante 2 automatizadas versus manuales



En la Parte IV,

"Automatización de pruebas", veremos

Sumérgete en el desarrollo de una estrategia exitosa de automatización de pruebas y analiza consideraciones como la creación de tus propias herramientas frente al uso de herramientas de terceros o de código abierto.

Hemos asumido que al menos una buena parte de las pruebas que guían la programación estarán automatizadas. Los escenarios de prueba manuales también pueden impulsar la programación si los comparte con los programadores con anticipación. Cuanto antes gires Si los convierte en pruebas automatizadas, más rápido obtendrá el beneficio. La mayoría de las pruebas manuales caen más en el cuadrante de "producto crítico", donde podríamos aprender cosas sobre la historia que no habíamos anticipado con el conjunto inicial de pruebas.

Eso no nos impide escribir pruebas que podrían no ser apropiadas para la automatización. No se preocupe por los detalles cuando escriba exámenes. podrías venir realizar pruebas únicas que es importante realizar pero no es importante repetir una y otra vez en una suite de regresión. Podría empezar a pensar en escenarios de un extremo a otro o trampolines para sesiones de prueba exploratorias que podrían facilitarse con cierta automatización pero que necesitan un ser humano inteligente para llevarlas a cabo. en su totalidad. Lo descubrirás más tarde. En este momento, queremos asegurarnos de capturar los requisitos críticos del cliente.

Comience con un enfoque simple, vea cómo funciona y aprovechelo. Lo importante es empezar a escribir pruebas orientadas al negocio para apoyar al equipo como desarrollas tu producto.

GESTIÓN DE PRUEBAS

Capítulo 14, "Un Estrategia ágil de automatización de pruebas", entra en más detalles sobre cómo gestionar pruebas automatizadas.

Si estamos automatizando pruebas, tiene sentido presentarlas en el cuadro de automatización. marco de herramientas, incluso si aún no son ejecutables. Queremos algún camino para todos pruebas, incluso aquellas que no serán automatizadas, para que sean accesibles para todos en el Equipo de desarrollo y comprensible para nuestros clientes. Hay un montón de Opciones disponibles que permiten a todos los miembros del equipo ver las pruebas. Los wikis son comunes. forma de compartir casos de prueba, y algunas herramientas como FitNesse utilizan una wiki o similar herramienta que permite que los requisitos narrativos, los ejemplos y las pruebas ejecutables coexistan en un solo lugar.

Las pruebas deben incluirse en el control de su código fuente, para que pueda realizar un seguimiento qué versiones de las pruebas van con qué versiones del código. En el mismo Al menos, tenga algún tipo de control de versiones para sus pruebas. Algunos equipos usan prueba. herramientas de gestión o marcos de prueba integrales que podrían integrar con gestión de requisitos, seguimiento de defectos u otros componentes.

RESUMEN

En este capítulo, analizamos las herramientas que quizás desee incluir en su kit de herramientas para ayudar crear pruebas orientadas al negocio que ayuden a impulsar el desarrollo y las pautas para asegúrese de que las herramientas ayuden en lugar de estorbar. Las herramientas y directrices incluyó lo siguiente:

Los equipos necesitan las herramientas adecuadas para obtener requisitos y ejemplos, desde el panorama general hasta los detalles, incluidas listas de verificación, mapas mentales, hojas de cálculo, maquetas, diagramas de flujo y diversas herramientas basadas en software.

Las herramientas para expresar ejemplos y automatizar pruebas, debajo y a través de la GUI, también son esenciales para una automatización ágil de pruebas. Algunas de estas herramientas incluyen herramientas de prueba unitaria, herramientas de desarrollo basadas en el comportamiento, FitNesse, Ruby con Watir, Selenium y Canoo WebTest.

La automatización de pruebas "casera" ayuda a los equipos a mantener bajo el costo total de propiedad de sus pruebas automatizadas.

Impulsar el desarrollo con pruebas orientadas al negocio es una forma en que los equipos ágiles se motivan a diseñar código comprobable.

Las estrategias de prueba para desarrollar su automatización deben incluir la creación de pruebas de forma incremental y asegurarse de que siempre se aprueben. Se pueden utilizar patrones de diseño para ayudarle a crear pruebas efectivas.

Las pruebas basadas en palabras clave y datos son un enfoque común que funciona con las herramientas que hemos analizado en este capítulo.

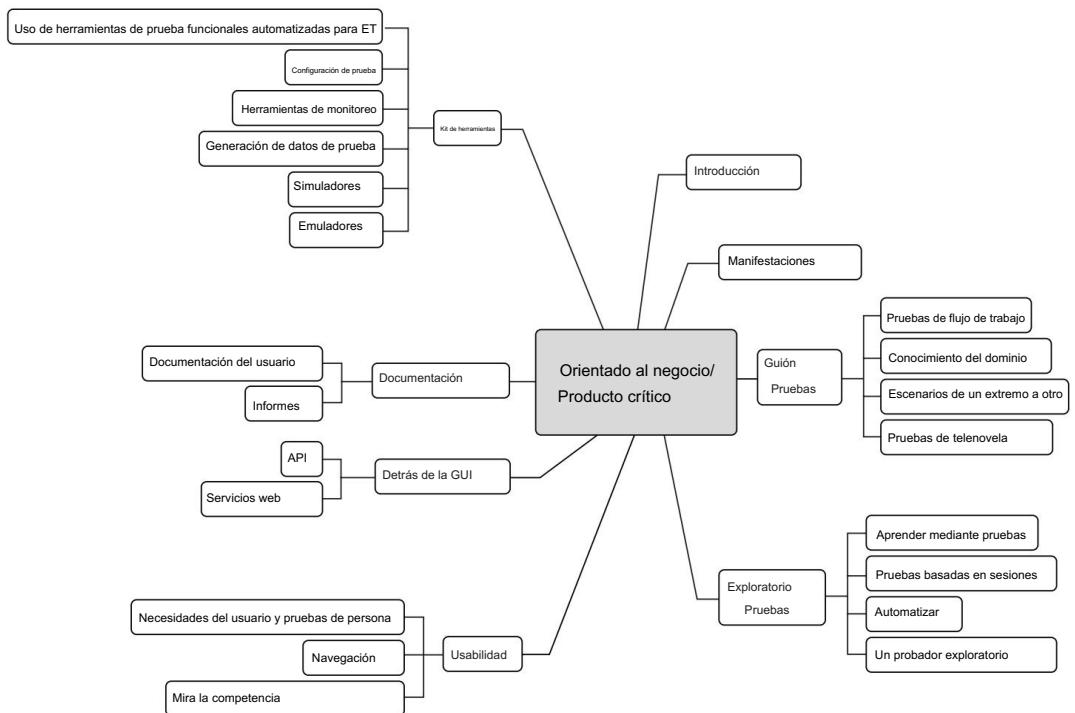
Considera la capacidad de prueba en el diseño de su código y elija sabiamente sus herramientas de prueba, porque deben funcionar con su código.

Necesitamos alguna forma de organizar las pruebas para que puedan usarse de manera efectiva y ponerse bajo control de versiones.

Esta página se dejó en blanco intencionalmente.

Capítulo 10

PRUEBAS DE CARA AL NEGOCIO QUE CRÍTICA EL PRODUCTO



Este capítulo cubre el tercer cuadrante de la matriz de pruebas. En el Capítulo 8, “Pruebas comerciales que respaldan al equipo”, hablamos sobre el segundo cuadrante y cómo utilizar pruebas comerciales para respaldar la programación. En este capítulo, le mostramos cómo criticar el producto con diferentes tipos de pruebas comerciales. También hablaremos sobre herramientas que podrían ayudar con estas actividades.

INTRODUCCIÓN AL CUADRANTE 3

Recuerde que las pruebas empresariales son aquellas que podría describir en términos eso sería (o debería) ser de interés para un experto en negocios. cuando mencionamos probar en enfoques tradicionales por fases, casi siempre significa criticar el producto después de su construcción. A estas alturas, podría pensar que en el desarrollo ágil esta parte de las pruebas debería ser fácil. Después de todo, acabamos de gastar todo eso. tiempo asegurándose de que funcione como se esperaba. Todos los requisitos han sido probados. tal como fueron construidos, incluida la seguridad y otros requisitos no funcionales, ¿bien? Todo lo que queda es posiblemente encontrar algunos errores oscuros o interesantes.

Como evaluadores, sabemos que la gente comete errores. No importa cuánto lo intentemos Si lo hacemos bien la primera vez, a veces nos equivocamos. Quizás usamos un ejemplo que no probó lo que pensábamos que hacía. O tal vez grabamos un error resultado esperado por lo que la prueba pasó, pero fue un falso positivo. Es posible que el experto en negocios haya olvidado algunas cosas que los usuarios reales necesitaban. El mejor cliente puede no saber lo que quiere (o no quiere) hasta que lo ve.

Criticar o evaluar el producto es lo que hacen los evaluadores o usuarios comerciales cuando evalúan y emiten juicios sobre el producto. Estos evaluadores forman percepciones basadas en si les gusta la forma en que se comporta, la apariencia, o el flujo de trabajo de nuevas pantallas. Es más fácil ver, sentir y tocar un producto. y responder que imaginar cómo se verá cuando se lo describan.

Es difícil automatizar pruebas comerciales que critiquen el producto, porque Estas pruebas se basan en el intelecto, la experiencia y el instinto humanos. Sin embargo, las herramientas automatizadas pueden ayudar con aspectos de las pruebas del Cuadrante 3 (ver Figura 10-1), como como configuración de datos de prueba. La última sección de este capítulo contiene ejemplos de tipos de herramientas que ayudan a los equipos a centrarse en los aspectos importantes de la evaluación el valor del producto.

Si bien muchas de las pruebas que analizamos en este capítulo son manuales, no se el error de pensar que esta prueba manual será suficiente para producir software de alta calidad y que puede salirse con la suya sin automatizar su pruebas de regresión. No tendrá tiempo para realizar ninguna prueba del Cuadrante 3 si No hemos automatizado las pruebas en los cuadrantes 1 y 2.

Evaluar o criticar el producto consiste en manipular el sistema bajo probar e intentar recrear experiencias reales de los usuarios finales. Comprensión diferentes escenarios de negocios y flujos de trabajo ayudan a que la experiencia más realista.

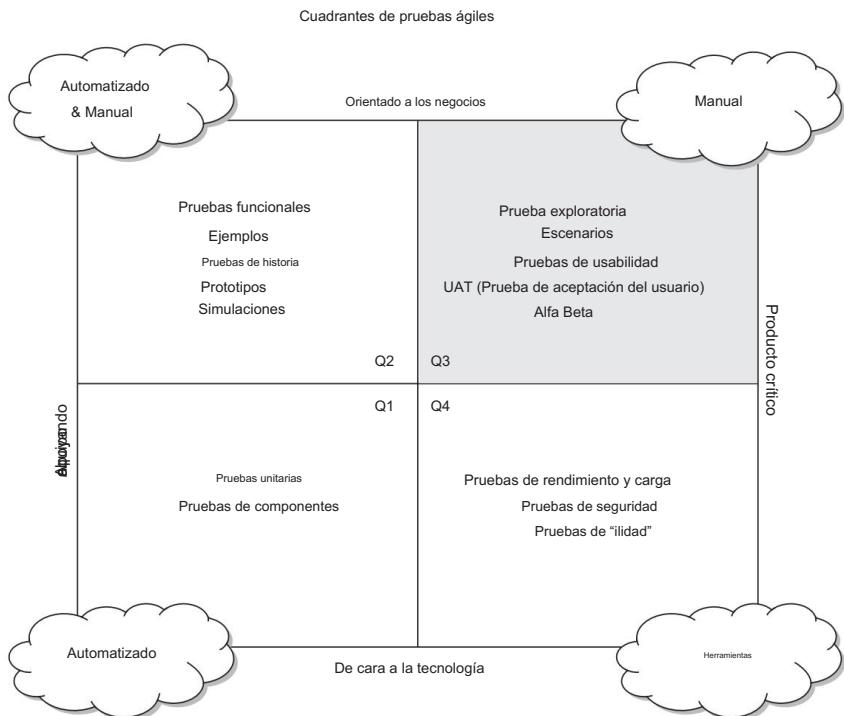


Figura 10-1 Pruebas del cuadrante 3

DEMOSTRACIONES

Recomendamos mostrar a los clientes lo que está desarrollando desde el principio y con frecuencia. Tan pronto como esté disponible una interfaz de usuario o un informe rudimentario durante el desarrollo de la historia, muéstreselo al propietario del producto u otro experto en el dominio del equipo. Sin embargo, no todos en el lado empresarial tendrán la oportunidad de ver los entregables de la iteración hasta la demostración de la iteración. Las demostraciones de final de iteración son una oportunidad para que los usuarios empresariales y los expertos en el dominio vean lo que se ha entregado en la iteración y revisen sus prioridades. Les da la oportunidad de decir: "Eso es lo que dije, pero no es lo que quise decir". Esta es una forma de criticar el producto.

La historia de Janet

Trabajé en un proyecto que tenía cinco equipos separados de ocho a diez miembros, todos desarrollando el mismo sistema. Aunque estaban en el mismo piso, la comunicación era un problema. Había muchas dependencias y superposiciones, por lo que los programadores dependían de reuniones de líderes de equipo para compartir información. Sin embargo, los usuarios empresariales y los evaluadores necesitaban ver qué estaban desarrollando otros equipos. Se basaron en demostraciones de final de iteración realizadas por cada equipo para saber qué estaban haciendo los demás equipos.

—Janet

Las demostraciones a los ejecutivos o a la alta dirección también pueden infundir confianza en su proyecto. Una de las desventajas de un proyecto por fases es no hay nada que ver hasta el final, y la gerencia tiene que colocar todo su confianza en los informes del equipo de desarrollo. La naturaleza incremental e iterativa del desarrollo ágil le brinda la oportunidad de demostrar valor comercial a medida que lo produce, incluso antes de lanzarlo. Una demostración en vivo puede ser una herramienta muy poderosa si los participantes hacen preguntas activamente sobre el nuevas características.

En lugar de esperar hasta el final de la iteración, puede aprovechar cualquier oportunidad para demostrar sus cambios. Un proyecto reciente en el que trabajó Janet utilizó reuniones programadas regularmente con los usuarios empresariales para demostrar nuevas características para obtener comentarios inmediatos. Se alimentaron todos los cambios deseados en la siguiente iteración.

Capítulo 19, "Conclusión de la iteración" habla sobre demostraciones y revisiones de final de iteración.

Elija una frecuencia para sus demostraciones que funcione para su equipo, de modo que que el ciclo de retroalimentación sea lo suficientemente rápido como para que usted pueda incorporar cambios en la liberación.

Las demostraciones informales pueden ser aún más productivas. Siéntese con un experto en negocios y muéstrelle la historia que su equipo está codificando actualmente. Hagan algunas pruebas exploratorias juntos. Hemos oido hablar de equipos que consiguen que sus partes interesadas hagan algunas pruebas exploratorias después de cada demostración de iteración para ayudarlos piense en mejoras e historias futuras para cambiar o desarrollar la funcionalidad que acaba de entregar.

PRUEBAS DE ESCENARIO

Los usuarios empresariales pueden ayudar a definir escenarios y flujos de trabajo plausibles que puedan imitar el comportamiento del usuario final. El conocimiento del dominio de la vida real es fundamental para crear escenarios precisos. Queremos probar el sistema de un extremo a otro, pero no necesariamente como una caja negra.

Una buena técnica para ayudar al equipo a comprender el negocio y el usuario. Necesidades es "pruebas de telenovela", término acuñado por Hans Buwalda [2003]. El idea aquí es tomar un escenario basado en la vida real, exagerarlo de una manera similar a la forma en que las telenovelas exageran el comportamiento y las emociones. y comprimirlo en una secuencia rápida de eventos. Piense en preguntas como, "¿Qué es lo peor que puede pasar y cómo sucedió?"

Ejemplo de prueba de telenovela

Lisa trabajó en un sitio minorista de Internet, donde descubrió que las pruebas de telenovelas eran efectivas. A continuación se muestra un ejemplo de un escenario de telenovela para probar los procesos de inventario, pedidos anticipados y pedidos pendientes del almacén de un minorista de Internet.

El juguete más popular en nuestra juguetería en línea en esta temporada navideña es la figura de acción Super Tester. Tenemos 20 pedidos anticipados esperando la recepción de los artículos en nuestro almacén. Finalmente, Jane, supervisora de almacén, recibe 100 figuras de Super Tester Action. Actualiza el sistema de inventario para mostrar que hay inventario disponible según la orden de compra y que ya no es un pedido anticipado. Nuestro sitio web ahora muestra figuras de acción Super Tester disponibles para entrega a tiempo para las fiestas. El sistema libera los pedidos anticipados, que se envían al almacén. Mientras tanto, Joe, el conductor del montacargas, se distrae con su teléfono celular y accidentalmente choca contra el estante que contiene las figuras de acción de Super Tester. Todos parecen estar destrozados hasta quedar irreconocibles. Jane, horrorizada, retira los 100 artículos del inventario disponible. Mientras tanto, se han acumulado más pedidos de este popular juguete en el elemento del sistema. Al revisar los escombros, Jane y Joe descubren que 14 de las figuras de acción han sobrevivido intactas. Jane los vuelve a agregar al inventario disponible.

Este escenario prueba varios procesos en el sistema, incluidos pedidos anticipados, recepción de órdenes de compra, pedidos pendientes, cancelaciones de almacén y liberación de pedidos anticipados. ¿Cuántos juguetes Super Tester aparecerán como disponibles en el sitio web de compras al final de todo eso? Mientras ejecutamos el escenario, probablemente encontraremos otras áreas que queramos investigar; tal vez la aplicación de órdenes de compra sea difícil de usar o las actualizaciones del inventario del almacén no se reflejen correctamente en el sitio web. Pensar y ejecutar este tipo de pruebas nos enseñará más sobre lo que nuestros usuarios y otros clientes externos necesitan que ejecutar pruebas funcionales predefinidas en áreas más limitadas de la aplicación. Además, ¡es divertido!

Capítulo 14, "Un Agile Test Automation Strategy", examina diferentes enfoques para obtener datos de prueba.

Como evaluador, a menudo "inventamos" datos de prueba, pero generalmente son simples, por lo que podemos Comprueba fácilmente nuestros resultados. Al probar diferentes escenarios, tanto los datos como el flujo debe ser realista. Descubra si los datos provienen de otro sistema o si se ingresa manualmente. Obtenga una muestra si puede pedírsela a los clientes que proporcionar datos para las pruebas. Los datos reales fluirán a través del sistema y pueden ser comprobado a lo largo del camino. En sistemas grandes, se comportará de manera diferente dependiendo sobre qué decisiones se toman.

Las herramientas para ayudar a definir los escenarios y los flujos de trabajo pueden ser sencillas. flujo de datos o Los diagramas de flujo del proceso ayudarán a identificar algunos de los escenarios comunes.

Estos escenarios pueden ayudarle a pensar en un problema complejo si toma el tiempo. Considere a los usuarios y su motivación.

La historia de Lisa

Nuestro equipo planeó reescribir la funcionalidad principal de la aplicación que procesa las compras y ventas diarias de fondos mutuos. Estas operaciones son el resultado de que los participantes del plan de jubilación realicen contribuciones, intercambien saldos de un fondo a otro o retiren dinero de sus cuentas. El compañero de trabajo de Lisa, Mike Thomas, estudió el flujo de procesamiento comercial existente y lo diagramaba para que el equipo pudiera entenderlo bien antes de intentar reescribir el código. La Figura 10-2 muestra una parte del diagrama de flujo. WT significa el custodio que realiza las operaciones reales. Se descargan y traducen tres tipos de archivos diferentes a formatos legibles: CFM, PRI y POS. Cada uno de estos archivos ingresa a una parte diferente de la aplicación para realizar el procesamiento y producir diversos resultados: operaciones liquidadas, un informe de excepción de ticker y un informe de posición de fondos.

—Lisa

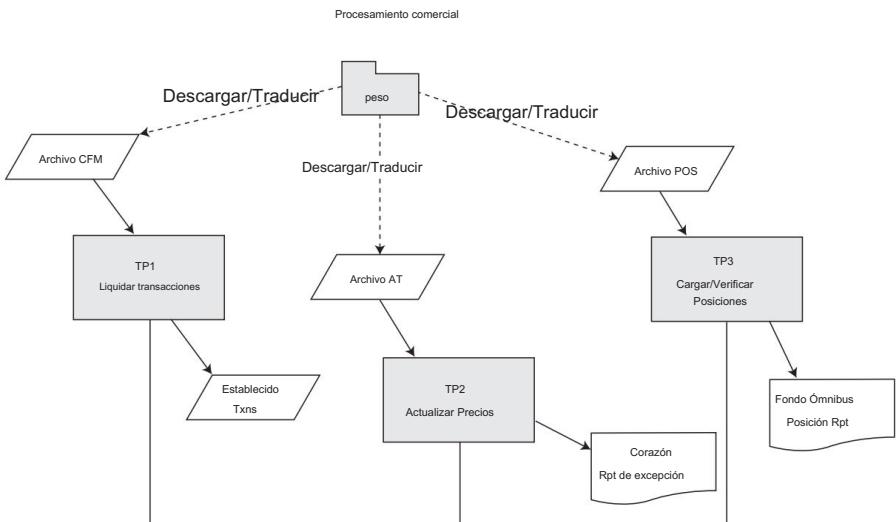


Figura 10-2 Porción de muestra de un diagrama de flujo de proceso

Al realizar pruebas de un extremo a otro, realice comprobaciones puntuales para asegurarse de que los datos, el estado las banderas, los cálculos, etc. se comportan como se esperaba. Utilice diagramas de flujo y otras ayudas visuales para ayudarle a comprender la funcionalidad. Muchas organizaciones

Las organizaciones dependen de informes para tomar decisiones, y esos informes parecen ser lo último que verificamos. Si sus escenarios han sido identificados correctamente, usted Es posible que pueda utilizar los informes de su aplicación para realizar una verificación final.

PRUEBA EXPLORATORIA

Las pruebas exploratorias (ET) son un enfoque importante para las pruebas en el entorno ágil. mundo. Como herramienta de investigación, es un complemento fundamental para las pruebas de la historia y nuestra suite de regresión automatizada. Es un enfoque sofisticado y reflexivo para pruebas sin un script y le permite ir más allá de las variaciones obvias que ya se han probado. Las pruebas exploratorias combinan aprendizaje, diseño de pruebas y ejecución de pruebas en un solo enfoque de prueba. Aplicamos heurísticas y técnicas de forma disciplinada para que el "hacer" revele más implicaciones que el simple hecho de pensar en un problema. A medida que realiza la prueba, aprende más sobre el sistema bajo prueba y puede usar esa información para ayudar a diseñar nuevas pruebas.

La bibliografía enumera más recursos que debe investigar para aprender más sobre las pruebas exploratorias.

Las pruebas exploratorias no son un medio para evaluar el software mediante pruebas exhaustivas. Está destinado a agregar otra dimensión a sus pruebas. Tú haces lo suficiente para ver si las historias "hechas" realmente están hechas para su satisfacción.

Un efecto secundario valioso de las pruebas exploratorias es el aprendizaje que surge de él. Revela áreas del producto que podrían necesitar pruebas más automatizadas y trae ideas para características nuevas o modificadas que conducen a nuevas historias.

Pruebas exploratorias

Michael Bolton es formador y consultor en enfoques de pruebas rápidas y exploratorias. Imparte un curso llamado Pruebas rápidas de software, que coescrcribe con el autor principal James Bach. Aquí está la definición de Michael de pruebas exploratorias.

Consulte la bibliografía para ver algunos enlaces a más información sobre las pruebas rápidas de software.

Cem Kaner no inventó las pruebas exploratorias, pero las identificó y las nombró en 1983, en la primera edición del enfoque que Pruebas Software de ordenador, como un todos los evaluadores utilizan cuando sus cerebros están ocupados con su trabajo. Él y James Bach, el otro destacado defensor de este enfoque, han definido durante mucho tiempo las pruebas exploratorias como "diseño, ejecución y aprendizaje simultáneos de pruebas". Kaner también define las pruebas exploratorias de manera más explícita como "un estilo de prueba que enfatiza la libertad y responsabilidad del evaluador individual para optimizar continuamente el valor de su trabajo al tratar el aprendizaje, el diseño de la prueba, la ejecución de la prueba y la interpretación de los resultados de la prueba como actividades que continúan en paralelo durante todo el proyecto". Eso es bastante bocado. ¿Qué significa?

Lo más importante que hay que recordar acerca de las pruebas exploratorias es que no es una técnica de prueba en sí misma. Más bien, es un enfoque o una mentalidad que se puede aplicar a cualquier técnica de prueba. Lo segundo que hay que recordar es que las pruebas exploratorias no se refieren simplemente a la ejecución de la prueba; Los evaluadores también pueden adoptar un enfoque exploratorio cuando diseñan nuevas pruebas al comienzo de la iteración o analizan los resultados de pruebas que ya se han realizado. Una tercera nota importante es que las pruebas exploratorias no son pruebas descuidadas, descuidadas o sin preparación. Un enfoque exploratorio puede requerir una preparación muy extensa y elaborada para ciertas pruebas, y el conocimiento y el conjunto de habilidades de un evaluador exploratorio, desarrollado a lo largo de años, es una forma de preparación a menudo invisible pero importante. Una prueba exploratoria podría realizarse manualmente o podría emplear un uso extensivo de automatización de pruebas, es decir, cualquier uso de herramientas para respaldar las pruebas. Entonces, si las pruebas exploratorias no son una técnica, ni una ejecución de pruebas, ni espontáneas, ni manuales, ¿qué es lo que hace que una actividad de prueba sea exploratoria? La respuesta está en el compromiso cognitivo del evaluador:

cómo responde el evaluador a una situación que cambia continuamente.

Supongamos que a un evaluador se le asigna la misión de probar un cuadro de diálogo de configuración para un editor de texto. Un evaluador que utilice un enfoque exploratorio utilizaría especificaciones y conversaciones sobre el comportamiento deseado para informar las ideas de prueba, pero tendería a registrar estas ideas con menos detalle que un evaluador que utilice un enfoque escrito. Un evaluador experto generalmente no necesita mucha instrucción explícita a menos que las ideas de la prueba requieran algunas acciones o datos específicos. Si es así, es posible que se anoten o se proporcionen a un programa que pueda ejercitálos rápidamente. Al ver el diálogo, el evaluador exploratorio interactuaría con él, generalmente realizando pruebas de acuerdo con las ideas originales de la prueba, pero también podría centrar su atención en otras ideas basadas en nuevos problemas o riesgos en el diálogo tal como aparecía frente a ella. . . ¿Pueden dos configuraciones entrar en conflicto de una manera que no esté cubierta por las pruebas existentes? El evaluador exploratorio investiga inmediatamente realizando una prueba sobre el terreno. ¿Tiene el diálogo un problema de usabilidad que podría interferir con el flujo de trabajo de un usuario? El evaluador exploratorio considera rápidamente una variedad de usuarios y escenarios y evalúa la importancia del problema. ¿Hay un retraso al presionar el botón OK? El evaluador exploratorio realiza algunas pruebas más para buscar un patrón general. ¿Existe la posibilidad de que algunas opciones de configuración no sean posibles en otra plataforma? El evaluador exploratorio nota la necesidad de realizar pruebas adicionales y continúa. Al recibir nuevas compilaciones, el evaluador exploratorio tendería a restar importancia a la repetición y enfatizaría la variación para descubrir problemas pasados por alto en pruebas más antigüas que ya no revelan información interesante. Este enfoque, que siempre ha sido fructífero, es aún más poderoso en entornos donde la necesidad de realizar pruebas repetidas es manejada por las pruebas de regresión automatizadas de bajo nivel de los desarrolladores.

Las pruebas exploratorias se caracterizan por el grado en que el evaluador está bajo su propio control, tomando decisiones informadas sobre lo que él o ella

va a hacer a continuación, y donde el último resultado de la última actividad informa conscientemente la siguiente elección. Los enfoques exploratorios y guionizados se encuentran en los polos opuestos de un continuo. En el extremo de la mentalidad basada en un guión, la decisión sobre qué hacer a continuación proviene exclusivamente de otra persona, en algún momento del pasado. En la mentalidad exploratoria, la decisión de continuar en la misma línea de investigación o elegir un nuevo camino proviene enteramente del evaluador individual, en el momento en que ocurre la actividad. El resultado de la última prueba informa fuertemente la decisión del evaluador.

Opciones para la próxima prueba. Otras influencias incluyen las partes interesadas para quienes la información de prueba podría ser importante, los criterios de calidad que son importantes para las partes interesadas, la cobertura de prueba que buscan las partes interesadas, los riesgos específicos asociados con el artículo que se está probando, las necesidades del usuario final del producto, las habilidades del evaluador, las habilidades de los desarrolladores, el estado del elemento bajo prueba, el cronograma del proyecto, el equipo y las herramientas que están disponibles para el evaluador y el grado en que puede usarlos de manera efectiva. Y esa es sólo una lista parcial.

Ninguna actividad de prueba realizada por un ser humano pensante está completamente escrita. Los seres humanos tenemos una capacidad extraordinaria para reconocer cosas incluso cuando la gente les dice que no lo hagan y, como resultado, podemos distraernos y desviarnos, pero podemos aprender y adaptarnos sorprendentemente rápido a nueva información e investigar sus causas y efectos. Las máquinas sólo reconocen aquello para lo que han sido programadas. Cuando se enfrentan a un resultado de prueba sorprendente, en el mejor de los casos lo ignoran; en el peor de los casos, bloquean o destruyen datos.

Sin embargo, ninguna actividad de prueba realizada en nombre de un cliente tampoco es completamente exploratoria. El evaluador exploratorio está inicialmente impulsado por la misión de prueba, que normalmente la establece el cliente al principio del proyecto. El trabajo exploratorio también puede guiarse por listas de verificación, modelos estratégicos, esquemas de cobertura, listas de riesgos: ideas que podrían surgir de otras personas en otros momentos. Cuanto más el evaluador esté controlado por estas ideas en lugar de guiado por ellas, más las pruebas adoptarán un enfoque guionizado.

Una buena exploración requiere una investigación continua del producto por parte de probadores humanos comprometidos, en colaboración con el resto de la comunidad del proyecto, en lugar de seguir un enfoque estructurado de procedimientos, realizado exclusivamente mediante automatización. La exploración enfatiza a los individuos y las interacciones sobre los procesos y herramientas. En un entorno ágil, donde el código se produce primero como prueba y se cubre con pruebas de regresión automatizadas, los evaluadores pueden tener no solo la confianza sino también la capacidad para desarrollar nuevas pruebas y buscar nuevos problemas en el momento.

La exploración enfatiza responder al cambio en lugar de seguirlo. a plan.

Los enfoques exploratorios utilizan la variación para impulsar una búsqueda activa de problemas en lugar de casos de prueba manuales o automatizados que simplemente confirman lo que ya sabíamos.

La exploración enfatiza el software funcional

sobre la documentación completa.

Y para que sea efectivo, bueno.

La exploración requiere retroalimentación frecuente entre evaluadores, desarrolladores, clientes y el resto de la comunidad del proyecto, no simplemente la repetición de pruebas que se prepararon al comienzo de la iteración, antes de que hubiéramos aprendido cosas importantes sobre el proyecto.

La exploración enfatiza

la colaboración con el cliente sobre los contratos negociados.

Exploratorio

Los enfoques son fundamentalmente ágiles.

Las pruebas exploratorias adoptan los mismos valores que el desarrollo ágil. Es una parte importante de la "mentalidad de prueba ágil" y fundamental para el éxito de cualquier equipo.

Las personas que no están familiarizadas con las pruebas exploratorias a menudo las confunden con las pruebas ad hoc. Las pruebas exploratorias no consisten en sentarse frente a un teclado y escribir.

Es posible que los evaluadores de "caja negra" no calificados no sepan cómo realizar pruebas exploratorias.

Las pruebas exploratorias comienzan con una carta de qué aspectos de la funcionalidad será explorado. Requiere pensamiento crítico, interpretación de los resultados y comparándolos con expectativas o sistemas similares. Seguir "huele" cuando

Las pruebas son un componente importante. Los evaluadores toman notas durante sus sesiones de prueba exploratorias para poder reproducir cualquier problema que vean y realicen.

más investigación según sea necesario.

Técnica: Pruebas exploratorias y evaluación de la información.

Jon Hagar, un experimentado evaluador, alumno y capacitador exploratorio, comparte algunas actividades, características y habilidades que son vitales para las pruebas exploratorias efectivas.

Las pruebas exploratorias utilizan la comprensión del sistema por parte del evaluador, junto con el pensamiento crítico, para definir "pruebas" experimentales enfocadas que se pueden ejecutar en períodos de tiempo cortos y luego retroalimentar el proceso de planificación de pruebas.

Un equipo ágil tiene muchas oportunidades para realizar pruebas exploratorias, ya que cada ciclo de desarrollo crea software funcional y listo para producción.

Al comenzar temprano en cada ciclo de desarrollo, considere realizar pruebas exploratorias basadas en:

- Riesgo (análisis): Las cosas críticas que usted y el cliente/usuario piensan que pueden salir mal o ser problemas potenciales que harán que la gente se sienta infeliz. • Modelos (mentales o de otro tipo) de cómo debería comportarse el software: usted y/o el cliente tienen una gran expectativa sobre cómo debería funcionar o verse la función recién producida, por lo que la prueban.
- Experiencia pasada: piense en cómo sistemas similares han fallado (o tenido éxito) en patrones predecibles que pueden refinarse en una prueba, y explórela.

- Lo que le dice su equipo de desarrollo: hable con sus desarrolladores y descubra qué “es importante para nosotros”.
- Lo más importante: lo que aprende (ve y observa) mientras realiza la prueba. Como evaluador en un equipo ágil, una gran parte de su trabajo es aprender constantemente sobre su producto, su equipo y su cliente. A medida que aprenda, debería ver rápidamente pruebas basadas en cosas como las necesidades del cliente, los errores comunes que el equipo parece estar cometiendo o las buenas/malas características del producto.

Algunas pruebas pueden ser buenas candidatas para conjuntos de regresión automatizados. Algunos podrían simplemente responder a su carta exploratoria y “terminar”. El equipo ágil debe pensar críticamente lo que está aprendiendo y “evolucionar” pruebas en consecuencia. El aspecto más importante aquí es estar “concentrado” durante las pruebas, donde busca lo “divertido”, lo inesperado o lo nuevo que las pruebas automatizadas pasarían por alto. Utilice la automatización para aquello en lo que es bueno (tareas repetitivas) y utilice humanos ágiles para aquello en lo que somos buenos (ver, pensar y afrontar lo inesperado).

Normalmente se necesitan varios componentes para realizar pruebas exploratorias útiles:

- Diseño de pruebas: un evaluador exploratorio como un buen diseñador de pruebas entiende destaca los numerosos métodos de prueba. Debería poder poner en juego diferentes métodos sobre la marcha durante la exploración. Esta agilidad es una gran ventaja de las pruebas exploratorias sobre los procedimientos automatizados (programados), en los que las cosas deben pensarse con antelación.
- Observación cuidadosa: los evaluadores exploratorios son buenos observadores. Están atentos a lo inusual e inesperado y son cuidadosos con las suposiciones de corrección. Es posible que observen características o patrones sutiles del software que los impulsen a cambiar la prueba en tiempo real.
- Pensamiento crítico: la capacidad de pensar abiertamente y con agilidad es una razón clave para que humanos pensantes realicen pruebas exploratorias no automatizadas. Los evaluadores exploratorios pueden revisar y redirigir una prueba en direcciones inesperadas sobre la marcha. También deberían poder explicar su lógica de búsqueda de defectos y proporcionar un estado claro de las pruebas. El pensamiento crítico es una habilidad humana que se aprende.
- Ideas diversas: los evaluadores experimentados y los expertos en la materia pueden producir más y mejores ideas. Los evaluadores exploratorios pueden aprovechar esta diversidad durante las pruebas. Una de las razones clave para las pruebas exploratorias es utilizar el pensamiento crítico para conducir las pruebas en direcciones inesperadas y encontrar errores.
- Recursos ricos: los evaluadores exploratorios deben desarrollar un gran conjunto de herramientas, técnicas, datos de prueba, amigos y fuentes de información a las que pueden recurrir. Los miembros del equipo de prueba ágil deben aumentar sus recursos exploratorios a lo largo de un proyecto y a lo largo de su carreras.

Para ayudarle a comprender un día en la vida de un evaluador exploratorio ágil, aquí tiene una breve historia del evaluador:

Llegué a las 8:00 am y revisé lo que había sucedido la noche anterior durante las pruebas automatizadas. Las pruebas automatizadas de la noche anterior encontraron algunos errores menores pero interesantes. Un campo de contraseña en un formulario de inicio de sesión había aceptado un carácter especial, que debería haber sido rechazado por la validación. Creé un esquema como punto de partida para mi "ataque" (un plan de alto nivel y/o una lista de riesgos).

Mientras pensaba en mi "plan de ataque", esbozé un pequeño modelo de estado del problema en un rotafolio y se lo mostré a un desarrollador y al representante de atención al cliente de mi equipo. Diseñé una prueba incorporando sus sugerencias, utilizando algunas entradas de estrés de datos que esperaba que la validación rechazara (un archivo de 1 MG de caracteres especiales). Ejecuté mi prueba con mi entrada de estrés y, efectivamente, el sistema los rechazó como se esperaba. Probé con un conjunto de datos diferente y el sistema falló con un desbordamiento del búfer en la base de datos.

Estaba aprendiendo y estábamos tras la pista de un error de seguridad potencialmente grave. A medida que avanzaba el día, exploré diferentes entradas al campo de contraseña y trabajé con el equipo para solucionar el error.

Puede aprender de los resultados de las pruebas automatizadas y de las pruebas exploratorias. Cada tipo de prueba se alimenta del otro. Desarrolle una amplia gama de habilidades para que pueda identificar problemas importantes y redactar pruebas para evitar que vuelvan a ocurrir.

El término prueba exploratoria fue popularizado por la "escuela impulsada por el contexto" de pruebas. Es una actividad altamente disciplinada y se puede aprender. Basado en sesiones La gestión de pruebas es un método de prueba diseñado para hacer que las pruebas exploratorias sean auditables y mensurables [Bach, 2003].

Pruebas basadas en sesiones

Las pruebas basadas en sesiones combinan responsabilidad y pruebas exploratorias. Él proporciona un marco para la experiencia de prueba exploratoria de un evaluador para que pueda informar los resultados de manera consistente.

La historia de Janet

James Bach [2003] compara las pruebas exploratorias con armar un rompecabezas. Cuando leí por primera vez su artículo con la analogía del rompecabezas, las pruebas exploratorias tenían mucho sentido para mí.

Empiezo un rompecabezas tirando todas las piezas del rompecabezas y luego ordenándolas en diferentes colores y piezas de borde. A continuación, junté las piezas de los bordes, lo que me da un marco para empezar. El borde del rompecabezas es análogo tanto a la declaración de misión, que me ayuda a concentrarme, como al control del tiempo de una sesión, que me mantiene dentro de ciertos límites.

Las pruebas basadas en sesiones son una forma de prueba exploratoria, pero tienen un límite de tiempo y son un poco más estructuradas. Aprendí sobre las pruebas basadas en sesiones de Jonathan Bach y descubrí que me dio la estructura que necesitaba para realizar bien las pruebas exploratorias. Utilizo las mismas habilidades que uso para un rompecabezas: busco patrones de color o formas o tal vez algo que simplemente no se ve bien, una anomalía. Mi proceso de pensamiento puede tomar esos patrones y darles sentido, utilizando heurísticas que he desarrollado para ayudarme a resolver un rompecabezas.

—Janet

Como resolver el rompecabezas juntando primero las piezas exteriores,
Podemos utilizar pruebas basadas en sesiones para proporcionarnos el marco en el que trabajamos. En pruebas basadas en sesiones, creamos una misión o un estatuto y luego programamos nuestro sesión para que podamos centrarnos en lo que es importante. Con demasiada frecuencia, como evaluadores, podemos ir desviarse y terminar persiguiendo un error que podría o no ser importante para lo que estamos probando actualmente.

Para obtener más información sobre las pruebas basadas en sesiones, consulte la bibliografía del trabajo de Jonathan Bach.

Las sesiones se dividen en tres tipos de tareas: diseño y ejecución de pruebas, detección de errores. investigación e informes, y configuración de sesiones. Medimos el tiempo que gastar en configuración versus ejecución de prueba real para que sepamos dónde gastamos la mayoría del tiempo. Podemos capturar resultados de manera consistente para que podamos informar al equipo.

Automatización y pruebas exploratorias

También podemos combinar pruebas exploratorias con la automatización de pruebas. Jonathan Kohl, en su artículo "Man and Machine" [2007], habla de pruebas interactivas. automatización para ayudar a las pruebas exploratorias. Utilice la automatización para realizar la configuración de prueba, generación de datos, tareas repetitivas o para avanzar en un flujo de trabajo hasta el lugar quieras empezar. Luego comienza a utilizar sus habilidades y experiencia en pruebas para encontrar los errores realmente "buenos", los insidiosos que de otro modo pasarían desapercibidos. También puede utilizar un conjunto de pruebas automatizadas para explorar. Solo modificalo un poco observe los resultados mientras se ejecuta, modifíquelo nuevamente y observe lo que sucede.

Un probador exploratorio

Con las pruebas exploratorias, cada evaluador tiene un enfoque diferente ante un problema, y tiene un estilo de trabajo único. Sin embargo, existen ciertos atributos que Convírtete en un buen evaluador exploratorio. Un buen probador:

Es sistemático, pero persigue "lores" (anomalías, piezas que no son consistentes)

Aprende a reconocer problemas mediante el uso de Oráculos (principio o mecanismo por el cual reconocemos un problema)

Elige un tema, función o declaración de misión para centrar las pruebas.

Sesiones de cajas de tiempo y viajes paralelos.

Piensa en lo que haría el usuario experto o novato

Explora junto con expertos en el dominio

Comprueba aplicaciones similares o competitivas.

Las pruebas exploratorias nos ayudan a aprender sobre el comportamiento de una aplicación.

Los evaluadores generalmente saben mucho sobre la aplicación que están probando. ¿Cómo lo hacen? juzgar si la aplicación es utilizable por usuarios menos técnicos o no

¿Está familiarizado con él? Las pruebas de usabilidad son vitales para muchos sistemas de software. Hablaremos sobre eso en la siguiente sección.

PRUEBAS DE USABILIDAD

Hay dos tipos de pruebas de usabilidad. El primer tipo es el que se hace desde el principio por parte de la gente de experiencia del usuario, utilizando herramientas como marcos de alambre para ayudar programación del variador. Ese tipo de pruebas pertenecen al Cuadrante 2. En esta sección, estamos hablando del tipo de prueba de usabilidad que critica el producto. Usamos herramientas como las personas y nuestra intuición para ayudarnos a ver el producto pensando en el usuario final.

Necesidades del usuario y pruebas de persona

Veamos un ejemplo de compras en línea. Pensamos en quién usará el sitio. ¿Serán personas que han comprado en línea antes o serán marcas?

¿Nuevos usuarios que no tienen idea de cómo proceder? Suponemos que será una mezcla de ambos, además de otros. Tómese el tiempo para pedirle a su grupo de marketing que obtener la demografía de los usuarios finales. Los números pueden ayudarte a planificar tus pruebas.

Una forma de utilizar personas es que su equipo invente varias personas diferentes. usuarios de su aplicación que representan diferentes niveles de experiencia y necesidades.

Para nuestra aplicación de venta minorista por Internet, podríamos tener las siguientes personas:

Nancy Newbie, una persona mayor que es nueva en las compras por Internet y está nerviosa por el robo de identidad.

Hudson Hacker, que busca formas de engañar a la página de pago

Enrico Executive, que hace todas sus compras online y envía regalos a todos sus clientes en todo el mundo.

Betty Bargain, que busca grandes ofertas

Debbie Ditherer, a quien le cuesta decidir qué artículos realmente quiere pedir

Podríamos colgar fotografías que representen a estas diferentes personas y sus biografías en nuestra área de trabajo para tenerlas siempre presentes. podemos probar el mismo escenario que cada persona por turno y ver qué diferentes experiencias podrían encontrarse.

Otra forma de abordar las pruebas de personalidad, que aprendimos de Brian Mar-ick y Elisabeth Hendrickson, es elegir un personaje ficticio o una celebridad famosa e imaginar cómo usarían nuestra aplicación. ¿La reina de Inglaterra podrá navegar por nuestro proceso de pago? ¿Cómo podría Homero Simpson buscar el artículo que quiere?

Proyectos del mundo real: Personas

El equipo de OneNote de Microsoft utiliza personas como parte de su proceso de prueba. Mike Tholfsen [2008], director de pruebas de OneNote, afirma que utilizan siete personas que podrían utilizar OneNote: tipos de clientes específicos, como abogados, estudiantes, agentes inmobiliarios y vendedores. Las personas que crean contienen información como:

- Descripción general del puesto
- "Un día en la vida"
- Usos principales de OneNote
- Lista de funciones que la persona podría utilizar
- Posibles estructuras de cuadernos
- Otras aplicaciones utilizadas
- Configuración y entorno de hardware

También puedes asumir los roles de usuarios novatos, intermedios y expertos. mientras exploras la aplicación. ¿Pueden los usuarios descubrir lo que se supone? ¿hacer sin instrucciones? Si tiene muchos usuarios nuevos, es posible que Necesito hacer que la interfaz sea muy simple.

La historia de Janet

Cuando comencé a probar un nuevo sistema de contabilidad de producción, me resultó muy difícil entender el flujo, pero a los contadores de producción del equipo les encantó. Después de trabajar con ella durante un tiempo, entendí la complejidad detrás de la aplicación y supe por qué no tenía por qué ser intuitiva para un usuario nuevo. Esta fue una buena lección para mí, porque siempre asumí que las aplicaciones tenían que ser fáciles de usar.

—Janet

Si su aplicación está diseñada a medida para tipos específicos de usuarios, es posible que deba Sea "inteligente" en lugar de intuitivo. Las sesiones de entrenamiento podrían ser suficientes para conseguir

sobre la falta inicial de usabilidad para que la interfaz pueda diseñarse para lograr la máxima eficiencia y utilidad.

Navegación La

navegación es otro aspecto de las pruebas de usabilidad. Es increíblemente importante pruebe los enlaces y asegúrese de que el orden de tabulación tenga sentido. Si un usuario tiene una opción de aplicaciones o sitios web, y tiene una mala primera experiencia, probablemente no lo harán usar su aplicación nuevamente. Algunas de estas pruebas son automatizables, pero es importante probar la experiencia real del usuario.

—
Consulte el Capítulo 8, "Presentación empresarial Pruebas que respaldan al equipo", para ver un ejemplo de las pruebas del Mago de Oz, que es un enfoque para diseñar para la usabilidad.

Si tiene acceso a los usuarios finales, involúcrelos en las pruebas de navegación. Empájese con un usuario real o observe cómo uno usa la aplicación y toma notas. Cuando estés diseñando una nueva interfaz de usuario, considera usar focus grupos para evaluar diferentes interfaces. Puedes comenzar con maquetas y flujos dibujados en papel, obtenga opiniones y pruebe maquetas HTML a continuación, para obtener retroalimentación temprana.

Mira la competencia

Al evaluar la usabilidad de su aplicación, piense en otras aplicaciones que sean similares. ¿Cómo realizan las tareas? ¿Los consideras fácil de usar o intuitivo? Si puede obtener acceso a software de la competencia, tome algo de tiempo para investigar cómo funcionan esas aplicaciones y compararlas con tu producto. Por ejemplo, estás probando una interfaz de usuario que acepta una fecha rango, y tiene una función de calendario emergente para seleccionar la fecha. Echa un vistazo a cómo funciona una función de calendario similar en el sitio web de reservas de una aerolínea.

—
Consulte la bibliografía para obtener enlaces a artículos de Jeff Patton, Gerard Meszaros y otros sobre pruebas de usabilidad.

Las pruebas de usabilidad son un campo bastante especializado. Si estás produciendo un interno aplicación para ser utilizada por unos pocos usuarios que estarán capacitados en su uso, probablemente no necesite invertir mucho en pruebas de usabilidad. Si está escribiendo el directorio de asistencia en línea para una compañía telefónica, la usabilidad podría ser su principal enfoque, por lo que necesita aprender todo lo que pueda al respecto o contratar a un experto en usabilidad.

—
El Capítulo 9, "Kit de herramientas para pruebas empresariales que respaldan al equipo", proporciona más detalles sobre las herramientas que facilitan estas pruebas.

DETRÁS DE LA GUI

En una presentación titulada "El hombre y la máquina" [2007], Jonathan Kohl habló sobre alternativas para probar interfaces. En lugar de pensar siempre en realizar pruebas a través de la interfaz de usuario, considere atacar el problema de otras maneras. Piense en probar todo el sistema desde todos los ángulos que pueda abordar. Considere utilizar herramientas como simuladores o emuladores.

Pruebas API

En el Capítulo 8 y el Capítulo 9, hablamos sobre las pruebas detrás de la GUI para controlar desarrollo. En esta sección, le mostramos que puede ampliar sus pruebas para el API para poder probar diferentes permutaciones y combinaciones.

Una API (interfaz de programación de aplicaciones) es una colección de funciones que puede ser ejecutado por otras aplicaciones o componentes de software. El usuario final normalmente nunca se da cuenta de que existe una API; ella simplemente interactúa con la interfaz de arriba.

Cada llamada API tiene una función específica con una serie de parámetros que aceptan diferentes entradas. Cada variación arrojará un resultado diferente. las pruebas faciles son entradas simples. Los patrones de prueba más complicados ocurren cuando los parámetros trabajan juntos para dar muchas variaciones posibles. A veces los parámetros son opcionales, por lo que es importante que comprenda las posibilidades.

También se deben considerar las condiciones de contorno, tanto para las entradas como para las Resultados previstos. Por ejemplo, utilice cadenas válidas e inválidas para los parámetros, varíe el contenido y varíe la longitud de la entrada de las cadenas.

Otra forma de realizar pruebas es variar el orden de las llamadas a la API. Cambiar la secuencia podría producir resultados inesperados y revelar errores que nunca desaparecerían. se puede encontrar a través de pruebas de UI. Puedes controlar las pruebas mucho más fácilmente que al utilizar la interfaz de usuario.

La historia de Lisa

Mi equipo estaba trabajando en una serie de historias para permitir a los patrocinadores de planes de jubilación cargar archivos de contribuciones de nómina. Escribimos casos de prueba de FitNesse para ilustrar las reglas de análisis de archivos, y el programador también escribió pruebas unitarias para ellos. Cuando se completó la codificación del analizador, quisimos arrojar muchas más combinaciones de datos al analizador, incluidas algunas realmente extrañas, y ver qué sucedía. Podríamos usar el mismo dispositivo que usamos para nuestras pruebas para impulsar el desarrollo, ingresar todas las combinaciones locas que se nos ocurran y ver los resultados. Probamos alrededor de 100 variaciones de datos válidos e inválidos. La Figura 10-3 muestra un ejemplo de algunas de las pruebas que probamos. De esta manera encontramos varios errores en el código.

No guardamos todas estas pruebas en la suite de regresión porque eran solo un medio para probar rápidamente cada combinación que se nos ocurriera. También podríamos haber realizado estas pruebas de forma semiautomática y ad hoc, sin molestarnos en escribir los resultados esperados en la tabla de verificación de resultados y simplemente observar los resultados para asegurarnos de que se vieran correctos.

—Lisa

Build the data

ContributionsFileFormatFixture		parse()	
line			
1	pAyRoLI, 701-00-0003, 40,"\$2,309.01","\$145.09", "125.00", "\$32.88"	valid	valid (mixed case, decimals, spaces)
2	payroll, 701-00-0008,167,"\$999,999,999.99","\$1,500,000",200000,"\$75,000"	valid	valid (outrageous amounts)
3	payroll, 701-00-0011,"3509,175,"\$0.01",25	invalid	invalid (invalid hours)
4	payroll, 701-00-0013,40,1,200,,100,50	invalid	invalid (unquoted comma)
5	payroll, 701-00-0016,40,1347.22,160,56.0,{},0	invalid	invalid (invalid characters)
6	payroll, 701-00-0021, 80, 2300.98, 174.01, 34.90, 84	valid	valid (variable whitespace)
7	loan, 700000041, 4100220,110	valid	valid (no dashes in SSN)
8	loan, 702-00-0054,"\$466"	invalid	invalid (missing field)

Operate and Check

ContributionsFileResultsFixture										
line	ssn	hours	comp	deferral	match	roth	loanId	loan	lineProblem	firstFieldProblem
1	701-00-0003	40	2309.01	145.09	125.00	32.88	null	null	null	null
2	701-00-0008	167	9999999999.99	1500000.00	200000.00	75000.00	null	null	null	null
3	701-00-0011	0	3509.00	175.00	0.01	25.00	null	null	null	invalid hours
4	701-00-0013	40	1.00	200.00	0.00	100.00	null	null	null	extra field
5	701-00-0016	40	1347.22	160.00	56.00	0.00	null	null	null	invalid Roth deferral
6	701-00-0021	80	2300.98	174.01	34.90	84.00	null	null	null	null
7	700-00-0041	null	null	null	null	null	4100220	110.00	null	null
8	702-00-0054	null	null	null	null	null	null	null	null	invalid loan identifier

Figura 10-3 Ejemplo de prueba de reglas de análisis

La historia de Janet

Recientemente trabajé con una aplicación web que interactúa con un sistema heredado a través de una API bien definida. Debido al diseño del sistema heredado y al hecho de que los datos son difíciles de replicar, el equipo aún no ha encontrado una manera de automatizar estas pruebas. Sin embargo, podríamos buscar en los archivos de registro para verificar que se pasaron las entradas correctas y se devolvió el resultado esperado. Es posible realizar valiosas pruebas exploratorias de las API con o sin el beneficio de la automatización.

—Janet

Las llamadas API se pueden desarrollar temprano en el ciclo de vida de una aplicación, lo que significa las pruebas también pueden realizarse temprano. Las pruebas a través de una API pueden dar confianza en el sistema antes de que se desarrolle una interfaz de usuario. Debido a que este tipo de pruebas pueden ser automatizado, deberá trabajar con sus programadores para comprender todos los parámetros y el propósito de cada función. Si sus programadores o Si el equipo de automatización desarrolla un arnés de prueba que sea fácil de usar, debería poder para crear metódicamente un conjunto de casos de prueba que ejerciten la funcionalidad.

Servicios web

Los servicios web generalmente requieren muchas pruebas de seguridad, estrés y confiabilidad.

Consulte el Capítulo 12, "Crítica del producto mediante pruebas orientadas a la tecnología", para

obtener más información sobre este tipo de pruebas.

Los servicios web son una arquitectura basada en servicios que proporciona una interfaz externa para que otros puedan acceder al sistema. Puede haber múltiples partes interesadas, y es posible que ni siquiera sepas quién utilizará tu producto. Tus pruebas deberán confirmar la calidad del servicio que esperan los clientes externos.

Considere los niveles de servicio que se han prometido a los clientes cuando esté creando sus planes de prueba. Tómese el tiempo para realizar pruebas exploratorias para simular el diferentes formas en que los usuarios pueden acceder a los servicios web.

El uso de estándares de servicios web también ofrece otras implicaciones para la actualidad.

herramientas de prueba. Al igual que con las llamadas API, la integración basada en servicios web destaca la Importancia de validar los puntos de interfaz. Sin embargo, también debemos considerar formatos y procesamiento de mensajes, tiempos de cola y tiempos de respuesta de mensajes.

El capítulo 12, "Resumen de los cuadrantes de prueba", tiene un ejemplo de prueba de servicios web.

El uso de herramientas de prueba que utilizan automatización basada en GUI es simplemente inadecuado para un proyecto de servicios web. Un lenguaje específico de dominio que encapsule los detalles de implementación "entre bastidores" funciona bien para probar servicios web.

DOCUMENTOS DE PRUEBA Y DOCUMENTACIÓN

Uno de los componentes del sistema que a menudo se pasa por alto durante las pruebas es documentación. Como desarrolladores ágiles, podemos valorar el software funcional más que la documentación, pero aún así valoramos la documentación! Manuales de usuario y ayuda en línea necesitan validación tanto como el software. Su equipo puede emplear especialistas como redactores técnicos que crean y verifican documentación. Como con todo otros componentes del producto, todo su equipo es responsable de la calidad de la documentación, y eso incluye tanto la copia impresa como la electrónica.

Documentación del usuario

Su equipo podría realizar pruebas del Cuadrante 2 para apoyar al equipo mientras producen documentación; de hecho lo alentamos. El equipo de Lisa escribe código que produce documentos cuyo contenido está especificado por regulaciones gubernamentales, y los programadores pueden escribir gran parte del código primero como prueba. Sin embargo, es difícil para pruebas automatizadas para juzgar si un documento tiene el formato correcto o utiliza una fuente legible. Tampoco pueden evaluar si el contenido de documentos como los manuales de usuario es exacto o útil. Porque la documentación tiene muchos componentes subjetivos, validarla es más bien una actividad crítica.

La historia de Janet

Los redactores técnicos y los evaluadores pueden trabajar en estrecha colaboración. Stephanie, una redactora técnica con la que trabajé en un proyecto, habló con los programadores para entender cómo funcionaba la aplicación. También revisaría la solicitud para asegurarse de haberla escrito correctamente. Esto parecía ser una duplicación del esfuerzo de prueba, por lo que Stephanie y yo nos sentamos y descubrimos un mejor enfoque.

Decidimos trabajar juntos en las historias a medida que se desarrollaban. En algunas historias, Stephanie fue la "probadora" principal y, a veces, yo asumí ese papel. Si yo fuera el líder, crearía mis condiciones y ejemplos de prueba y Stephanie los usaría como base para la documentación. Cuando Stephanie era la líder, escribía su documentación y luego yo la usaba para determinar los casos de prueba.

Hacerlo de esta manera permitió probar la documentación y cuestionar las pruebas antes de ejecutarlas. Trabajar codo con codo de esta manera resultó ser un experimento muy exitoso. La documentación resultante coincidió con el comportamiento del software y fue mucho más útil para los usuarios finales.

—Janet

No olvides consultar también el texto de ayuda. ¿Los enlaces al texto de ayuda son fácilmente identificables? ¿Son consistentes en toda la interfaz de usuario? ¿Se presenta claramente el texto de ayuda? Si se abre en una ventana emergente y los usuarios bloquean las ventanas emergentes en su navegador, ¿cuál es el impacto? ¿La ayuda cubre todos los temas necesarios? En los proyectos de Lisa, el texto de ayuda tiende a ser de baja prioridad, por lo que a menudo no se realiza en absoluto. Esa es una decisión de negocios, pero si considera que un área de la aplicación necesita texto o documentación de ayuda adicional, plantee el problema a su equipo y a sus clientes.

Informes

Otro componente del sistema que a menudo se pasa por alto desde la perspectiva de las pruebas son los informes. Los informes son fundamentales para muchos usuarios a la hora de tomar decisiones, pero a menudo se dejan para el final y no se realizan o no se realizan mal ejecutados. Los informes pueden adaptarse para satisfacer las necesidades específicas del cliente, pero hay muchas herramientas de terceros disponibles para generar informes. Informes puede ser parte de la aplicación misma o generarse a través de un sistema de informes separado para los usuarios finales.

Analizamos los informes de prueba junto con las otras actividades de prueba del Cuadrante 3 en para criticar el producto, pero le recomendamos que también escriba pruebas de informes Quad-rant 2 que guiarán la codificación y ayudarán al equipo a comprender las necesidades del cliente a medida que produce informes. Ciertamente se pueden escribir pruebas primero. Sin embargo, al igual que los documentos, es necesario consultar un informe para saber si es. Es bastante fácil de leer y presenta la información de forma comprensible.

Uno de los mayores desafíos al probar informes no es el formato sino obtener los datos correctos. Cuando se intenta crear datos de prueba para informes, puede resultar difícil obtener una buena sección transversal de datos realistas. También suelen ser los casos extremos que hacen que los informes fallen, por lo que incorporar esos datos extra no es factible. En la mayoría de los casos, es mejor utilizar datos de producción (o datos copiados del sistema de producción a un entorno de prueba) para probar las diferentes variaciones de informes.

La historia de Lisa

Nuestra aplicación incluye una serie de informes, muchos de los cuales ayudan a las empresas a cumplir con los requisitos gubernamentales. Si bien contamos con pruebas de humo automatizadas para cada informe, cualquier cambio en un informe, o incluso una actualización en la herramienta que utilizamos para generar informes, requiere pruebas manuales y visuales exhaustivas. Tenemos que mirar como halcones: ¿un número ha sido truncado por un carácter? ¿Pasó un fragmento de texto a la página siguiente? ¿Se incluyen los datos correctos? Los datos incorrectos o faltantes pueden significar problemas con la agencia reguladora.

Otro desafío es verificar los datos contenidos en el informe. Si tuviera que utilizar la misma consulta que utiliza el informe, no prueba nada. A veces me cuesta crear mis propias consultas SQL para comparar los datos reales con los que aparecen en un informe. Presupuestamos tiempo adicional para probar informes, incluso los que parecen simples. unos.

Debido a que los informes son tan subjetivos, encontramos que diferentes partes interesadas tienen diferentes preferencias sobre cómo se presentan los datos. El administrador del plan que tiene que explicar un informe a un usuario por teléfono tiene una idea diferente de lo que es fácil de entender que el abogado de la empresa que decide qué datos deben incluirse en el informe. Nuestro propietario de producto nos ayuda a lograr el consenso de todas las áreas del negocio.

El contenido y el formato de un informe son importantes, por supuesto, pero en el caso de los informes en línea, la velocidad con la que aparecen también es fundamental. Los administradores de nuestro plan querían total libertad para especificar cualquier rango de fechas para algunos informes del historial de transacciones. Nuestro DBA, que codificó los informes, advirtió que para una gran empresa plan de jubilación, los datos de transacciones de más de unos pocos meses podrían tardar varios minutos en procesarse. Con el tiempo, las empresas crecieron, tuvieron cada vez más transacciones y, finalmente, la interfaz de usuario empezó a agotarse antes de poder entregar el informe. Al realizar las pruebas, pruebe los peores escenarios, que eventualmente podrían convertirse en el escenario más común.

—Lisa


Consulte el Capítulo 8,
"Presentación empresarial
Pruebas que respaldan
al equipo", para
obtener información
sobre el uso de rodajas finas.

Si estás abordando un proyecto que implica muchos informes, no cedas ante la tentación de dejarlos para el final. Incluya algunos informes en cada iteración si puede. Un informe podría ser una sola historia o incluso dividirse en una par de historias. Utilice maquetas para ayudar a los clientes a decidir sobre el contenido y el formato del informe. Encuentre la "porción delgada" o la "ruta crítica" en el informe, codifique eso primero y muéstreselo a su cliente antes de agregar la siguiente porción. El desarrollo incremental funciona tan bien con informes como con otro software.

A veces, los propios clientes no están seguros de cómo debería verse un informe o cómo abordarlo de forma incremental. Y a veces nadie en el equipo anticipa lo difícil que resultará el esfuerzo de prueba.

La historia de Lisa

Al igual que otras cuentas financieras, los planes de jubilación deben proporcionar estados de cuenta periódicos a los titulares de cuentas que detallen todo el dinero que entra y sale de la cuenta. Estos estados de cuenta muestran el cambio de valor entre los saldos inicial y final y otra información pertinente, como los nombres de los beneficiarios de las cuentas. Nuestra empresa quería mejorar los estados de cuenta, como herramienta de marketing y para reducir la cantidad de llamadas de titulares de cuentas que no entendían sus estados de cuenta.

No teníamos acceso a los estados de cuenta de nuestros competidores directos, por lo que el propietario del producto pidió voluntarios para traer estados de cuenta de bancos y otras instituciones financieras para obtener ideas. Meses de discusiones y experimentación con maquetas produjeron un nuevo formato de estado de cuenta, que incluía datos que no estaban en el informe anterior, como los resultados de rendimiento de cada fondo mutuo.

Las historias para desarrollar el nuevo estado de cuenta se distribuyeron a lo largo de dos trimestres de iteraciones. Durante el primer trimestre se realizaron historias para recopilar nuevos datos. Las pruebas resultaron mucho más difíciles de lo que pensábamos. Utilizamos pruebas de FitNesse para verificar la captura de los diferentes elementos de datos, lo que nos dio una falsa sensación de seguridad. Fue difícil cubrir todas las variaciones y nos perdimos algunas con las pruebas automatizadas. Tampoco anticipamos que los cambios para recopilar nuevos datos pudieran tener un efecto adverso en los datos que ya se muestran en los estados de cuenta existentes.

Como resultado, no realizamos pruebas manuales adecuadas de los estados de cuenta. Se nos escaparon errores sutiles. Cuando se ejecutó el trabajo de producir estados de cuenta trimestrales, comenzaron a llegar llamadas de clientes. Tuvimos una lucha loca para diagnosticar y corregir los errores tanto en el código como en los datos. Todo el proyecto se retrasó una cuarta parte mientras descubríamos mejores formas de probar y agregamos comprobaciones internas y un mejor registro del código.

—Lisa

Las iteraciones cortas significan que puede resultar difícil encontrar tiempo para realizar pruebas exploratorias adecuadas y otras actividades del Cuadrante 3. Veamos herramientas que podrían ayudar a acelerar estas pruebas y reserve tiempo para pruebas manuales y visuales vitales.

HERRAMIENTAS PARA AYUDAR CON LAS PRUEBAS EXPLORATORIAS

Las pruebas exploratorias son pruebas manuales. Algunas de las mejores pruebas ocurren porque una persona presta atención a detalles que a menudo pasan desapercibidos si nos siguen.

siguiendo un guión. La intuición es algo que no podemos hacer que una máquina aprenda.

Sin embargo, existen muchas herramientas que pueden ayudarnos en nuestra búsqueda de la excelencia.

Las herramientas no deberían reemplazar la interacción humana; deberían mejorar la experiencia. Las herramientas pueden proporcionar a los evaluadores más poder para encontrar errores difíciles de reproducir que a menudo quedan archivados porque nadie puede controlarlos.

Las pruebas exploratorias no son convencionales, entonces ¿por qué no deberían serlo también las herramientas?

Piense en formas de bajo esfuerzo y alto valor en las que se pueden incorporar herramientas tus pruebas.



Consulte la bibliografía para obtener referencias a Jonathan Kohl, escritos sobre el uso conjunto del poder humano y de automatización para realizar pruebas óptimas.

Las computadoras son buenas para realizar tareas repetitivas y realizar cálculos.

Estas son dos áreas en las que son mucho mejores que los humanos, así que usemos ellos para esas tareas. Debido a que las pruebas deben seguir el ritmo de la codificación, cualquier La ventaja de tiempo que podemos obtener es una ventaja.

En las siguientes secciones, veremos algunas áreas donde la automatización puede aprovechar las pruebas exploratorias. Los que cubrimos son configuración de pruebas, generación de datos de pruebas, monitoreo, simuladores y emuladores.

Configuración de prueba

Pensemos en lo que hacemos cuando probamos. Acabamos de encontrar un error, pero no uno que sea fácilmente reproducible. Estamos bastante seguros de que sucede como resultado de interacciones entre componentes. Volvemos al principio y probamos uno escenario tras otro. Pronto pasamos todo el día intentando reproducir este error.

Pregúntate cómo puedes hacer esto más fácil. Hemos descubierto que uno de los más Las tareas que requieren mucho tiempo son la configuración de la prueba y llegar al punto de partida correcto. para su prueba real. Si utiliza pruebas basadas en sesiones, entonces ya lo sabe cuánto tiempo dedica a configurar la prueba, porque ha estado realizando un seguimiento de esa pérdida de tiempo en particular. Esta es una excelente oportunidad para algunos. automatización.

Las herramientas utilizadas para las pruebas empresariales que respaldan al equipo descrito en El Capítulo 9 también es valioso para las pruebas exploratorias manuales. Automatizado Se pueden ejecutar scripts de prueba funcionales para configurar datos y escenarios para iniciar sesiones de prueba exploratorias. Pruebas configuradas para aceptar parámetros de tiempo de ejecución son particularmente poderosos para establecer un punto de partida para evaluar la producto.

La historia de Lisa

Todos nuestros scripts de prueba Watir aceptan una serie de parámetros de tiempo de ejecución. Cuando necesito un plan de jubilación con un conjunto específico de opciones y tipos específicos de participantes, puedo iniciar uno o dos scripts de Watir con algunas variables configuradas en la línea de comando.

Cuando los scripts se detienen, tengo una sesión de navegador con todos los datos que necesito para las pruebas ya configurados. Esto es tan rápido que puedo probar permutaciones que nunca lograría usando teclas totalmente manuales.

—Lisa

Los scripts de prueba que utiliza para las pruebas de regresión funcional y para guiar el desarrollo no son las únicas herramientas que ayudan a eliminar el aburrimiento de las pruebas exploratorias manuales.

Existen otras herramientas para ayudar a configurar los datos de prueba, así como para ayudarle a evaluar los resultados de sus sesiones de prueba.

Cualquiera que sea la herramienta que esté utilizando, piense en cómo se puede adaptar para ejecutar el escenario una y otra vez con diferentes entradas conectadas. Janet también ha utilizado con éxito Ruby con Watir para configurar pruebas que se ejecutarán varias veces para ayudar a identificar errores. Herramientas que controlan el navegador o la interfaz de usuario de la misma manera que un usuario final haría que sus pruebas sean más confiables porque puede reproducirlas. Vuelva a su monitor y esté atento a cualquier cosa que pueda no verse como debería durante la configuración. Cuando llegues al lugar donde se realizó la prueba, comienza, luego puede usar sus excelentes habilidades de prueba para rastrear el fuente del error.

Generación de datos de prueba

PerlClip es un ejemplo de una herramienta que puede utilizar para probar un campo de texto con diferentes tipos de entradas. James Bach lo ofrece de forma gratuita en su sitio web, www.satisfice.com, y puede ser de gran ayuda para validar campos. Por ejemplo, si tiene un campo que acepta una entrada máxima de 200 caracteres, Probar este campo y sus límites manualmente sería muy tedioso. Utilice PerlClip para crear una cadena, colóquela en su biblioteca de automatización y haga que su herramienta de automatización llame a la cadena para probar el valor.

Herramientas de monitoreo

Herramientas como el comando tail -f de Unix/Linux, o LogWatch de James Bach, pueden ayudar a monitorear los archivos de registro para detectar condiciones de error. Los IDE también proporcionan análisis de registros herramientas. Muchos mensajes de error nunca se muestran en la pantalla, por lo que si probando a través de la GUI, nunca los verás. Familiarícese con herramientas como estas, porque pueden hacer que sus pruebas sean más efectivas y eficientes. Si no eres

Para asegurarse de dónde registra su sistema las advertencias y los errores, consulte a sus desarrolladores. Ellos probablemente tenga muchas ideas sobre cómo monitorear el sistema.

Simuladores

Ver el "Sistema Pruebe el ejemplo en el Capítulo 12, "Resumen de los cuadrantes de prueba", para ver cómo un simulador fue fundamental para probar todo el sistema.

Los simuladores son herramientas utilizadas para crear datos que representan características clave y comportamiento de datos reales para el sistema bajo prueba. Si no tienes acceso a Si utiliza datos reales para su sistema, los datos simulados a veces funcionarán casi tan bien. La otra ventaja de utilizar un simulador es bombar datos a un sistema con el tiempo. Se puede utilizar para ayudar a generar condiciones de error que son difíciles de solucionar. crear en circunstancias normales y puede reducir el tiempo en las pruebas de límites.

Configurar datos y escenarios de prueba es la mitad del panorama. También necesitas tener una forma de observar los resultados de sus pruebas. Consideremos algunas herramientas para ese propósito.

Emuladores

Un emulador duplica la funcionalidad de un sistema para que se comporte como el sistema bajo prueba. Hay muchas razones para utilizar un emulador. Cuando usted Si necesita probar código que interactúe con otros sistemas o dispositivos, los emuladores son inestimable.

Dos ejemplos de emuladores

WestJet, una compañía aérea canadiense, ofrece a los huéspedes la posibilidad de utilizar sus dispositivos móviles para realizar el check-in en aeropuertos que admitan esta función. Al probar esta aplicación, es mejor tanto para los programadores como para los evaluadores probar varios dispositivos lo antes posible. Para que esto sea factible, utilizan emuladores descargables para probar la aplicación Web Check-in de forma rápida y frecuente durante una iteración. Los dispositivos reales, cuyo uso es caro, se pueden utilizar con moderación para verificar la funcionalidad ya probada.

El equipo también creó otro tipo de emulador para ayudar a realizar pruebas con el sistema heredado con el que se interactúa. Los programadores del sistema heredado tienen diferentes prioridades y cronogramas de entrega, y una acumulación de solicitudes. Para evitar que esto detenga el nuevo desarrollo, los programadores de la aplicación web han creado un tipo de emulador para la API en el sistema heredado que devuelve valores predeterminados para llamadas API específicas. Desarrollan contra este emulador y, cuando los cambios reales están disponibles, prueban y realizan modificaciones. Este cambio de proceso les ha permitido avanzar mucho más rápido de lo que antes era posible. Ha demostrado ser una herramienta sencilla pero muy poderosa.

Los emuladores son una herramienta que ayuda a que las pruebas y la codificación sigan avanzando juntas mano a mano. Usarlos es una forma de que las pruebas se mantengan al día con el desarrollo en iteraciones cortas. Al planificar sus lanzamientos e iteraciones, piense en los tipos de herramientas que podrían ayudar a crear escenarios de prueba similares a los de producción. Vea si puede utilizar las herramientas que ya está utilizando para automatizar las pruebas de conducción. desarrollo como ayuda para las pruebas exploratorias.

Impulsar el desarrollo con pruebas es fundamental para el éxito de cualquier proyecto. Sin embargo, Nosotros, los humanos, no siempre cumpliremos del todo correctamente todos los requisitos para el comportamiento deseado del sistema. Nuestros propios expertos en negocios pueden pasar por alto importantes aspectos de funcionalidad o interacción con otras partes del sistema cuando Proporcionan ejemplos de cómo debería funcionar una característica. Tenemos que utilizar técnicas para ayudar a los equipos de clientes y desarrolladores a aprender más sobre el sistema para que puedan seguir mejorando el producto.

RESUMEN

Una gran parte del esfuerzo de prueba se dedica a criticar el producto desde una perspectiva empresarial. Este capítulo le dio algunas ideas sobre los tipos de pruebas. puede hacer para que sus esfuerzos de prueba sean más efectivos.

Demuestre el software a las partes interesadas para obtener comentarios tempranos que ayudarán a dirigir la construcción del material correcto.

Utilice escenarios y flujos de trabajo para probar todo el sistema de un extremo a otro.

Utilice pruebas exploratorias para complementar la automatización y aprovechar el intelecto y las percepciones humanas.

Sin tener en cuenta la usabilidad al realizar pruebas y codificar, las aplicaciones pueden convertirse en estanterías. Esté siempre atento a cómo se utiliza el sistema.

Probar detrás de la GUI es la forma más efectiva de acceder a la funcionalidad de la aplicación. Investigue un poco para ver cómo puede abordar su aplicación.

Incorpora todo tipo de pruebas para realizar una buena suite de regresión.

No se olvide de probar la documentación y los informes.

Las herramientas de automatización pueden realizar tareas tediosas y repetitivas, como la configuración de datos y escenarios de prueba, y liberar más tiempo para importantes pruebas exploratorias manuales.

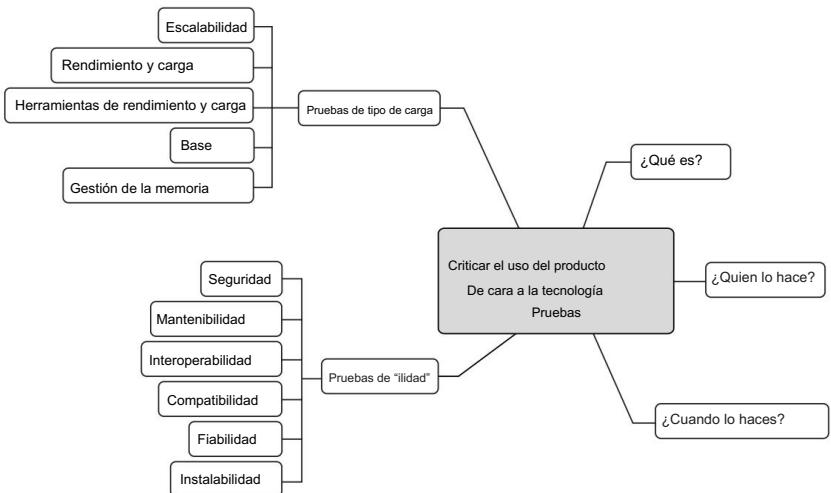
Las herramientas que ya está utilizando para automatizar las pruebas funcionales también pueden resultar útiles para aprovechar las pruebas exploratorias.

Las herramientas de monitoreo, uso de recursos y análisis de registros integradas en los sistemas operativos e IDE ayudan a los evaluadores a evaluar el comportamiento de la aplicación. Los simuladores y emuladores permiten realizar pruebas exploratorias incluso cuando no se puede duplicar el entorno de producción exacto. Incluso cuando se utilizan pruebas para impulsar el desarrollo, los requisitos para el comportamiento deseado o la interacción con otros sistemas pueden pasarse por alto o malinterpretarse. Las actividades del cuadrante 3 ayudan a los equipos a seguir agregando valor al producto.

Esta página se dejó en blanco intencionalmente.

Capítulo 11

CRITICAR EL USO DEL PRODUCTO PRUEBAS DE TECNOLOGÍA



Este capítulo se centra en la esquina inferior derecha de nuestro cuadrante de pruebas. Hemos analizado cómo impulsar el desarrollo con pruebas tanto orientadas a los negocios como a la tecnología. Una vez escrito el código, ya no impulsamos el desarrollo, sino que buscamos formas de criticar el producto. En el capítulo anterior, examinamos formas de criticar desde un punto de vista empresarial. Ahora analizamos formas de criticar desde un punto de vista tecnológico. Estas pruebas son un medio importante para evaluar si nuestro producto ofrece el valor comercial adecuado.

INTRODUCCIÓN AL CUADRANTE 4 Las

historias individuales son piezas del rompecabezas, pero hay más en una aplicación que eso. Las pruebas tecnológicas que critican el producto son más

más preocupados por los requisitos no funcionales que por los funcionales.

Nos preocupamos por las deficiencias del producto desde el punto de vista técnico.

En lugar de utilizar el lenguaje del dominio empresarial, describimos los requisitos utilizando un vocabulario del dominio de programación. Esta es la provincia del Cuadrante 4 (ver Figura 11-1).

Los requisitos no funcionales incluyen cuestiones de configuración, seguridad, rendimiento, gestión de la memoria, las "utilidades" (por ejemplo, fiabilidad, interoperabilidad y escalabilidad), recuperación e incluso conversión de datos. No todos los proyectos se preocupan por todos estos temas, pero es una buena idea tener una lista de verificación para asegurarse de que el equipo piense en ellos y le pregunte al cliente qué tan importante es cada uno.

Nuestro cliente debe pensar en todos los atributos y factores de calidad que son importantes y hacer concesiones informadas. Sin embargo, muchos clientes se centran

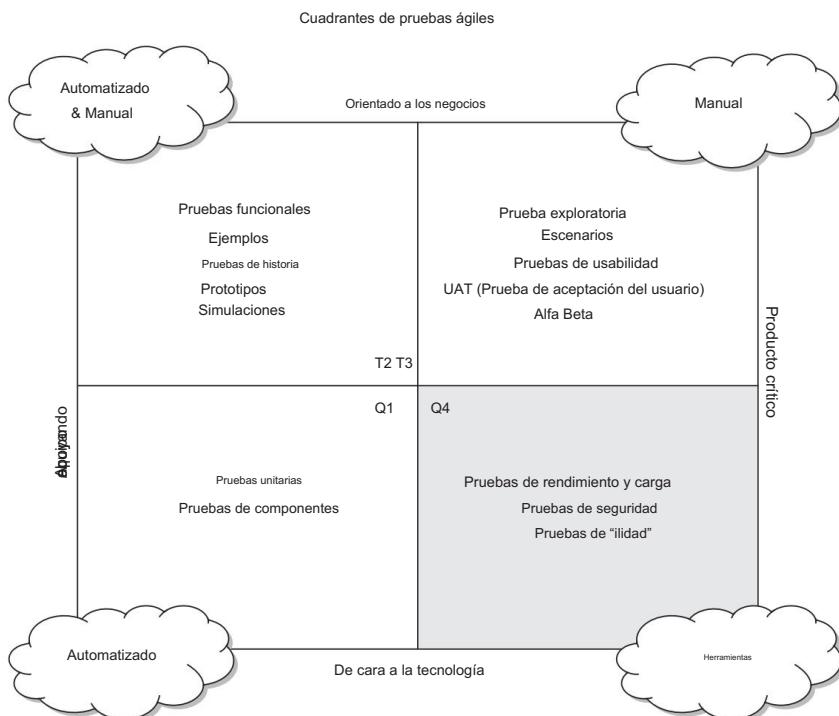


Figura 11-1 Pruebas del cuadrante 4

en el lado comercial de la aplicación y no comprenden la importancia de muchos requisitos no funcionales en su función de ayudar a definir el nivel de calidad necesaria para el producto. Podrían suponer que el equipo de desarrollo solo se encargará de cuestiones como el rendimiento, la confiabilidad y la seguridad.

Creemos que el equipo de desarrollo tiene la responsabilidad de explicar el consecuencias de no abordar estos requisitos no funcionales o interfuncionales. Realmente todos somos parte de un equipo de producto que quiere entregar buen valor, y estos factores orientados a la tecnología podrían exponer problemas decisivos.

Muchos de estos problemas no funcionales y multifuncionales se consideran de bajo riesgo para muchas aplicaciones y, por lo tanto, no se agregan al plan de prueba. Sin embargo, Cuando planifiques tu proyecto, debes pensar en los riesgos de cada uno. de estas áreas, abórdalas en su plan de prueba e incluya las herramientas y recursos necesarios para probarlas en su plan de proyecto.

La historia de Lisa

En el pasado, especialistas en áreas como pruebas de rendimiento y seguridad me preguntaron por qué no escuchaban mucho sobre las pruebas de "ilidad" en conferencias ágiles o en publicaciones sobre desarrollo ágil. Al igual que Janet, siempre he considerado críticas estas áreas de las pruebas, por lo que no era ésta mi percepción. Pero mientras lo pensaba, tuve que aceptar que este no era un tema muy discutido en ese momento (aunque eso es cambiado recientemente).

¿Por qué las discusiones ágiles no incluirían consideraciones tan importantes como las pruebas de carga? Mi teoría es que se debe a que el desarrollo ágil está impulsado por los clientes, a partir de historias de usuarios. Los clientes simplemente asumen que el software se diseñará para adaptarse adecuadamente a la carga potencial, a un ritmo de rendimiento razonable. No siempre se les ocurre verbalizar esas preocupaciones. Si no se les pide que los aborden, los programadores pueden pensar o no en priorizarlos. Creo que un área en la que los evaluadores han contribuido en gran medida a los equipos ágiles es al plantear preguntas como: "¿Cuántos usuarios simultáneos debería admitir la aplicación?" y "¿Cuál es el tiempo promedio de respuesta requerido?"

—Lisa

Debido a que los tipos de pruebas en este cuadrante son tan diversos, daremos ejemplos de herramientas que podrían ser útiles a medida que avanzamos en lugar de un conjunto de herramientas separado. sección. Las herramientas, ya sean de cosecha propia o adquiridas, son esenciales para tener éxito Esfuerzos de prueba del cuadrante 4. Aún así, las personas que hacen el trabajo cuentan, así que Considere quién en un equipo ágil puede realizar estas pruebas.

¿QUIEN LO HACE ?

Toda la literatura ágil habla de equipos generalistas; cualquiera debería ser capaz de elegir una tarea y realizarla. Sabemos que eso no siempre es práctico, pero La idea es poder compartir el conocimiento para que las personas no se conviertan en silos. de información.

Sin embargo, hay muchas tareas que necesitan conocimientos especializados. Un buen ejemplo son las pruebas de seguridad. No estamos hablando de seguridad dentro de una aplicación, como quién tiene derechos de acceso para administrarlo. Porque ese tipo de seguridad es Realmente forma parte de los requisitos funcionales y será cubierto por historias periódicas, verificando que funciona dentro de los primeros tres cuadrantes. Estaban Hablando sobre buscar fallas de seguridad externas y conocer los tipos de vulnerabilidades en los sistemas que explotan los piratas informáticos. Ese es un conjunto de habilidades especializadas.

Las pruebas de rendimiento pueden ser realizadas por evaluadores y programadores que colaboren y construir herramientas simples para sus necesidades específicas. Algunas organizaciones compran herramientas de prueba de carga que requieren miembros del equipo que se especialicen en eso. herramienta para construir los guiones y analizar e interpretar los resultados. Puede resultar difícil para una organización de desarrollo de software, especialmente una pequeña, tener suficientes recursos para duplicar una carga de nivel de producción precisa para una prueba, por lo que Es posible que se necesiten proveedores externos de pruebas de rendimiento.

El Capítulo 15, "Actividades del probador en la planificación de versiones o temas", explica cómo planificar el trabajo con equipos externos.

Las organizaciones más grandes pueden tener grupos como expertos en bases de datos que su equipo puede utilizar para ayudar con la conversión de datos, grupos de seguridad que le ayudarán identificar riesgos para su aplicación, o un equipo de soporte de producción que pueda ayudar prueba la recuperación o la comutación por error. Construya una relación cercana con estos especialistas. Necesitarán trabajar juntos como un equipo virtual para recopilar la información que necesita sobre su producto.

Cuanto más diversas sean las habilidades de su equipo, menos probable será que necesite consultores externos para ayudarle. Identifique los recursos que necesita para cada uno proyecto. Muchos equipos consideran que un buen probador técnico o un herrero puede asumir muchas de estas tareas. Si alguien que ya está en el equipo puede aprender cualquier conocimiento especializado que se requiera, genial; de lo contrario, traiga la experiencia que necesita.

Habilidades dentro del equipo

Jason Holzer, propietario de producto para pruebas de propiedades (rendimiento, seguridad, estabilidad y confiabilidad) en Ultimate Software, nos dice que un buen programador puede escribir un motor multiproceso para llamar a una función simultáneamente y probar el rendimiento. Jason siente que los equipos ágiles tienen las habilidades para realizar sus propias pruebas de desempeño; es posible que simplemente no se den cuenta.

Las pruebas de rendimiento requieren un entorno dedicado y controlado.

Se necesitan algunas herramientas especializadas, como un generador de perfiles para medir el rendimiento del código. Pero, en opinión de Jason, las pruebas de rendimiento, estabilidad, escalabilidad y confiabilidad (PSR) pueden y deben realizarse a nivel unitario. Existe una mentalidad que sostiene que estas pruebas son demasiado complejas y requieren especialistas cuando en realidad los equipos sí poseen las habilidades necesarias.

Jason considera que la conciencia de los aspectos "PSR" del código debe ser parte de la cultura del equipo.

Si las partes interesadas dan alta prioridad al rendimiento, la estabilidad, la escalabilidad y similares, Jason recomienda que el equipo hable sobre formas de verificar estos aspectos de la aplicación. Cuando los equipos comprenden la prioridad de cualidades como el rendimiento y la confiabilidad, descubren cómo mejorar su código para garantizarlas. No necesitan depender de un equipo externo especializado. Jason explica su punto de vista.

La resistencia potencial que veo hoy a este plan es que alguien crea que los programadores no saben cómo realizar pruebas de PSR y que será necesario mucho entrenamiento. En mi opinión, una afirmación más precisa es que los programadores no son conscientes de que las pruebas de PSR son una alta prioridad y una clave para la calidad. No creo que tenga nada que ver con saber cómo realizar la prueba de PSR. La prueba PSR es una combinación de matemáticas, ciencias, análisis, programación y resolución de problemas. Estoy dispuesto a apostar que si llevara a cabo una competencia en cualquier organización de desarrollo de software en la que le pidiera a cada equipo que implementara un algoritmo de búsqueda de árbol, y el equipo con el algoritmo más rápido ganaría, cada equipo haría pruebas de PSR y proporcionaría PSR. métricas sin enseñarles nada nuevo.

Las pruebas de PSR en realidad solo me dicen "¿Qué tan rápido?" (actuación), "¿Cómo ¿largo?" (estabilidad), "¿Con qué frecuencia?" (confiabilidad) y "¿Cuánto?" (escalabilidad). Entonces, mientras exista conciencia y la organización se plantea seriamente esas preguntas con todo lo que desarrolle, las pruebas de PSR se integrarán exitosamente en un equipo.

Eche un segundo vistazo a las habilidades que su equipo ya posee y haga una lluvia de ideas sobre los tipos de pruebas de "ilidad" que se pueden realizar con los recursos que ya tiene. Si necesita equipos externos, planifíquelo en su planificación de lanzamiento e iteración.

Independientemente de si su equipo aporta o no recursos adicionales para En este tipo de pruebas, su equipo sigue siendo responsable de asegurarse de que se realicen las pruebas mínimas. La información que proporcionan estas pruebas puede dar lugar a nuevos historias y tareas en áreas como cambiar la arquitectura para una mejor escalabilidad o implementar una solución de seguridad para todo el sistema. Asegúrese de completar el circuito de retroalimentación desde las pruebas que critican el producto hasta las pruebas que impulsan cambios que mejorará los aspectos no funcionales del producto.

El hecho de que este sea el cuarto de los cuatro cuadrantes de pruebas ágiles no significa que estas pruebas sean las últimas. Su equipo debe pensar en cuándo realizar pruebas de rendimiento, seguridad y "ilidad" para asegurarse de que su producto brinde el valor comercial adecuado.

¿ CUANDO LO HACES ?

Al igual que con las pruebas funcionales, cuanto antes se completen las pruebas tecnológicas que respaldan al equipo, más barato será solucionar cualquier problema que se encuentre. Sin embargo, muchas de las pruebas multifuncionales son costosas y difíciles de realizar en pequeñas partes.

Se pueden escribir historias técnicas para abordar requisitos específicos, como por ejemplo: "Como usuario Abby, necesito recuperar el informe X en menos de 20 segundos para poder tomar una decisión rápidamente". Esta historia trata sobre el rendimiento y requiere que se escriban pruebas especializadas, y se puede hacer junto con la historia para codificar el informe o en una iteración posterior.

Considere una fila separada en su guión gráfico para las tareas que necesita el producto en su conjunto. El equipo de Lisa utiliza esta área para colocar tarjetas como "Evaluar herramientas de prueba de carga" o "Establecer una línea base de prueba de rendimiento". Janet ha utilizado con éxito tarjetas de diferentes colores para mostrar que la historia está destinada a uno de los roles de experto tomados de otras áreas de la organización.

Es posible que algunas pruebas de rendimiento deban esperar hasta que se haya creado gran parte de la aplicación si está intentando establecer una línea de base de los flujos de trabajo completos de un extremo a otro. Si el rendimiento y la confiabilidad son una prioridad máxima, debe encontrar una manera de probarlos al principio del proyecto. Priorice las historias para que un hilo de acero o una rebanada fina se complete temprano. Debería poder crear una prueba de rendimiento que pueda ejecutarse y continuar ejecutándose a medida que agrega más y más funciones al flujo de trabajo. Esto puede permitirle detectar problemas de rendimiento con antelación y rediseñar la arquitectura del sistema para realizar mejoras. Para muchas aplicaciones, la funcionalidad correcta es irrelevante sin el rendimiento necesario.

El momento de pensar en las pruebas no funcionales es durante el lanzamiento o la planificación del tema. Planee comenzar temprano, abordando pequeños incrementos según sea necesario. Para cada iteración, vea qué tareas necesita su equipo para determinar si el diseño del código es confiable, escalable, utilizable y seguro. En la siguiente sección, veremos algunos tipos diferentes de pruebas del Cuadrante 4.

Pruebas de rendimiento desde el principio

Ken De Souza, desarrollador/probador de software en NCR [2008], respondió a una pregunta en la lista de correo de pruebas ágiles sobre cuándo realizar pruebas de estrés y de rendimiento en un proyecto ágil con una explicación de cómo aborda las pruebas de rendimiento.

Sugeriría diseñar sus pruebas de rendimiento desde el principio. Desarrollamos datos desde la primera iteración y ejecutamos una prueba de rendimiento simple para asegurarnos de que todo se mantenga unido. Esto es más para ver que la funcionalidad de los scripts de rendimiento se mantiene unida.

Utilicé JMeter porque puedo conectar FTP, SOAP, HTTP, RegEx, etc., todo desde unos pocos subprocessos, con una sola instancia ejecutándose. Puedo probar mis llamadas desde el principio (o al menos tener la infraestructura para hacerlo).

Mi objetivo final es que cuando el producto esté a punto de lanzarse, no tenga que realizar la prueba de rendimiento; Sólo tengo que subir los hilos y soltarme. Todas mis métricas y tareas ya se han probado durante meses, por lo que estoy bastante seguro de que cualquiera puede ejecutar mi prueba de rendimiento.

Las pruebas de rendimiento se pueden abordar utilizando principios ágiles para crear herramientas y probar componentes de forma incremental. Al igual que con las funciones del software, concéntrese en obtener la información de rendimiento que necesita, una pequeña parte a la vez.

PRUEBAS DE "ILIDAD"

Si pudiéramos centrarnos en el comportamiento y la funcionalidad deseados de la aplicación, la vida sería muy sencilla. Desafortunadamente, tenemos que preocuparnos por cualidades tales como seguridad, mantenibilidad, interoperabilidad, compatibilidad, confiabilidad e instalación. Echemos un vistazo a algunas de estas "ilidades".

Seguridad

Vale, no termina en -ilidad, pero lo incluimos en el grupo "ilidad" porque Utilice pruebas tecnológicas para evaluar los aspectos de seguridad del producto. La seguridad es una prioridad absoluta para todas las organizaciones hoy en día. Toda organización necesita garantizar la confidencialidad y la integridad de su software. Ellos queremos verificar conceptos como el no repudio, una garantía de que el mensaje ha sido enviado por la parte que afirma haberlo enviado y recibido por la parte que afirma haberlo recibido. La aplicación necesita realizar la correcta autenticación, confirmación de la identidad de cada usuario y autorización, para

para permitir al usuario acceder únicamente a los servicios que está autorizado a utilizar. Pruebas No es fácil abordar tantos aspectos diferentes de la seguridad.

En la prisa por ofrecer funcionalidad, tanto los expertos en negocios como en desarrollo Es posible que los equipos de organizaciones recién creadas no estén pensando primero en la seguridad. Sólo quieren que algún software funcione para poder hacer negocios. La autorización es a menudo el único aspecto de las pruebas de seguridad que consideran parte de la funcionalidad empresarial.

La historia de Lisa

Mi equipo actual es un buen ejemplo. La empresa estaba interesada en automatizar la funcionalidad para gestionar los planes 401(k). Se esforzaron por proteger el software y los datos, pero no era una prioridad de prueba. Cuando “entendí la religión” después de escuchar algunas buenas presentaciones sobre pruebas de seguridad en conferencias, compré un libro sobre pruebas de seguridad y comencé a piratear el sitio. Encontré algunos problemas graves que solucionamos, pero nos dimos cuenta de que necesitábamos un enfoque integral para garantizar la seguridad. Escribimos historias para implementar esto. También comenzamos a incluir un “seguridad” Tarjeta de tareas con cada historia para que tengamos en cuenta las necesidades de seguridad mientras desarrollamos y probamos.

—Lisa

Presupuestar este tipo de trabajos tiene que ser una prioridad empresarial. Hay una gama de alternativas disponibles, dependiendo de las prioridades y recursos de su empresa. Comprenda sus necesidades y los riesgos antes de invertir mucho tiempo y energía.

La historia de Janet

Un equipo con el que trabajé tiene un equipo de seguridad corporativo independiente. Cada vez que se agrega una funcionalidad a la aplicación que podría exponer una falla de seguridad, el equipo corporativo ejecuta la aplicación a través de una aplicación de prueba de seguridad y genera un informe para el equipo. Realiza pruebas estáticas utilizando una sonda de caja negra enlazada en el código y ha expuesto algunas áreas débiles que los desarrolladores pudieron abordar. No ofrece una imagen general del nivel de seguridad de la aplicación, pero eso no se consideró una preocupación importante.

—Janet

Los evaluadores expertos en pruebas de seguridad pueden realizar pruebas de seguridad basadas en riesgos. pruebas, que se basan en el análisis del riesgo arquitectónico, los patrones de ataque, o casos de abuso y mal uso. Cuando se requieran habilidades especializadas, traiga lo que necesita, pero el equipo aún es responsable de asegurarse de que las pruebas se hace.

Existe una variedad de herramientas automatizadas para ayudar con la verificación de seguridad.

Las herramientas de análisis estático, que pueden examinar el código sin ejecutar la aplicación, pueden detectar posibles fallos de seguridad en el código que, de otro modo, no aparecerían en años. Las herramientas de análisis dinámico, que se ejecutan en tiempo real, pueden

Pruebe vulnerabilidades como inyección SQL y secuencias de comandos entre sitios. Manual

Las pruebas exploratorias realizadas por un evaluador de seguridad experto son indispensables para detectar problemas que las pruebas automatizadas pueden pasar por alto.

Perspectivas de las pruebas de seguridad

Las pruebas de seguridad son un tema muy amplio en sí mismo. Grig Gheorghiu comparte algunos aspectos destacados sobre los recursos que pueden ayudar a los equipos ágiles con las pruebas de seguridad.

Al igual que las pruebas funcionales, las pruebas de seguridad se pueden realizar desde dos perspectivas: de adentro hacia afuera (pruebas de caja blanca) y de afuera hacia adentro (pruebas de caja negra). Las pruebas de seguridad de adentro hacia afuera suponen que el código fuente de la aplicación bajo prueba está disponible para los evaluadores. El código se puede analizar estéticamente con una variedad de herramientas que intentan descubrir errores de codificación comunes que pueden hacer que la aplicación sea vulnerable a ataques como desbordamientos de búfer o ataques de formato de cadena.

El hecho de que los evaluadores tengan acceso al código fuente de la aplicación también significa que pueden mapear lo que algunos libros llaman "la superficie de ataque" de la aplicación, que es la lista de todas las entradas y recursos utilizados por el programa bajo prueba. Armados con un conocimiento de la superficie de ataque, los evaluadores pueden aplicar una variedad de técnicas que intentan romper la seguridad de la aplicación. Una clase muy efectiva de tales técnicas se llama fuzzing y se basa en la inyección de fallas. Utilizando esta técnica, los evaluadores intentan hacer que la aplicación falle alimentándola con varios tipos de entradas (de ahí el término). Estas entradas pueden ser cadenas cuidadosamente diseñadas utilizadas en ataques de inyección SQL, cambios de bytes aleatorios en archivos de entrada determinados o cadenas ~~aleatorias de falla~~ utilizadas como argumentos de línea de comando.

El enfoque de afuera hacia adentro es el que utilizan principalmente los atacantes que intentan penetrar los servidores o la red que aloja su aplicación. Como evaluador de seguridad, debe tener la misma mentalidad que los atacantes, lo que significa que debe usar su creatividad para descubrir y explotar vulnerabilidades en su propia aplicación. También debe mantenerse actualizado con las últimas noticias de seguridad y actualizaciones relacionadas con la plataforma/

sistema operativo en el que se ejecuta su aplicación, lo cual no es una tarea fácil.

Entonces, ¿qué deben hacer los evaluadores ágiles cuando se enfrentan a la tarea aparentemente insuperable de probar la seguridad de su aplicación? Aquí hay algunos pasos prácticos y pragmáticos que cualquiera puede seguir:

1. Adopte un proceso de integración continua (CI) que ejecute periódicamente un conjunto de pruebas automatizadas en su aplicación.

2. Aprenda a utilizar una o más herramientas de análisis de código estático de código abierto. Agregue un paso a su proceso de CI que consista en ejecutar estas herramientas en el código de su aplicación. Marque el paso como fallido si las herramientas encuentran vulnerabilidades críticas.
3. Instale un escáner de vulnerabilidades de seguridad automatizado como Nessus (<http://www.nessus.org/nessus/>). Nessus se puede ejecutar en modo de línea de comandos, sin GUI, lo que lo hace adecuado para su inclusión en una herramienta de CI. Agregue un paso a su proceso de CI que consista en ejecutar Nessus en su aplicación. Capture la salida de Nessus en un archivo y analice ese archivo en busca de agujeros de seguridad de gran importancia encontrados por el escáner. Marque el paso como fallido cuando encuentre dichos agujeros.
4. Aprenda a utilizar una o más herramientas de fuzzing de código abierto. Agregue un paso a su proceso de CI que consista en ejecutar estas herramientas en el código de su aplicación. Marque el paso como fallido si las herramientas encuentran vulnerabilidades críticas.

Al igual que con cualquier esfuerzo de prueba automatizado, ejecutar estas herramientas no garantiza que su código y su aplicación estén libres de defectos de seguridad.

Sin embargo, ejecutar estas herramientas contribuirá en gran medida a mejorar la calidad de su aplicación en términos de seguridad. Como siempre, se aplica la regla 80/20. Estas herramientas probablemente encontrarán el 80% de los errores de seguridad más comunes y requerirán el 20% de su presupuesto de seguridad.

Para encontrar el 20% restante de los defectos de seguridad, le recomendamos gastar el otro 80% de su presupuesto de seguridad en expertos en seguridad de alta calidad. Podrán probar exhaustivamente la seguridad de su aplicación mediante el uso de técnicas como inyección SQL, inyección de código, inclusión remota de código y secuencias de comandos entre sitios. Si bien existen algunas herramientas que intentan automatizar algunas de estas técnicas, no son rival para un profesional capacitado que se toma el tiempo para comprender el funcionamiento interno de su aplicación para poder diseñar el ataque perfecto contra ella.

Las pruebas de seguridad pueden ser intimidantes, así que reserve tiempo para adoptar una mentalidad de hacker y decidir cuál es el enfoque correcto para la tarea en cuestión. Utilice los recursos que sugiere Grig para informarse. Aproveche estas herramientas y técnicas para lograr pruebas de seguridad con un retorno de la inversión razonable.

Sólo este breve vistazo a las pruebas de seguridad muestra por qué la capacitación especializada y las herramientas son muy importantes para hacer un buen trabajo. Para la mayoría de las organizaciones, esto la prueba es absolutamente necesaria. Una intrusión de seguridad podría ser suficiente para sacar una empresa del negocio. Incluso si la probabilidad fuera baja, lo que está en juego son demasiado elevados para posponer estas pruebas.

Un código cuyo mantenimiento cuesta mucho podría no acabar con la rentabilidad de una organización tan rápido como una violación de seguridad, pero podría conducir a una muerte larga y lenta. En el En la siguiente sección consideramos formas de verificar la mantenibilidad.

Mantenibilidad

La mantenibilidad no es algo que sea fácil de probar. En proyectos tradicionales, a menudo se realiza mediante el uso de inspecciones o revisiones de código completo. Los equipos ágiles suelen utilizar programación en pares, que tiene incorporada una revisión continua del código. Allá Hay otras formas de garantizar que el código y las pruebas se mantengan.

Aleñamos a los equipos de desarrollo a desarrollar estándares y directrices que siguen para el código de la aplicación, los marcos de prueba y las pruebas mismas. Equipos que desarrollan sus propios estándares, en lugar de que ellos sean establecidos por algún otro equipo independiente, será más probable que los sigan porque tienen sentido para ellos.

Los tipos de estándares a los que nos referimos incluyen convenciones de nomenclatura para métodos, nombres o nombres de prueba. Todas las pautas deben ser fáciles de seguir y realizar mantenibilidad más fácil. Algunos ejemplos son: "El éxito siempre es cero y el fracaso debe ser un valor negativo", "Cada clase o módulo debe tener una sola responsabilidad" o "Todas las funciones deben ser de entrada y salida únicas".

Los estándares para desarrollar la GUI también hacen que la aplicación sea más comprobable. y mantenible, porque los evaluadores saben qué esperar y no necesitan preguntarse si un comportamiento es correcto o incorrecto. También aumenta la capacidad de prueba si están automatizando pruebas desde la GUI. Estándares simples como "Usar nombres para todos los objetos GUI en lugar de utilizar de forma predeterminada el identificador asignado por la computadora" o "No puedes tener dos campos con el mismo nombre en una página" ayuda al equipo alcanzar un nivel en el que el código sea mantenible, al igual que las pruebas automatizadas que proporcionarle cobertura.

El código mantenible admite la propiedad del código compartido. Es mucho más fácil para un programador para moverse de un área a otra si todo el código está escrito en el mismo estilo y fácilmente comprensible para todos los miembros del equipo. La complejidad agrega riesgo y también hace que el código sea más difícil de entender. El valor XP de la simplicidad debe aplicarse al código. Los estándares de codificación simples también pueden incluir pautas como "Evite la duplicación: no copie y pegue métodos". Estos mismos Los conceptos se aplican a los marcos de prueba y a las pruebas mismas.

La mantenibilidad también es un factor importante para las pruebas automatizadas. herramientas de prueba se han quedado atrás de las herramientas de programación en características que las hacen fáciles de usar.

mantener, como complementos IDE para facilitar la escritura y el mantenimiento de scripts de prueba. más simple y más eficiente. Esto está cambiando rápidamente, así que busque herramientas que proporcionen una fácil refactorización y búsqueda y reemplazo, y otras utilidades que Facilite la modificación de los scripts.

La mantenibilidad de la base de datos también es importante. El diseño de la base de datos debe ser flexible y utilizable. Cada iteración puede traer tareas para agregar o eliminar tablas, columnas, restricciones o activadores, o para realizar algún tipo de conversión de datos. Estas tareas se convierten en un cuello de botella si el diseño de la base de datos es deficiente o si la base de datos está repleta de datos no válidos.

La historia de Lisa

Un error de regresión grave no se detectó y causó problemas de producción. Hicimos una prueba que debería haber detectado el error. Sin embargo, faltaba una restricción en el esquema utilizado por el conjunto de regresión. Nuestros esquemas de prueba habían crecido de manera desordenada a lo largo de los años. Algunos tenían columnas que ya no existían en el esquema de producción. A algunos les faltaban varias restricciones, desencadenantes e índices. Nuestro DBA tuvo que realizar cambios manualmente en cada esquema según fuera necesario para cada historia en lugar de ejecutar el mismo script en cada esquema para actualizarlo. Presupuestamos tiempo en varios sprints para recrear todos los esquemas de prueba de modo que fueran idénticos y también coincidieran con la producción.

—Lisa

Planifique tiempo para evaluar el impacto de la base de datos en la velocidad del equipo y refactorícela. tal como lo hace con el código de producción y prueba. Mantenibilidad de todos los aspectos del Los entornos de aplicación, prueba y ejecución son más una cuestión de evaluación. y refactorización que las pruebas directas. Si su velocidad está disminuyendo, ¿es porque es difícil trabajar en partes del código o es que la base de datos es difícil de modificar?

Interoperabilidad

La interoperabilidad se refiere a la capacidad de diversos sistemas y organizaciones. trabajar juntos y compartir información. Las pruebas de interoperabilidad analizan Funcionalidad de extremo a extremo entre dos o más sistemas en comunicación. Estas pruebas se realizan en el contexto del usuario (humano o una aplicación de software) y analizan el comportamiento funcional.

En el desarrollo ágil, las pruebas de interoperabilidad se pueden realizar en las primeras etapas del ciclo de desarrollo. Contamos con un sistema implementable y funcional al final de cada iteración para que podamos implementar y configurar pruebas con otros sistemas.



En el capítulo 20,
"Entrega exitosa",
discutimos
más sobre el
importancia de este
nivel de prueba.

El cuadrante 1 incluye pruebas de integración de código, que son pruebas entre componentes, pero existe un nivel completamente diferente de pruebas de integración en sistemas empresariales.

Es posible que se encuentre integrando sistemas a través de interfaces abiertas o propietarias. La API que desarrolle para su sistema podría permitir a sus usuarios fácilmente establezca un marco para que puedan probar fácilmente. Pruebas más fáciles para su cliente hace que la aceptación sea más rápida.

En un proyecto en el que trabajó Janet, se instalaron sistemas de prueba en las instalaciones del cliente. para que pudieran comenzar a integrarlos con sus propios sistemas lo antes posible. Las interfaces de los sistemas existentes se cambiaron según fue necesario y se probaron con cada nuevo despliegue.

Si el sistema en el que trabaja su equipo tiene que funcionar junto con sistemas externos, es posible que no pueda representarlos a todos en sus entornos de prueba, excepto con stubs y drivers que simulan el comportamiento de los demás sistemas o equipo. Esta es una situación en la que se realizan pruebas una vez finalizado el desarrollo. podría ser inevitable. Es posible que tenga que programar un tiempo de prueba en un entorno de prueba compartido por varios equipos.

Considere todos los sistemas con los que el suyo necesita comunicarse y asegúrese de planificar con anticipación para tener un entorno apropiado para las pruebas ellos juntos. También necesitará planificar recursos para probar que su aplicación es compatible con los diversos sistemas operativos, navegadores, clientes, servidores y hardware con los que podría usarse. Discutiremos la compatibilidad.

prueba a continuación.

Compatibilidad

El tipo de proyecto en el que está trabajando dicta cuántas pruebas de compatibilidad se requieren. Si tiene una aplicación web y sus clientes están en todo el mundo, deberá pensar en todos los tipos de navegadores y sistemas operativos. Si entrega una aplicación empresarial personalizada, probablemente pueda reducir la cantidad de pruebas de compatibilidad, ya que es posible que pueda para dictar qué versiones son compatibles.

Como cada nueva pantalla se desarrolla como parte de una historia de interfaz de usuario, es una buena idea comprobar su operatividad en todos los navegadores compatibles. Una tarea sencilla puede ser agregado a la historia para probar en todos los navegadores.

Una organización en la que trabajaba Janet tuvo que probar la compatibilidad con la lectura. Software para personas con discapacidad visual. Aunque la empresa no tenía una prueba formal

laboratorio, tenía máquinas de prueba disponibles cerca del área del equipo para un fácil acceso. El Los evaluadores realizaron comprobaciones periódicas para asegurarse de que la nueva funcionalidad aún estuviera disponible. compatible con herramientas de terceros. Fue fácil solucionar los problemas que descubierto temprano durante el desarrollo.

Tener máquinas de prueba disponibles con diferentes sistemas operativos o navegadores. o aplicaciones de terceros que necesitan funcionar con el sistema bajo prueba hace Es más fácil para los evaluadores garantizar la compatibilidad con cada nueva historia o al mismo tiempo. final de una iteración. Cuando comiences un nuevo tema o proyecto, piensa en el recursos que podría necesitar para verificar la compatibilidad. Si estás empezando con una marca nuevo producto, es posible que tenga que construir un laboratorio de pruebas para ello. Asegúrate de que tu equipo obtiene información sobre el hardware, los sistemas operativos, los navegadores, y versiones de cada uno. Si el porcentaje de uso de una nueva versión del navegador ha crecido lo suficiente, podría ser el momento de comenzar a incluir esa versión en su pruebas de compatibilidad.

Cuando seleccione o cree herramientas de prueba funcionales, asegúrese de que haya una manera fácil de ejecutar el mismo script con diferentes versiones de navegadores, sistemas operativos y hardware. Por ejemplo, el equipo de Lisa podría utilizar el mismo conjunto de regresión GUI pruebas en cada uno de los servidores que se ejecutan en Windows, Solaris y Linux. Los scripts de prueba funcionales también se pueden utilizar para pruebas de confiabilidad. Veamos eso a continuación.

Fiabilidad

La confiabilidad del software puede denominarse como la capacidad de un sistema para realizar y mantener sus funciones en circunstancias rutinarias así como en situaciones inesperadas. circunstancias. El sistema también debe realizar y mantener sus funciones. con consistencia y repetibilidad. El análisis de confiabilidad responde a la pregunta, "¿Cuánto tiempo funcionará antes de que se rompa?" Algunas estadísticas utilizadas para medir la confiabilidad son:

Tiempo medio hasta la falla: El tiempo promedio o medio entre la operación inicial y la primera aparición de una falla o mal funcionamiento. En otras palabras, ¿cuánto tiempo puede funcionar el sistema antes de fallar la primera vez?

Tiempo medio entre fallas: una medida estadística de confiabilidad, que se calcula para indicar el tiempo promedio anticipado entre fallas.

Cuanto más tiempo mejor.

En proyectos tradicionales, solíamos programar semanas de pruebas de confiabilidad que Intenté ejecutar simulaciones que coincidieran con el trabajo de un día normal. Ahora, deberíamos ser capaz de entregar al final de cada iteración, entonces, ¿cómo podemos programar pruebas de confiabilidad?

Contamos con pruebas unitarias y de aceptación automatizadas que se ejecutan de forma regular. A hacer una prueba de confiabilidad, simplemente necesitamos usar esas mismas pruebas y ejecutarlas y más. Lo ideal sería utilizar estadísticas recopiladas que muestren el uso diario, crear un script que refleje el uso y ejecutarlo en una versión estable para cualquier ocasión.

el tiempo que su equipo considere adecuado para demostrar estabilidad. Puedes ingresar al azar datos en las pruebas para simular el uso de producción y asegurarse de que la aplicación no falla debido a entradas no válidas. Por supuesto, es posible que desees reflejar uso pico para asegurarse de que también maneje las horas punta.

Puedes crear historias en cada iteración para desarrollar estos guiones y agregar nuevos funcionalidad a medida que se agrega a la aplicación. Tus pruebas de aceptación podrían ser muy específico como, "La funcionalidad X debe realizar 10,000 operaciones en un Período de 24 horas por un mínimo de 3 días."

Cuidado: ejecutar mil pruebas sin problemas graves no significa que tiene un software confiable. Tienes que realizar las pruebas correctas. Hacer un prueba de confiabilidad efectiva, piense en su aplicación y cómo se usa en todos día, todos los días, durante un período de tiempo. Especifique pruebas que tengan como objetivo demostrar que su aplicación podrá satisfacer las necesidades de sus clientes. incluso durante las horas pico.

Pregunte al equipo del cliente por sus criterios de fiabilidad en forma de valores medibles. objetivos. Por ejemplo, podrían considerar confiable el sistema si ocurren diez o menos errores por cada 10,000 transacciones, o si la aplicación web está disponible. 99,999% del tiempo. La recuperación de cortes de energía y otros desastres podría serán parte de los objetivos de confiabilidad, y se indicarán en el formulario de Servicio Acuerdos de Nivel. Sepa cuáles son. Algunas industrias tienen sus propios estándares y pautas de confiabilidad del software.

Impulsar el desarrollo con el programador adecuado y las pruebas de clientes deberían mejorar la confiabilidad de la aplicación, porque esto generalmente conduce a un mejor diseño y menos defectos. Escriba historias y tareas adicionales según sea necesario para entregar un sistema que cumpla con los estándares de confiabilidad de la organización.

Su producto puede ser confiable una vez que esté en funcionamiento, pero también debe ser instalable por todos los usuarios, en todos los entornos compatibles. Esta es otra área donde seguir principios ágiles nos da una ventaja.

Instalabilidad

Una de las piedras angulares de un equipo ágil exitoso es la integración continua. Esto significa que una compilación está lista para probarse en cualquier momento del día. Muchos

Los equipos eligen implementar una o más de las compilaciones exitosas en entornos de prueba diariamente.

Automatizar la implementación crea repetibilidad y hace que la implementación sea un ningún evento. Esto es emocionante para nosotros porque hemos experimentado semanas de intentar para integrar e instalar un nuevo sistema. Sabemos que si compilamos una vez e implementamos la misma compilación en múltiples entornos, habremos desarrollado coherencia.

La historia de Janet

En un proyecto en el que trabajé, la implementación fue automática y se probó en múltiples entornos durante el ciclo de desarrollo. Sin embargo, hubo problemas durante la implementación en el sitio del cliente. Agregamos un paso al final del juego para que el grupo de soporte tomara el lanzamiento y hiciera una prueba de instalación completa como si fuera el sitio del cliente. Pudimos revisar las notas de implementación y eliminamos muchos de los problemas que el cliente habría visto de otro modo.

—Janet

Capítulo 20, "Entrega exitosa"
tiene más información sobre pruebas de instalación.

Al igual que con cualquier otra funcionalidad, los riesgos asociados con la instalación deben ser evaluado y la cantidad de pruebas determinada en consecuencia. Nuestro consejo es hágalo temprano y con frecuencia, y automatice el proceso si es posible.

Resumen de “ilidad”

Hay otras “ilidades” para probar, dependiendo del dominio de su producto. El software crítico para la seguridad, como el utilizado en dispositivos médicos y sistemas de control de aeronaves, requiere pruebas de seguridad exhaustivas, y las pruebas de regresión probablemente contendría pruebas relacionadas con la seguridad. Redundancia del sistema y pruebas de comutación por error Sería especialmente importante para un producto de este tipo. Es posible que su equipo necesite Mire los datos de la industria sobre problemas de seguridad relacionados con el software y use código adicional. revisiones. Configurabilidad, auditabilidad, portabilidad, solidez y extensibilidad son solo algunas de las cualidades que su equipo podría necesitar evaluar con pruebas tecnológicas.

Cualquiera que sea la “capacidad” que necesita probar, utilice un enfoque incremental. Comience por obtener los requisitos del equipo del cliente y ejemplos de sus objetivos para esa área particular de calidad. Escriba pruebas orientadas al negocio para asegurarse de que El código está diseñado para cumplir esos objetivos. En la primera iteración, el equipo podría hacer Investigar un poco y idear una estrategia de prueba para evaluar el nivel de calidad existente del producto. El siguiente paso podría ser crear un entorno de prueba adecuado, investigar herramientas o comenzar con algunas pruebas manuales.

A medida que aprenda cómo la aplicación está a la altura de los requisitos de los clientes, cierre el círculo con nuevas pruebas de los Cuadrantes 1 y 2 que acercan la aplicación a los objetivos de esa propiedad en particular. Un enfoque incremental

También se recomienda para pruebas de rendimiento, carga y otras pruebas que se abordan en la siguiente sección.

RENDIMIENTO, CARGA, ESTRÉS, Y PRUEBAS DE ESCALABILIDAD

Las pruebas de rendimiento, carga, estrés y escalabilidad se incluyen en el cuadrante 4 debido a su enfoque tecnológico. A menudo se requieren habilidades especializadas, aunque muchos equipos han descubierto formas de realizar sus propias pruebas en estos árees. Hablemos primero de escalabilidad, porque a menudo se olvida.

Escalabilidad

Las pruebas de escalabilidad verifican que la aplicación sigue siendo confiable cuando hay más usuarios. se agregan. Lo que eso realmente significa es: "¿Puede su sistema manejar la capacidad de ¿Una base de clientes en crecimiento? Suena simple, pero en realidad no lo es y es un problema. que un equipo ágil normalmente no puede resolver por sí solo.

Es importante pensar en todo el sistema y no sólo en la aplicación. sí mismo. Por ejemplo, la red suele ser el cuello de botella, porque no puede soportar el aumento del rendimiento. ¿Qué pasa con la base de datos? ¿Se ampliará? Será el ¿El hardware que está utilizando soporta las nuevas cargas que se están considerando? ¿Es simple? ¿Solo para agregar nuevo hardware o es el cuello de botella?

La historia de Janet

En una organización en la que trabajé recientemente, su base de clientes había crecido muy rápidamente y la solución en la que habían invertido había alcanzado su capacidad debido a limitaciones de hardware. No fue una simple cuestión de agregar un nuevo servidor, porque la solución no fue diseñada de esa manera. Era necesario monitorear el sistema para reiniciar los servicios durante el uso pico.

Para crecer, la organización tuvo que cambiar las soluciones para adaptarse a su crecimiento futuro, pero esto no se reconoció hasta que empezaron a surgir problemas.

Lo ideal sería que la organización hubiera reemplazado el antiguo sistema antes de que fuera un problema. Este es un ejemplo de por qué es importante comprender su sistema y su capacidad, así como las proyecciones de crecimiento futuro.

—Janet

Deberá salir del equipo para obtener las respuestas que necesita abordar.

problemas de escalabilidad, así que planifique con anticipación.

Pruebas de rendimiento y carga

Las pruebas de rendimiento generalmente se realizan para ayudar a identificar cuellos de botella en un sistema. o para establecer una línea de base para pruebas futuras. También se hace para garantizar el cumplimiento de los objetivos y requisitos de desempeño y para ayudar a las partes interesadas. tomar decisiones informadas relacionadas con la calidad general de la aplicación que se está probando.

Las pruebas de carga evalúan el comportamiento del sistema a medida que más y más usuarios acceden al sistema al mismo tiempo. Las pruebas de estrés evalúan la solidez de la aplicación bajo cargas superiores a las esperadas. ¿La aplicación se ampliará a medida que

¿El negocio crece? Características como el tiempo de respuesta pueden ser más críticas que la funcionalidad para algunas aplicaciones.

Grig Gheorghiu [2005] enfatiza la necesidad de expectativas claramente definidas para obtener valor de las pruebas de rendimiento. Él dice: "Si no sabes dónde estás quieras ir en términos del sistema, entonces importa poco en qué dirección tomar (¿recuerdas a Alicia y el gato de Cheshire?)". Por ejemplo, probablemente Quiero saber el número de usuarios simultáneos y la respuesta aceptable. tiempo para una aplicación web.

Herramientas de prueba de carga y rendimiento



Consulte la bibliografía para obtener enlaces a sitios donde puede buscar herramientas

Una vez que haya definido sus objetivos de rendimiento, puede utilizar una variedad de herramientas para Cargue el sistema y compruebe si hay cuellos de botella. Esto se puede hacer en el nivel de unidad, con herramientas como JUnitPerf, httpperf o un arnés propio.

Apache JMeter, The Grinder, Pounder, fptt y OpenWebLoad son más

ejemplos de las muchas herramientas de prueba de carga y rendimiento de código abierto disponibles al momento de escribir este artículo. Algunos de ellos, como JMeter, se pueden utilizar en un variedad de tipos de servidores, desde SOAP hasta LDAP y correo POP3. También hay disponibles muchas opciones de herramientas comerciales, incluidas NeoLoad, WebLoad, eValid, LoadTest, LoadRunner y SOATest.

Utilice estas herramientas para buscar cuellos de botella en el rendimiento. El equipo de Lisa usa JProfiler para buscar cuellos de botella en las aplicaciones y pérdidas de memoria, y JConsole para analizar uso de la base de datos. Existen herramientas similares para .NET y otros entornos, incluido .NET Memory Profiler y ANTS Profiler Pro. Como señala Grig, hay perfiladores específicos de bases de datos para identificar problemas de rendimiento a nivel de base de datos; Pídale a sus expertos en bases de datos que trabajen con usted. Los administradores de su sistema pueden

ayudarle a utilizar comandos de shell como top o herramientas como PerfMon para monitorear CPU, memoria, intercambio, E/S de disco y otros recursos de hardware. Herramientas similares son disponibles a nivel de red, por ejemplo, NetScout.

También puede utilizar las herramientas con las que el equipo esté más familiarizado. En un proyecto, Janet trabajó muy de cerca con uno de los programadores para crear las pruebas. Ella lo ayudó a definir las pruebas necesarias en función del desempeño del cliente. y cargar expectativas, y las automatizó usando JUnit. juntos ellos analizó los resultados para informar al cliente.

Establecer una línea de base es un buen primer paso para evaluar el desempeño. El siguiente sección explora este aspecto de las pruebas de rendimiento.

Base

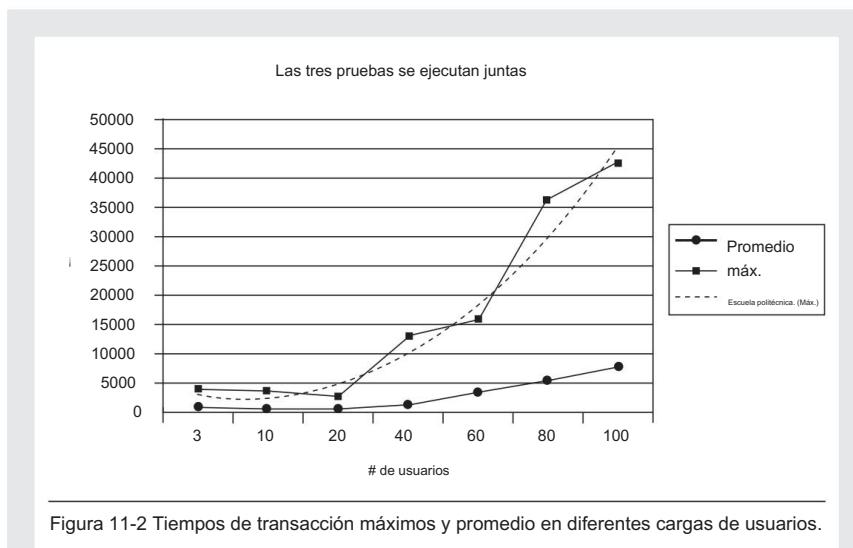
El ajuste del rendimiento puede convertirse en un gran proyecto, por lo que es esencial proporcionar una línea de base que puede comparar con nuevas versiones del software en cuanto a rendimiento. Incluso si el rendimiento no es su mayor preocupación en este momento, no ignorarlo. Es una buena idea obtener una línea base de desempeño para saberlo más tarde. en qué dirección se dirige su tiempo de respuesta. La empresa de Lisa alberga un sitio web. que ha tenido una pequeña carga. Obtuvieron una línea base de prueba de carga en el sitio para que a medida que creciera, sabrían cómo se estaba viendo afectado el rendimiento.

Resultados de las pruebas de referencia de rendimiento

Mike Busse, compañero de trabajo de Lisa, asumió la tarea de obtener líneas base de rendimiento para su aplicación web que gestiona planes de jubilación. Evaluó herramientas de prueba de carga, implementó una (JMeter) y se dispuso a obtener una línea de base. Informó los resultados tanto en un resumen de alto nivel como en una hoja de cálculo con resultados detallados.

Las pruebas simularon aumentar lentamente la carga hasta 100 usuarios simultáneos. Se utilizaron tres scripts de prueba, cada uno para una actividad de usuario común, y se ejecutaron por separado y todos juntos. Los datos recopilados incluyeron:

- Tiempo máximo de una transacción
- Número máximo de conexiones ocupadas.
- Un gráfico del tiempo máximo de una transacción frente al número de usuarios (ver Figura 11-2 para un ejemplo de un gráfico)
- Número de usuarios que estaban en el sistema cuando se alcanzó el tiempo máximo de una transacción fue igual a ocho segundos



Un aspecto importante de la presentación de informes de resultados fue proporcionar definiciones de términos como **transacción** y **tiempo máximo**, que los resultados sean significativos para todos. Por ejemplo, el tiempo máximo de una transacción se define como la transacción más larga de todas las transacciones completadas durante la prueba.

El informe de Mike también incluyó suposiciones hechas para la prueba de desempeño:

- Ocho segundos es un umbral de transacción que no nos gustaría cruzar.
- El servidor web de prueba es equivalente a cualquiera de los dos servidores web en producción.
- La carga que el sistema puede manejar, según lo determinado por estas pruebas, se puede duplicar en producción porque la carga se distribuye entre dos servidores web.
- La distribución de tareas en la prueba que combina las tres pruebas es precisa. tasa en un grado razonable.

Mike también identificó deficiencias en la línea base de desempeño. Más de una transacción puede contribuir a cargar una página, lo que significa que el tiempo máximo de carga de la página podría ser mayor que el tiempo máximo de una transacción. La máquina de prueba no duplica el entorno de producción, que cuenta con dos máquinas y software de equilibrio de carga para distribuir las transacciones.

El informe finalizó con una conclusión sobre la cantidad de usuarios simultáneos que podría admitir el sistema de producción. Esto sirve como una guía a tener en cuenta a medida que aumenta la carga de producción. La carga actual es menos de la mitad de este número, pero hay incógnitas, como si todos los usuarios de producción están activos o no han cerrado sesión.

Asegúrese de que sus pruebas de rendimiento imiten adecuadamente las condiciones de producción. Haga que los resultados sean significativos definiendo cada prueba y métrica, explicando cómo se correlacionan los resultados con el entorno de producción y qué se puede hacer con los resultados, y proporcionando los resultados en forma gráfica.

Si existen criterios de desempeño específicos que se han definido para funcionalidad, sugerimos que se realicen pruebas de rendimiento como parte de la iteración para garantizar que se encuentren los problemas antes de que sea demasiado tarde para solucionarlos.

La evaluación comparativa se puede realizar en cualquier momento durante un lanzamiento. Si nueva funcionalidad Se agrega algo que podría afectar el rendimiento, como consultas complicadas, vuelve a ejecutar las pruebas para asegurarse de que no haya efectos adversos. De esta manera, tienes tiempo para optimizar la consulta o el código al principio del ciclo cuando el desarrollo El equipo todavía está familiarizado con la función.

Cualquier prueba de rendimiento, carga o estrés no será significativa a menos que se ejecute en un entorno que imita el entorno de producción. hablemos mas sobre ambientes.

Entornos de prueba

Las ejecuciones finales de las pruebas de rendimiento ayudarán a los clientes a tomar decisiones sobre aceptando su producto. Para obtener resultados precisos, las pruebas deben realizarse en equipos similares al de producción. A menudo los equipos utilizan máquinas más pequeñas y extrapolan los resultados para decidir si el rendimiento es suficiente para el negocio necesita. Esto debe señalarse claramente al informar los resultados de las pruebas.

También se puede hacer hincapié en la aplicación para ver qué carga puede soportar antes de que falle. Se realiza en cualquier momento durante el lanzamiento, pero generalmente no se considera de alta prioridad. por los clientes a menos que tenga un sistema de misión crítica con mucha carga.

Un recurso que se ve afectado por el aumento de la carga es la memoria. En la siguiente sección, analizamos la gestión de la memoria.

Gestión de la memoria

La memoria generalmente se describe en términos de la cantidad (normalmente la mínima o la máxima) de memoria que se utilizará para RAM, ROM, discos duros y pronto. Debe tener en cuenta el uso de la memoria y estar atento a las fugas, porque pueden causar fallas catastróficas cuando la aplicación está en producción durante el uso pico. Algunos lenguajes de programación son más susceptibles a Problemas de memoria, por lo que comprender las fortalezas y debilidades del código.

le ayudará a saber a qué prestar atención. Las pruebas de problemas de memoria pueden realizarse como parte de las pruebas de rendimiento, carga y estrés.

La recolección de basura es una herramienta que se utiliza para liberar memoria al programa. Sin embargo, puede enmascarar problemas graves de memoria. Si ves la memoria disponible disminuyendo constantemente con el uso y luego, de repente, aumentando hasta el máximo disponible, podría sospechar que la recolección de basura se ha activado. detectar anomalías en el patrón o si el sistema comienza a ralentizarse en condiciones uso intensivo. Es posible que necesite monitorear por un tiempo y trabajar con los programadores para encontrar el problema. La solución podría ser algo simple, como programar la recolección de basura con mayor frecuencia o establecer un nivel de activación más alto.

Cuando esté trabajando con los programadores en una historia, pregúntales si esperan problemas con la memoria. Puedes probar específicamente si sabes que hay Podría ser un riesgo en la zona. No siempre es fácil detectar pérdidas de memoria, pero hay herramientas para ayudar. Esta es un área donde los programadores deberían tener herramientas. fácilmente disponible. Colabora con ellos para verificar que la aplicación esté libre de Problemas de memoria. Realice las pruebas de rendimiento y carga descritas en la sección anterior para verificar que no haya problemas de memoria.

No es necesario ser un experto en cómo realizar pruebas tecnológicas que critica el producto para ayudar a su equipo a planificarlo y ejecutarlo. Tu equipo puede evaluar qué pruebas necesita de este cuadrante. Hable sobre estas pruebas como planeas tu liberación; Puedes crear un plan de prueba específicamente para el rendimiento. y cárgalo si no lo has hecho antes. Necesitará tiempo para obtener la experiencia necesaria, ya sea adquiriéndola identificando y aprendiendo las habilidades, o trayendo ayuda externa. Al igual que con todos los esfuerzos de desarrollo, divida las pruebas tecnológicas en pequeñas tareas que puedan abordarse y desarrollarse. cada iteración.

RESUMEN

En este capítulo, hemos explorado el cuarto cuadrante de pruebas ágiles, las pruebas tecnológicas que critican el producto.

El equipo de desarrollo debe evaluar si tiene o puede adquirir la experiencia para realizar estas pruebas, o si necesita planificar la incorporación de personal externo. recursos.

Un enfoque incremental para estas pruebas, completando tareas en cada iteración, garantiza tiempo para abordar cualquier problema que surja y evitar problemas de producción.

El equipo debe considerar varios tipos de pruebas de “líedad”, incluidas pruebas de seguridad, mantenibilidad, interoperabilidad, compatibilidad, confiabilidad e instalabilidad, y debe ejecutar estas pruebas en los momentos apropiados.

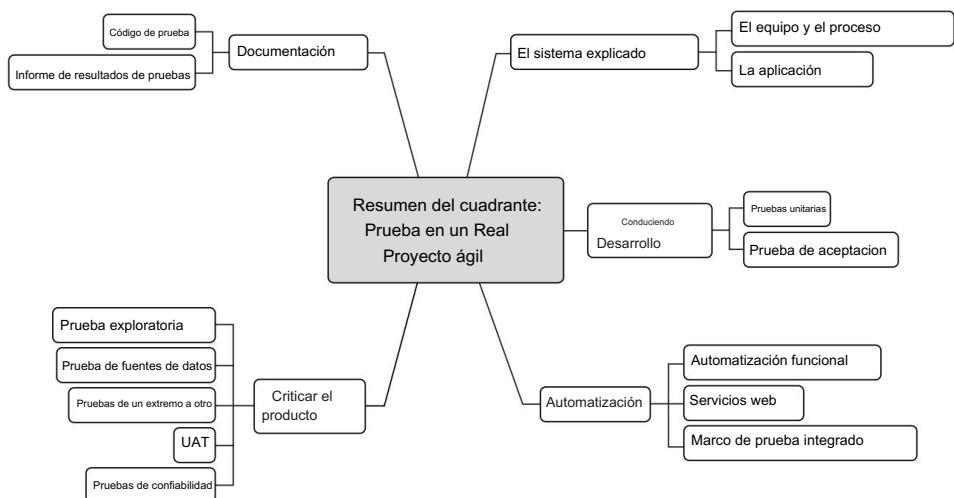
Las pruebas de rendimiento, escalabilidad, estrés y carga deben realizarse desde el inicio del proyecto.

Investigue los problemas de administración de memoria que podrían afectar su producto y planifique pruebas para verificar que la aplicación no tenga problemas de memoria.

Esta página se dejó en blanco intencionalmente.

Capítulo 12

RESUMEN DE CUADRANTES DE PRUEBA



En el Capítulo 6, presentamos los cuadrantes de prueba y en los capítulos siguientes hablamos sobre cómo utilizar los conceptos en su proyecto ágil. En este capítulo, lo reuniremos todo con un ejemplo de un equipo ágil que utilizó pruebas de los cuatro cuadrantes.

REVISIÓN DE LOS CUADRANTES DE PRUEBAS

Acabamos de dedicar cinco capítulos a hablar sobre cada uno de los cuadrantes (consulte la Figura 12-1) y ejemplos de herramientas que puede utilizar para los diferentes tipos de pruebas. El siguiente truco es saber qué pruebas necesita su proyecto y cuándo realizarlas. En este capítulo, lo guiaremos a través de un ejemplo de la vida real de un proyecto ágil que utilizó pruebas de los cuatro cuadrantes de pruebas ágiles.

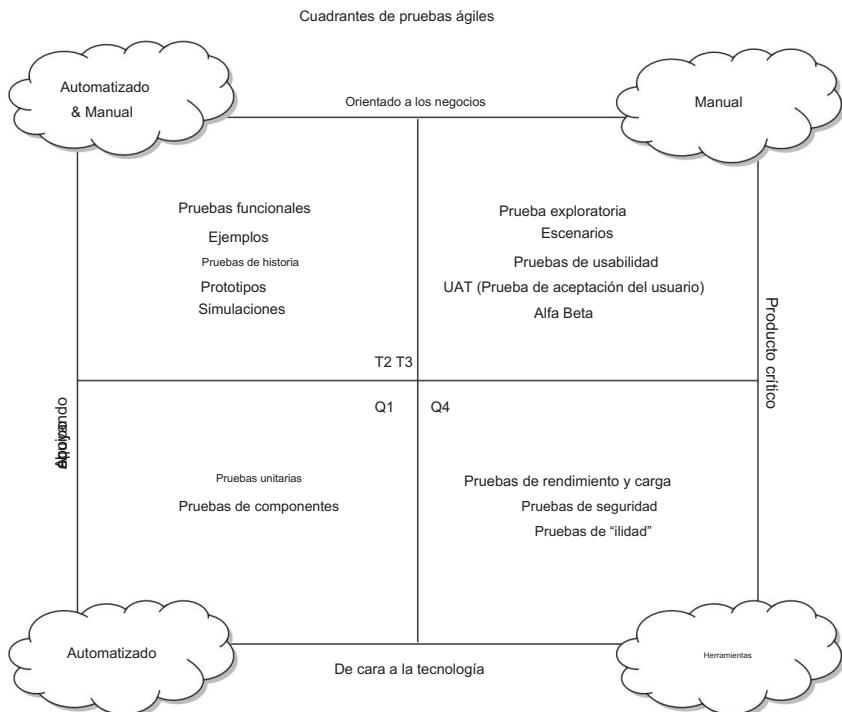


Figura 12-1 Cuadrantes de pruebas ágiles

UN EJEMPLO DE PRUEBA DEL SISTEMA

La siguiente historia trata sobre el éxito de una organización al probar todo su sistema utilizando una variedad de herramientas locales y de código abierto. Janet trabajó con este equipo y Paul Rogers fue el arquitecto de pruebas principal. Esta es la historia de Pablo.

La aplicación El sistema

resuelve el problema de monitorear pozos remotos de producción de petróleo y gas. La solución combina un dispositivo de monitoreo remoto que puede transmitir datos y recibir ajustes desde una estación central de monitoreo usando un protocolo propietario a través de un canal de comunicación satelital.

La Figura 12-2 muestra la arquitectura del sistema de monitoreo remoto de datos. Los dispositivos de medición en los pozos petroleros, Unidades Terminales Remotas (RTU), utilizan un va-

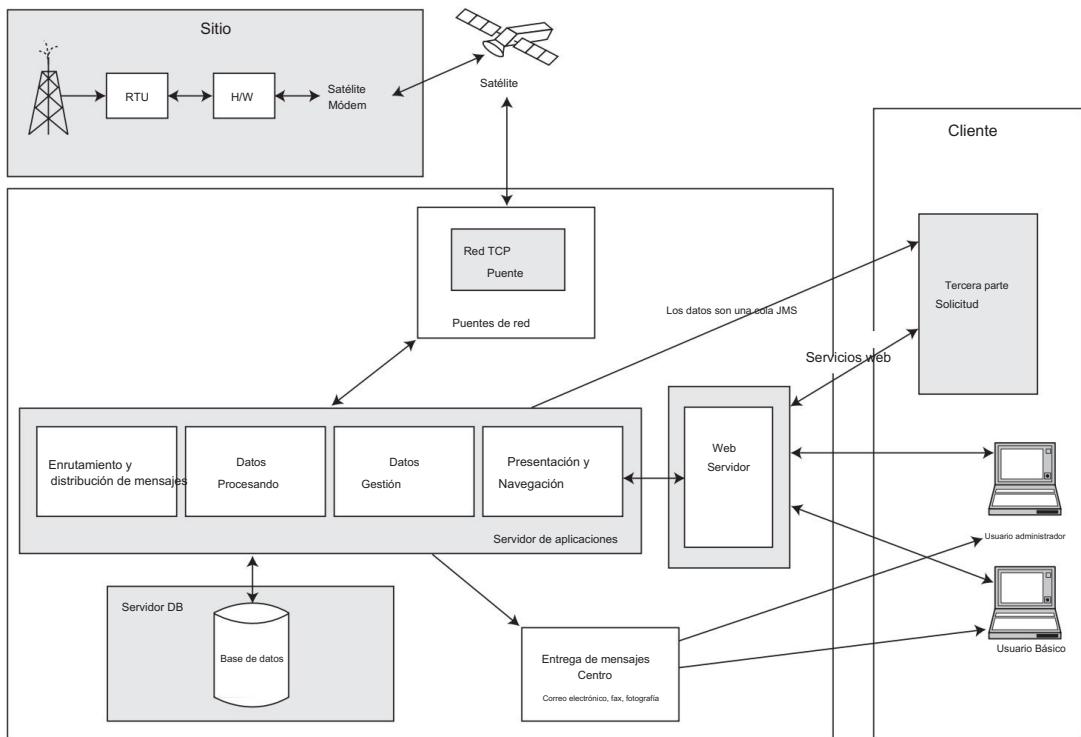


Figura 12-2 Arquitectura del sistema de monitoreo remoto de datos

Variedad de protocolos para comunicarse con el dispositivo de medición. Estos datos de cada RTU se transmiten vía satélite a servidores ubicados en la oficina principal del cliente. Luego se pone a disposición de los usuarios a través de una interfaz web. Un sistema de notificación, por correo electrónico, fax o teléfono, está disponible cuando una lectura particular está fuera de los límites operativos normales. También están disponibles un feed Java Message Service (JMS) y servicios web para ayudar a la integración con otras aplicaciones de los clientes.

La aplicación de software era un enorme sistema heredado que tenía pocas pruebas unitarias. El equipo estaba reconstruyendo lentamente la aplicación con nueva tecnología.

El equipo y el proceso

El equipo estaba formado por cuatro programadores de software, dos programadores de firmware, tres o cuatro evaluadores, un ingeniero de producto y un gerente externo. El cliente "real" estaba en otro país. El equipo de desarrollo utiliza XP.

prácticas, incluida la programación en pares y TDD. El equipo del cliente utilizó el sistema de seguimiento de defectos para el trabajo pendiente, pero la mayor parte de la visibilidad del historias fue a través de fichas. Se utilizaron tarjetas de historia durante la iteración. reuniones de planificación y el tablero de tareas siguió el progreso.

Scrum se utilizó como mecanismo externo de presentación de informes a la organización y los clientes. El equipo tuvo iteraciones de dos semanas y lanzó el producto. aproximadamente cada cuatro meses. Esto variaba dependiendo de la funcionalidad que se estaba desarrollado. Se llevaron a cabo retrospectivas como parte de cada sesión de planificación de iteraciones y se tomaron medidas sobre los tres elementos prioritarios discutidos.

La integración continua a través de CruiseControl proporcionó compilaciones constantes para los testers y las demostraciones realizadas al final de cada iteración. Cada evaluador tenía un entorno local para probar la aplicación web, pero había tres entornos de prueba disponibles para el sistema. El primero fue probar nuevos historias y se actualizó según fue necesario con la última versión. El segundo fue para probar problemas informados por el cliente, porque tenía la última versión lanzada para los clientes. El tercer entorno fue un entorno de prueba completamente independiente. que estaba disponible para probar implementaciones completas, enlaces de comunicación y el firmware y hardware. Fue en este entorno donde ejecutamos nuestra carga y pruebas de confiabilidad.

PRUEBAS QUE IMPULSAN EL DESARROLLO

Las pruebas que impulsaron el desarrollo incluyeron pruebas unitarias y pruebas de aceptación.

Pruebas unitarias

El Capítulo 7, "Pruebas tecnológicas que respaldan al equipo", explica más sobre las pruebas unitarias y TDD.

Las pruebas unitarias son pruebas tecnológicas que respaldan la programación. Los que se desarrollan como parte del desarrollo basado en pruebas no solo ayudan al programador a entender la historia correctamente sino que también ayudan a diseñar el sistema.

Los programadores del proyecto Remote Data Monitoring participaron en Test Desarrollo impulsado (TDD) y programación en pareja de todo corazón. Todo nuevo La funcionalidad fue desarrollada y probada usando programación de pares. Todas las historias entregados a los evaluadores fueron respaldados por pruebas unitarias y se detectaron muy pocos errores encontrado después de completar la codificación. Los errores que se encontraron fueron generalmente relacionados con la integración.

Sin embargo, cuando el equipo comenzó, el sistema heredado tenía pocas pruebas unitarias para apoyar la refactorización. A medida que se implementaron cambios en el proceso, los desarrolladores

decidió empezar a solucionar el problema. Cada vez que tocaron un fragmento de código en el sistema heredado, agregaron pruebas unitarias y refactorizaron el código según fuera necesario. Gradualmente, el sistema heredado se volvió más estable y pudo resistir refactorizaciones importantes cuando fue necesario. Experimentamos el poder de pruebas unitarias!

Prueba de aceptacion

El ingeniero de producto (el representante del cliente) se hizo cargo de la creación del prueba de aceptacion. Estas pruebas variaron en formato dependiendo de la historia real. Aunque tuvo dificultades al principio, el ingeniero de producto se volvió bastante bueno dando las pruebas a los programadores antes de que comenzaran a codificar. El equipo creó un plantilla de prueba, que evolucionó con el tiempo, que cumplió con los requisitos tanto de los programadores como de las necesidades de los probadores.

A veces las pruebas se escribían de manera informal, pero incluían datos, una configuración requerida si no era inmediatamente obvia, diferentes variaciones que eran críticas a la historia y algunos ejemplos. El equipo descubrió que los ejemplos ayudaron aclarar las expectativas de muchas de las historias.

El equipo de prueba automatizó las pruebas de aceptación lo antes posible, generalmente en el mismo tiempo que se desarrollaban las historias. Por supuesto, el ingeniero de producto estuvo disponible para responder cualquier pregunta que surgiera durante el desarrollo.

Consulte el Capítulo 8,
"Presentación empresarial
Pruebas que respaldan al
equipo", para obtener
más información sobre
cómo impulsar el desarrollo
con pruebas de aceptación.

Estas pruebas de aceptación tuvieron tres propósitos. Eran pruebas de cara al negocio que apoyaron el desarrollo porque fueron entregados al equipo antes de codificar comenzó. En segundo lugar, el equipo de pruebas los utilizó como base para la automatización. eso alimentó la suite de regresión y proporcionó ideas futuras para pruebas exploratorias. El tercer propósito fue confirmar que la implementación cubrió las necesidades del cliente. El ingeniero de producto realizó esta verificación de la solución.

AUTOMATIZACIÓN

La automatización involucró la estructura de prueba funcional, los servicios web y las pruebas integradas.

La estructura de prueba funcional automatizada

Ruby se utilizó con Watir como herramienta elegida para la automatización funcional. estructura. Se decidió tener la mayor flexibilidad y oportunidad. para la personalización que se requería para el sistema bajo prueba.

El código de prueba automatizado incluía tres capas distintas, como se muestra en la Figura 12-3.

La capa más baja, la Capa 1, incluía a Watir y otras clases, como los madereros.
que escribió en los archivos de registro.

La segunda capa, Capa 2, era la capa de acceso a la página, donde las clases que contenían código para acceder a páginas web individuales vividas. Por ejemplo, en la solicitud bajo prueba (AUT) había una página de inicio de sesión, una página para crear usuario y una página para editar usuario. Las clases escritas en Ruby contenían código que podía realizar ciertas funciones en la AUT, como una clase que inicia sesión en la aplicación, una clase para editar un usuario y una clase para asignar derechos de acceso a un usuario. Estas clases no contenían datos. Por ejemplo, la clase de inicio de sesión no sabía con qué nombre de usuario iniciar sesión.

La tercera y superior capa, la Capa 3, era la capa de prueba y contenía los datos necesario para realizar una prueba. Llamó clases de Capa 2, que a su vez llamaron Capa 1.

Por ejemplo, la prueba real llamaría a LogIn y pasaría a Janet como nombre de usuario.
y Passw0rd como contraseña. Esto significaba que podía alimentarse de muchas maneras diferentes.
conjuntos de datos fácilmente.

Iniciar sesión ('Janet', 'Contraseña')

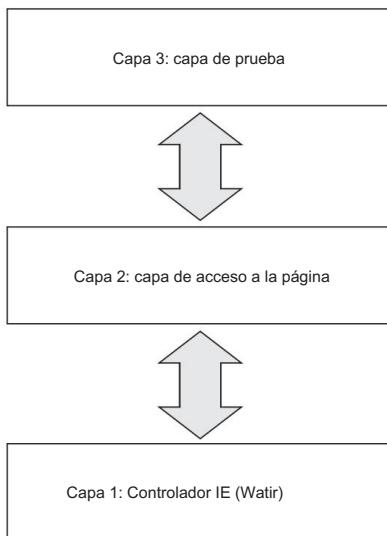


Figura 12-3 Capas de prueba funcional

La capa 2 también sabía cómo manejar los mensajes de error que generaba la aplicación. Por ejemplo, cuando se ingresó un nombre de usuario no válido en la página de inicio de sesión, la clase de inicio de sesión detectó el mensaje de error y luego devolvió el problema a las pruebas en la Capa 3.

Esto significa que se podrían usar las mismas clases de Capa 2 para ambas pruebas de camino feliz. y para pruebas negativas. En el caso negativo, la Capa 3 esperaría que la Capa 2 devolvería un error y luego verificaría si la prueba falló por el motivo correcto accediendo a los mensajes de error que la Capa Dos eliminó del navegador.

Las pruebas funcionales utilizaron Ruby con Watir para controlar el DOM en el navegador y podía acceder a casi todos los objetos de la página. el automatizado El conjunto de pruebas se ejecutó en compilaciones nocturnas para brindarle al equipo comentarios consistentes sobre Comportamiento de aplicaciones de alto nivel. Esto fue un salvavidas ya que el equipo continuó construir las pruebas unitarias. Esta arquitectura acomodó de manera eficiente las pruebas empresariales que respaldan al equipo.

Servicios web

Los clientes utilizaban los servicios web para interactuar con algunas de sus otras aplicaciones. El grupo de desarrollo utilizó Ruby para escribir un cliente para probar cada servicio. Ellos Desarrollaron. Para estas pruebas, el marco de pruebas unitarias de Ruby, Test::Unit, fue usado.

El equipo de pruebas amplió las pruebas de servicios web para cubrir más de 1000 casos de prueba diferentes y solo tardó unos minutos en ejecutarse. le dieron al equipo una cantidad asombrosa de cobertura en un corto período de tiempo.

El equipo demostró el cliente de prueba a los clientes, quienes decidieron utilizar eso también. Sin embargo, los clientes decidieron posteriormente que no funcionaba ellos, por lo que comenzaron a escribir sus propias pruebas, aunque de una manera mucho más ad hoc moda usando Ruby.

Usaron IRB, la interfaz interactiva proporcionada por Ruby, y alimentaron valores en un método exploratorio. Le dio al cliente un entorno interactivo para descubrir qué funcionó y qué no. También les permitió familiarizarse con Ruby y cómo estábamos probando, y les dio mucha más confianza en nuestras pruebas. Gran parte de sus pruebas de aceptación del usuario se realizaron mediante IRB.

Tres enfoques diferentes en las pruebas de servicios web sirvieron para tres propósitos diferentes. Los programadores lo utilizaron para ayudar a probar a su cliente e impulsar su desarrollo. Los evaluadores lo utilizaron para criticar el producto de una manera automatizada muy eficiente.

manera, y los clientes pudieron probar los servicios web entregados a ellos usando IRB.

Pruebas integradas

Además de la interfaz web, el sistema RDM constaba de un pequeño dispositivo integrado que se comunicaba con el equipo de medición mediante varios protocolos. Utilizando Ruby se desarrollaron varias pruebas para probar parte de su interfaz administrativa. Esta interfaz era un sistema de línea de comandos similar a FTP.

Estas pruebas basadas en datos estaban contenidas en una hoja de cálculo de Excel. Un rubí El script leería comandos de Excel usando la interfaz OLE y enviaría al dispositivo integrado. El guión luego compararía la respuesta. desde el dispositivo con el resultado esperado, también guardado en la hoja de cálculo. Errores fueron resaltados en rojo. Estas pruebas automatizadas tardaron aproximadamente una hora. ejecutar, mientras que realizar las mismas pruebas manualmente llevaría ocho horas.

Si bien esto proporcionó mucha cobertura de prueba, en realidad no probó la razón por la que Se utilizó un dispositivo que era para leer datos de las RTU. Se escribió un simulador en Ruby con una GUI de FOX (FXRuby). Esto permitió introducir datos simulados en el dispositivo. Debido a que el simulador podía controlarse de forma remota, se incorporó a pruebas automatizadas que ejercieron la capacidad del dispositivo integrado para leer datos, responder a condiciones de error y generar alarmas cuando la entrada los datos excedieron un umbral predeterminado.

Las pruebas integradas son muy técnicas, pero con el poder proporcionado por el simulador, todo el equipo pudo participar en las pruebas del dispositivo. El simulador fue escrito para respaldar las pruebas para el equipo de pruebas, pero el programador del firmware lo encontró valioso y lo utilizó también para ayudar con sus esfuerzos de desarrollo. Ese fue un efecto secundario positivo e inesperado. Cuadrante 2 Las pruebas que apoyan al equipo pueden incorporar una variedad de tecnologías, ya que hizo en este proyecto.

CRITICAR EL PRODUCTO CON PRUEBAS DE CARA AL NEGOCIO

Las pruebas comerciales que critican el producto se describen en esta sección.

Prueba exploratoria

Las pruebas automatizadas fueron simples y fáciles de usar para todos los miembros del equipo. Se podrían ejecutar scripts de prueba individuales para configurar condiciones específicas, permitiendo



Pruebas exploratorias,
pruebas de usabilidad
y otras
Las pruebas del
cuadrante 3 se analizan en
Capítulo 10,
"De cara al negocio
Pruebas que critican
el producto."

Se pueden realizar pruebas exploratorias efectivas sin tener que dedicar mucho tiempo.
ingresar datos manualmente. Esto funcionó para los tres marcos de prueba: funcional, servicios web e integrado.

El equipo realizó pruebas exploratorias para complementar la prueba automatizada.
suites y obtenga la mejor cobertura posible. Esta interacción humana con el
El sistema encontró problemas que la automatización no encontró.

Las pruebas de usabilidad no eran un requisito crítico para el sistema, pero los evaluadores
Miré para que la interfaz tuviera sentido y fluyera sin problemas. los probadores
utilizó ampliamente pruebas exploratorias para criticar el producto. El ingeniero de producto también utilizó
pruebas exploratorias para sus pruebas de verificación de soluciones.

Prueba de fuentes de datos

Como se muestra en la Figura 12-2, los datos del sistema están disponibles en un JMS
cola, así como el navegador web. Para probar la cola JMS, el desarrollo
El grupo escribió un proxy Java. Se conectó a una cola e imprimió cualquier llegada
datos a la consola. También escribieron un cliente Ruby que recibió estos datos a través de un
pipe, que luego estaba disponible en el sistema de prueba automatizado Ruby.

Los correos electrónicos se enviaron automáticamente cuando se encontraron condiciones de alarma.
Los correos electrónicos de alarma contenían tanto correos electrónicos de texto sin formato como correos
electrónicos con archivos adjuntos. Los archivos adjuntos MIME contenían datos útiles para realizar pruebas, por lo que Ruby
Se escribió un cliente de correo electrónico que admitía archivos adjuntos.

Las pruebas de un extremo a otro

El cuadrante 3 incluye pruebas funcionales de extremo a extremo que demuestran el comportamiento
deseado de cada parte del sistema. Desde el principio fue evidente
que el correcto funcionamiento de todo el sistema de Monitoreo Remoto de Datos podría
sólo podrá determinarse cuando se hayan utilizado todos los componentes. Una vez que se escribieron el
simulador, las pruebas de dispositivos integrados, las pruebas de servicios web y las pruebas de aplicaciones,
Fue una cuestión relativamente simple combinarlos para producir una prueba automatizada.
de todo el sistema. Una vez más se utilizaron hojas de cálculo Excel para realizar la prueba.
Se escribieron datos y clases Ruby para acceder a los datos y a los resultados esperados.

Las pruebas de un extremo a otro se complicaron por la respuesta impredecible del
Ruta de transmisión por satélite. Se estableció un valor de tiempo de espera predefinido y, si la prueba
El valor real no coincidía con el valor esperado, la prueba realizaría un ciclo hasta que
coincidieron o se alcanzó el tiempo de espera. Cuando el tiempo de espera expiró, la prueba fue
se considera que ha fracasado. La mayoría de los problemas de transmisión fueron encontrados y eliminados.

Por aquí. Habría sido muy improbable que los hubieran encontrado. con pruebas manuales, porque eran problemas esporádicos.

Debido a que las pruebas de extremo a extremo como estas pueden ser frágiles, es posible que no se mantengan como parte del conjunto de regresión automatizada. Si todos los componentes del sistema están bien cubiertos con pruebas de regresión automatizadas, pruebas automatizadas de un extremo a otro puede que no sea necesario. Sin embargo, debido a la naturaleza de este sistema, no era Es posible realizar una prueba completa sin automatización.

Pruebas de aceptación del usuario

La Prueba de Aceptación del Usuario (UAT) es la crítica final del producto por parte del cliente, quien debería haber estado involucrado en el proyecto desde el principio. En esto Por ejemplo, el cliente real estaba en Francia, a miles de kilómetros del equipo de desarrollo. El equipo tuvo que ser inventivo para tener una UAT exitosa. El cliente vino a trabajar con los miembros del equipo un par de veces durante el año y por eso pude interactuar con el equipo un poco más fácilmente que si hubieran nunca conocido.

Después de que el equipo introdujo el desarrollo ágil, Janet fue a Francia para facilitar la primera UAT en las instalaciones del cliente. Funcionó bastante bien y el lanzamiento fue aceptado después de que se solucionaron algunos problemas críticos. El equipo aprendió mucho de esa experiencia.

La segunda aprobación de la UAT se realizó internamente. Para prepararse, el equipo trabajó con el cliente para desarrollar un conjunto de pruebas que el cliente podría realizar para verificar la nueva funcionalidad. El cliente pudo probar la aplicación durante todo el ciclo de desarrollo, por lo que UAT no produjo ningún problema. El cliente Llegó, realizó las pruebas y cerró la sesión en un día.

No podemos enfatizar lo suficiente la importancia de trabajar con el cliente. Incluso Aunque el ingeniero de producto era el representante del cliente, era crucial Obtenga tiempo cara a cara con el cliente real. La relación que se había construido con el tiempo fue fundamental para el éxito del proyecto. Janet cree firmemente que La UAT tuvo éxito porque el cliente sabía lo que estaba haciendo el equipo. por el camino.

Fiabilidad

La confiabilidad, una de las "habilidades" abordadas por las pruebas del Cuadrante 4, fue un factor crítico. factor del sistema porque estaba monitoreando sitios remotos que a menudo eran

Consulte el Capítulo 10, "De cara al negocio Pruebas que critican el producto", para obtener más información sobre las pruebas del cuadrante 4, como pruebas de confiabilidad.

inaccesible, especialmente en invierno. El simulador que se desarrolló para probar el sistema integrado se configuró en un entorno separado y se ejecutó durante semanas midiendo la estabilidad (otra "ilidad") de todo el sistema. Las correcciones al diseño del sistema podrían planificarse y codificarse como necesario. Este es un buen ejemplo de por qué no deberías esperar hasta el final del proyecto para realizar las pruebas tecnológicas que critican el producto.

DOCUMENTACIÓN

En esta sección se presenta el enfoque adoptado para la documentación.

Documentar el código de prueba

Durante el desarrollo, quedó claro que un sistema de documentación formal era necesario para el código de prueba. La solución más sencilla fue utilizar RDoc, similar a Javadoc, pero para Ruby. RDoc extrajo comentarios etiquetados de la fuente código y páginas web generadas con detalles de archivos, clases y métodos. El Los documentos se generaron todas las noches utilizando un archivo por lotes y estaban disponibles al equipo completo. Fue fácil encontrar qué dispositivos de prueba se crearon.

La documentación del código de prueba ayudó a documentar las pruebas y hacer Fue más fácil encontrar lo que estábamos probando y lo que hicieron las pruebas. Era muy potente y fácil de usar.

Informar los resultados de la prueba

Aunque se estaban realizando pruebas exhaustivas, había poca evidencia de ello fuera del equipo de prueba. Los registros generados durante la automatización Las pruebas proporcionaron buena información para detectar problemas, pero no eran adecuadas para un público más amplio.

Para aumentar la visibilidad de las pruebas que se realizan, el equipo de pruebas desarrolló un Sistema de registro y generación de informes utilizando Apache, PHP y MySQL. cuando una prueba se ejecutó, registró el resultado en la base de datos. Un proyecto web front-end permitido partes interesadas para determinar qué pruebas se ejecutaron, la tasa de aprobación/falla y otra información.

También creímos en hacer visible nuestro progreso (bueno o malo) tanto como fuera posible. Con este fin, creamos tablas y gráficos a lo largo del camino y publicamos ellos en áreas comunes. La Figura 12-4 muestra algunos de los gráficos que creamos.

Capítulo 16, "Golpe the Ground Run-ning", ofrece más ejemplos de formas en que los equipos informan los resultados de las pruebas.

Capítulo 18, "Codificación y pruebas", También analiza los usos de gráficos grandes y visibles.

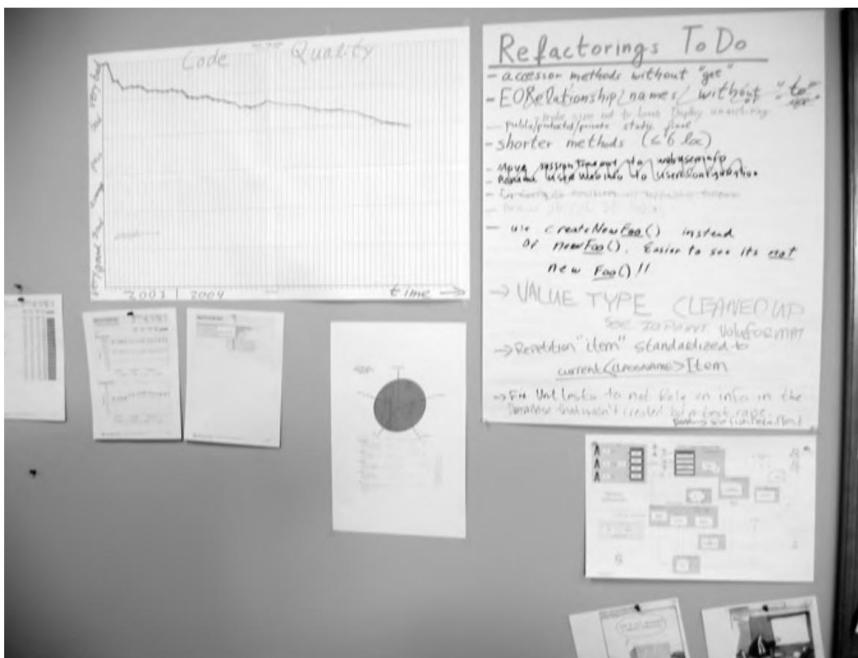


Figura 12-4 Grandes gráficos visibles utilizados por el equipo del proyecto del sistema de monitoreo remoto

UTILIZANDO LOS CUADRANTES DE PRUEBAS ÁGILES

Este ejemplo demuestra cómo las prácticas de prueba de las cuatro pruebas ágiles. Los cuadrantes se combinan durante la vida de un proyecto de desarrollo complejo para lograr una entrega exitosa. La experiencia de este equipo ilustra muchos de los principios que hemos estado enfatizando. Todo el equipo, incluidos los programadores, los evaluadores, el representante del cliente y el cliente real, contribuyeron a esfuerzos para resolver los problemas de automatización. Experimentaron con diferentes enfoques. Combinaron sus herramientas locales y de código abierto de diferentes maneras para realizar pruebas en todos los niveles, desde el nivel unitario hasta el de un extremo a otro. pruebas del sistema y UAT. El éxito del proyecto demuestra el éxito. del enfoque de prueba.

Mientras planifica cada epopeya, lanzamiento o iteración, trabaje con su equipo de clientes para comprender las prioridades del negocio y analizar los riesgos. Utilice los cuadrantes para ayudar a identificar todos los diferentes tipos de pruebas que serán necesarias y cuándo deben realizarse. ¿Es el desempeño el criterio más importante? Es el

¿La máxima prioridad es la capacidad de interactuar con otros sistemas? ¿Es quizás la usabilidad el aspecto más importante?

Invierta en una arquitectura de prueba que se adapte a la complejidad del sistema. bajo prueba. Planificar para obtener los recursos y la experiencia necesarios en el momento adecuado. para pruebas especializadas. Para cada tipo de prueba, su equipo debe trabajar en conjunto para Elija herramientas que resuelvan sus problemas de prueba. Utilice retrospectivas para evaluar continuamente si su equipo tiene los recursos que necesita para tener éxito y si todas las pruebas necesarias se están especificando a tiempo para cumplir su propósito, y automatizar adecuadamente.

¿Parece imposible realizar pruebas de un extremo a otro? ¿A tu equipo le resulta difícil escribir pruebas unitarias? Como hizo el equipo de Janet, haga que todos experimenten con diferentes enfoques y herramientas. Los cuadrantes proporcionan un marco para una lluvia de ideas productiva sobre formas creativas de lograr las pruebas que permitirán al El equipo aporta valor al negocio.

RESUMEN

En este capítulo, describimos un proyecto real que utilizó pruebas de los cuatro ágiles. cuadrantes de prueba para superar desafíos de prueba difíciles. Usamos ejemplos de este proyecto para mostrar cómo los equipos pueden tener éxito con todo tipo de pruebas. Algunas lecciones importantes del proyecto del Sistema de Monitoreo Remoto de Datos son:

Todo el equipo debe elegir o crear herramientas que resuelvan cada problema de prueba.

Es posible que se necesiten combinaciones de herramientas comerciales comunes, como hojas de cálculo y scripts de prueba personalizados, para lograr tareas complejas. pruebas.

Invierta tiempo en crear la arquitectura de prueba adecuada que funcione para todos los miembros del equipo.

Encuentre formas de mantener a los clientes involucrados en todo tipo de pruebas, incluso si se encuentran en una ubicación remota.

Informe los resultados de las pruebas de una manera que mantenga a todas las partes interesadas informadas sobre la iteración y el progreso del proyecto.

No olvides documentar... pero sólo lo que es útil.

Piense en los cuatro cuadrantes de las pruebas a lo largo de sus ciclos de desarrollo.

Utilice las lecciones aprendidas durante las pruebas para criticar el producto a fin de impulsar el desarrollo en iteraciones posteriores.

Esta página se dejó en blanco intencionalmente.

Parte IV

AUTOMATIZACIÓN

La automatización de pruebas es una práctica ágil fundamental. Los proyectos ágiles dependen de la automatización. Una automatización suficientemente buena libera al equipo para entregar código de alta calidad con frecuencia. Proporciona un marco que permite al equipo maximizar su velocidad mientras manteniendo un alto nivel. Control de código fuente, compilaciones y pruebas automatizadas. suites, implementación, monitoreo y una variedad de scripts y herramientas eliminan tedio, garantizar la confiabilidad y permitir que el equipo haga su mejor trabajo en todo momento.

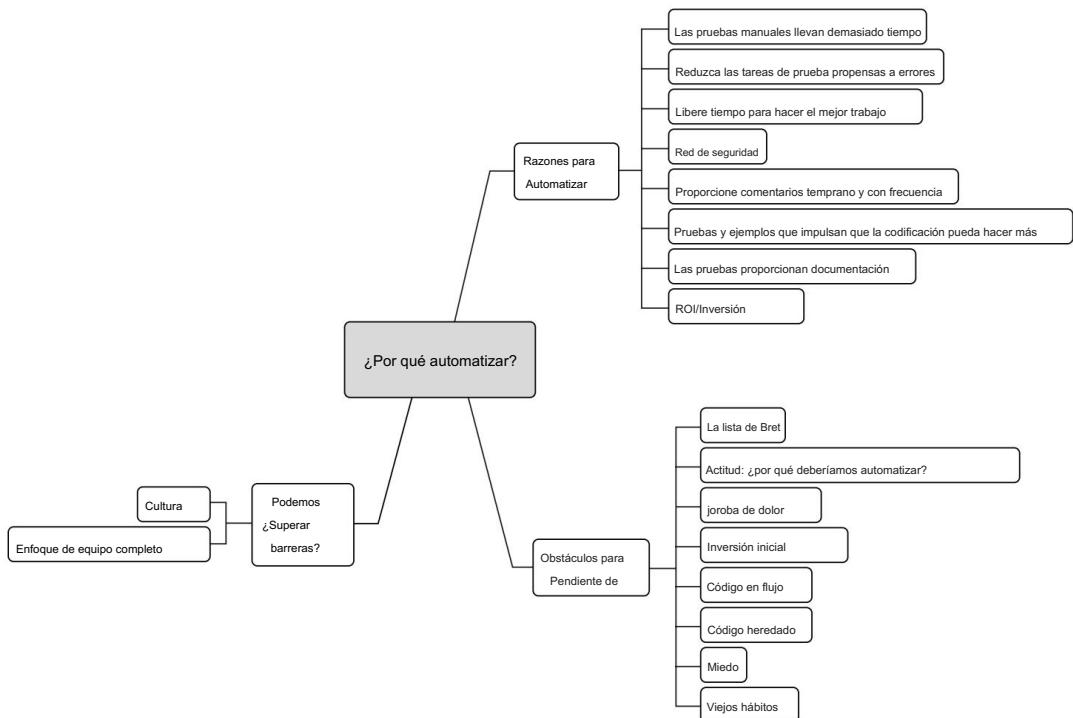
La automatización también es un tema amplio. Incluye tareas como escribir shells simples. scripts, configurar propiedades de sesión y crear pruebas automatizadas sólidas. La gama y el número de herramientas automatizadas parecen crecer exponencialmente a medida que avanzamos. Conozca mejores formas de producir software. Afortunadamente, el número de excelentes libros prestados que enseñan formas de automatizar parece crecer con la misma rapidez.

Este libro se centra en el papel del evaluador en el desarrollo ágil. Debido a que la automatización es clave para un desarrollo ágil exitoso, necesitamos hablar de ello, pero No puedo comenzar a cubrir todos los aspectos del tema. Lo que queremos explicar es por qué usted, como evaluador, debe adoptar la automatización y cómo usted y su equipo puede superar los numerosos obstáculos que pueden obstaculizar sus esfuerzos de automatización. Esta sección describe cómo puede aplicar valores, principios y prácticas ágiles para desarrollar una estrategia de automatización práctica, superar barreras y ganar impulso en la automatización de pruebas.

Esta página se dejó en blanco intencionalmente.

Capítulo 13

POR QUÉ QUEREMOS AUTOMATIZAR PRUEBAS Y LO QUE NOS DETIENE



¿Por qué automatizamos las pruebas, el proceso de construcción, la implementación y otras tareas? Los equipos ágiles se centran en tener siempre software en funcionamiento, lo que les permite lanzar software listo para producción con la frecuencia necesaria. Lograr este objetivo requiere pruebas constantes. En este capítulo, analizamos las razones por las que queremos automatizar y los desafíos que dificultan el avance de la automatización.

¿POR QUÉ AUTOMATIZAR?

Hay varias razones para automatizar además de decir que es necesario tener La automatización para tener éxito utilizando Agile. Nuestra lista incluye lo siguiente:

- Las pruebas manuales llevan demasiado tiempo.
- Los procesos manuales son propensos a errores.
- La automatización libera a las personas para que puedan hacer su mejor trabajo.
- Las pruebas de regresión automatizadas proporcionan una red de seguridad.
- Las pruebas automatizadas brindan retroalimentación temprana y con frecuencia.
- Las pruebas y ejemplos que impulsan la codificación pueden hacer más.
- Las pruebas proporcionan documentación.
- La automatización puede ser un buen retorno de la inversión.

Exploraremos cada uno de estos con un poco más de detalle.

Las pruebas manuales llevan demasiado tiempo

La razón más básica por la que un equipo quiere automatizar es que simplemente requiere demasiado mucho tiempo para completar todas las pruebas necesarias manualmente. Como tu aplicación se hace más y más grande, el tiempo para probar todo se hace más y más largo, a veces exponencialmente, dependiendo de la complejidad de la AUT (Aplicación bajo prueba).

Los equipos ágiles pueden entregar software listo para producción al final de cada iteración corta al tener software listo para producción todos los días. ejecutando un un conjunto completo de pruebas de regresión al menos diariamente es una práctica indispensable, y no puedes hacerlo con pruebas de regresión manual. Si no tiene ninguna automatización ahora, tendrá que realizar la prueba de regresión manualmente, pero no deje que eso se detenga. que usted comience a automatizarlo.

Si ejecuta sus pruebas de regresión manualmente, llevará cada vez más tiempo probando todos los días, en cada iteración. Para que las pruebas sigan el ritmo de la codificación, los programadores deben tomarse el tiempo para ayudar con las pruebas de regresión manual o el equipo debe contratar más evaluadores. Inevitablemente, tanto los aspectos técnicos la deuda y la frustración crecerán.

Si el código ni siquiera tiene que pasar un conjunto de pruebas automatizadas de regresión a nivel unitario, los evaluadores probablemente dedicarán gran parte de su tiempo a investigar, intentar reproducir e informar esos errores simples, y menos tiempo a encontrar errores potencialmente graves a nivel del sistema. Además, debido a que el equipo no realiza desarrollos de prueba primero, es más probable que el diseño del código sea menos comprobable y Es posible que no proporcione la funcionalidad deseada por la empresa.

Probar manualmente una serie de escenarios diferentes puede llevar mucho tiempo, especialmente si está ingresando entradas en una interfaz de usuario. Configurar datos para una variedad de escenarios complejos puede ser una tarea abrumadora si no se cuenta con sistemas automatizados. De esta manera de acelerarlo. Como resultado, sólo un número limitado de escenarios pueden ser probados y se pueden pasar por alto defectos importantes.

Los procesos manuales son propensos a errores

Las pruebas manuales se vuelven repetitivas, especialmente si sigues pruebas escritas, y las pruebas manuales se vuelven aburridas muy rápidamente. Es demasiado fácil cometer errores y pasar por alto incluso errores simples. Se omitirán pasos e incluso pruebas completas. Si el equipo se enfrenta a una fecha límite ajustada, existe la tentación de tomar atajos y la El resultado es un problema omitido.

Debido a que las pruebas manuales son lentas, es posible que aún estés realizando pruebas a la medianoche del día último día de la iteración. ¿Cuántos errores notarás entonces?

Las compilaciones, la implementación, el control de versiones y la supervisión automatizados también contribuyen Un largo camino para mitigar el riesgo y hacer que su proceso de desarrollo sea más coherente. La automatización de estas pruebas programadas elimina la posibilidad de errores, porque cada prueba se realiza exactamente de la misma manera cada vez.

El dicho de "construir una vez, implementar en muchas" es el sueño de todo evaluador hecho realidad. La automatización de los procesos de construcción e implementación le permite saber exactamente qué está probando en cualquier entorno determinado.

La automatización libera a las personas para hacer su mejor trabajo

Escribir código primero como prueba ayuda a los programadores a comprender los requisitos y diseñar el código en consecuencia. Tener compilaciones continuas ejecuta todas las pruebas unitarias y Las pruebas de regresión funcional significan más tiempo para realizar exploraciones interesantes. pruebas. Automatizar la configuración de las pruebas exploratorias significa aún más tiempo para sondear partes potencialmente débiles del sistema. porque no gastaste tiempo ejecutando tediosos scripts manuales, tienes la energía para hacer un buen trabajo, pensando en diferentes escenarios y aprendiendo más sobre cómo funciona la aplicación. obras.

Si pensamos constantemente en cómo automatizar las pruebas para una solución o una nueva característica, es más probable que pensemos en la capacidad de prueba y en un diseño de calidad en lugar de en una truco rápido que podría resultar frágil. Eso significa mejor código y mejores pruebas.

La automatización de las pruebas puede ayudar a mantener la coherencia en toda la aplicación.

La historia de Janet

Consulte el Capítulo 9, "Kit de herramientas para empresas Pruebas que respaldan al equipo", Capítulo 12, "Resumen de los cuadrantes de pruebas" y Capítulo 14, "Una estrategia ágil de automatización de pruebas", para obtener más información sobre Ruby y Watir.

Jason (uno de mis compañeros evaluadores) y yo estábamos trabajando en algunos scripts de automatización de GUI usando Ruby y Watir, y estábamos agregando constantes para los nombres de los botones para las pruebas. Rápidamente nos dimos cuenta de que los botones de cada página no tenían nombres consistentes. Pudimos cambiarlos y resolvimos esos problemas de coherencia muy rápidamente, y tuvimos una manera fácil de hacer cumplir las convenciones de nomenclatura.

—Janet

Libros como Pragmatic Project Automation [2004] pueden guiarle en la automatización de las tareas diarias de desarrollo y liberar a su equipo para actividades importantes, como las pruebas exploratorias.

Dar a los evaluadores un mejor trabajo

Chris McMahon describió los beneficios que experimentó debido a la automatización de las pruebas de regresión en una publicación en la lista de correo de pruebas ágiles en noviembre de 2007:

Nuestra automatización de pruebas de regresión de UI ha crecido un 500% desde abril [de 2007]. Esto nos permite centrar la atención de seres humanos reales en pruebas más interesantes.

Chris continuó explicando: "Ahora que tenemos mucha automatización, tenemos tiempo libre para pensar realmente en lo que es necesario realizar las pruebas en humanos. Para cualquier prueba que no sea trivial, prácticamente hemos institucionalizado una sesión de lluvia de ideas de prueba antes de comenzar la ejecución". Por lo general, Chris y sus compañeros de equipo combinan dos evaluadores o un evaluador y un desarrollador.

A veces, un evaluador genera ideas y las revisa a través de un mapa mental, una página wiki o una lista en las notas de la versión. Chris observó: "Casi siempre se nos ocurren buenas ideas de prueba mediante emparejamientos que ninguno de los individuos habría encontrado de forma independiente".

Refiriéndose a sus frecuentes lanzamientos de funciones importantes, Chris dice: "Gracias a la buena automatización de pruebas, tenemos tiempo para invertir en asegurarnos de que todo el producto sea atractivo y funcional para personas reales. Sin la automatización, probar este producto sería aburrido y estúpido. Tal como están las cosas, los evaluadores tenemos un trabajo importante e interesante que hacer para cada versión".

Estamos de acuerdo con Chris en que la parte más interesante de la automatización de pruebas es la forma en que amplía nuestra capacidad para mejorar el producto a través de pruebas exploratorias innovadoras.

Los proyectos tienen éxito cuando las buenas personas son libres de hacer su mejor trabajo. La automatización adecuada de las pruebas hace que eso suceda. Pruebas de regresión automatizadas que detectar cambios en la funcionalidad existente y proporcionar comentarios inmediatos. un componente principal de esto.

Las pruebas de regresión automatizadas proporcionan una red de seguridad

La mayoría de los profesionales que han estado en el negocio del software durante algunos años saben la sensación de pavor cuando se enfrentan a corregir un error o implementar una Nueva característica en código mal diseñado que no está cubierto por pruebas automatizadas. Apriete un extremo del globo aquí y otra parte sobresaldrá. ¿Será ¿romper?

Saber que el código tiene suficiente cobertura mediante pruebas de regresión automatizadas proporciona un gran sentimiento de confianza. Claro, un cambio puede producir un efecto inesperado, pero lo sabremos en cuestión de minutos si es a nivel de unidad.

u horas si se encuentra en un nivel funcional superior. Hacer el cambio primero significa probar pensar en el comportamiento cambiado antes de escribir el código y escribir un prueba para verificarlo, lo que aumenta esa confianza.

La historia de Janet

Recientemente tuve una conversación con uno de los evaluadores de mi equipo que cuestionó el valor de las pruebas automatizadas. Mi primera respuesta fue "Es una red de seguridad" para el equipo.

Sin embargo, cuestionó esa premisa. ¿No nos volvemos dependientes de las pruebas en lugar de solucionar la causa raíz del problema?

Me hizo pensar un poco más en mi respuesta. Tenía razón en un sentido: Si nos volvemos complacientes con los desafíos de nuestras pruebas y dependemos únicamente de las pruebas automatizadas para encontrar nuestros problemas, y luego simplemente los solucionamos lo suficiente para que la prueba pase, no nos hacemos ningún favor.

Sin embargo, si utilizamos las pruebas para identificar áreas problemáticas y solucionarlas de la manera correcta o refactorizarlas según sea necesario, entonces estamos utilizando la red de seguridad de la automatización de la manera correcta. La automatización es fundamental para el éxito de un proyecto ágil, especialmente a medida que la aplicación crece en tamaño.

—Janet

Cuando no cuentan con un conjunto automatizado de pruebas que actúe como red de seguridad, el los programadores pueden comenzar a ver a los propios evaluadores como una red de seguridad. Es fácil imaginar que el proceso de pensamiento de Joe Programmer es el siguiente: "Debería Regrese y agregue algunas pruebas unitarias automatizadas para formatEmployeeInfo, pero no Sé que Susie Tester revisará cada página donde se use manualmente. Ella verá si algo anda mal, así que simplemente estaría duplicando su esfuerzo".

Es bueno que un programador piense tan bien en los talentos del evaluador, pero Joe se dirige hacia una pendiente resbaladiza. Si no automatiza estas pruebas unitarias, ¿qué otras pruebas podría omitir? Susie va a estar tremadamente ocupada mirando todas esas páginas.

Los equipos que tienen una buena cobertura de las pruebas de regresión automatizadas pueden realizar cambios en el código sin miedo. No tienen que preguntarse: "Si cambio este módulo `formatEmployeeInfo`, ¿romperé algo en la interfaz de usuario?"

Las pruebas les dirán de inmediato si rompieron algo o no. Pueden ir mucho más rápido que los equipos que dependen exclusivamente de pruebas manuales.

Las pruebas automatizadas brindan retroalimentación, tempranamente y con frecuencia. Despues de que se supera una prueba automatizada para una funcionalidad, debe continuar pasando hasta que la funcionalidad se cambie intencionalmente. Cuando planificamos cambios en la aplicación, cambiamos las pruebas para adaptarnos a ellos. Cuando una prueba automatizada falla inesperadamente, es posible que un cambio de código haya introducido un defecto de regresión. La ejecución de un conjunto automatizado de pruebas cada vez que se registra un código nuevo ayuda a garantizar que los errores de regresión se detecten rápidamente. La retroalimentación rápida significa que el cambio aún está fresco en la mente de algunos programadores, por lo que la solución de problemas será más rápida que si el error no se encontrara hasta alguna fase de prueba semanas después. Fallar rápidamente significa que los errores son más baratos de corregir.

Las pruebas automatizadas se ejecutan periódicamente y, a menudo, actúan como detector de cambios. Le permiten al equipo la oportunidad de saber qué ha cambiado desde la última construcción. Por ejemplo, ¿hubo efectos secundarios negativos con la última versión? Si su suite de automatización tiene suficiente cobertura, puede detectar fácilmente efectos de gran alcance que los evaluadores manuales nunca podrán encontrar.

La mayoría de las veces, si las pruebas de regresión no están automatizadas, no se ejecutarán en cada iteración, y mucho menos todos los días. El problema surge muy rápidamente durante el final del juego, cuando el equipo necesita completar todas las pruebas de regresión.

Los errores que se habrían detectado al principio se encuentran al final del juego. Muchos de los beneficios de realizar pruebas tempranas se pierden.

Pruebas y ejemplos que impulsan que la codificación pueda hacer más

En el Capítulo 7, "Pruebas tecnológicas que respaldan al equipo", hablamos sobre el uso de pruebas y ejemplos para impulsar la codificación. Hemos hablado de lo importante que es impulsar la codificación con pruebas tanto unitarias como de clientes. También queremos enfatizar que si estas pruebas se automatizan, se vuelven valiosas por una razón diferente. Se convierten en la base de un conjunto de regresión muy sólido.

La historia de Lisa

Después de que mi equipo se familiarizó con las pruebas unitarias, la refactorización, la integración continua y otras prácticas tecnológicas, pudimos detectar errores de regresión y funcionalidades implementadas incorrectamente durante el desarrollo.

Por supuesto, esto no significó que nuestros problemas estuvieran completamente resueltos; A veces todavía pasamos por alto o malinterpretamos los requisitos. Sin embargo, contar con un marco de automatización nos permitió comenzar a centrarnos en hacer un mejor trabajo a la hora de capturar los requisitos en las pruebas iniciales. También tuvimos más tiempo para pruebas exploratorias.

Con el tiempo, nuestra tasa de defectos disminuyó drásticamente, mientras que el deleite de nuestros clientes con el valor comercial entregado aumentó.

—Lisa

La bibliografía contiene un artículo de Jennitta Andrea [2008] sobre etiqueta de equipo para TDD.

TDD y SDD (desarrollo basado en pruebas de historias) mantienen a los equipos pensando primero en las pruebas. Durante las reuniones de planificación, hablan sobre las pruebas y la mejor manera de hazlo. Diseñan código para que las pruebas pasen, por lo que la capacidad de prueba nunca es un problema. El conjunto de pruebas automatizadas crece junto con el código base, proporcionando una red de seguridad para una refactorización constante. Es importante que todo el equipo practique TDD y escriba consistentemente pruebas unitarias, o la red de seguridad tendrá agujeros.

El equipo tampoco acumula demasiada deuda técnica y su velocidad es seguramente será estable o incluso aumentará con el tiempo. Esa es una de las razones por las que Los gerentes comerciales deberían estar felices de permitir que los equipos de software se tomen el tiempo para implementar buenas prácticas correctamente.

Las pruebas son una excelente documentación

En la Parte III, explicamos cómo los equipos ágiles utilizan ejemplos y pruebas para guiar el desarrollo. Cuando se automatizan pruebas que ilustran ejemplos de comportamiento deseado, se convierten en documentación "viva" de cómo funciona realmente el sistema. obras. Es bueno tener documentación narrativa sobre cómo funciona una funcionalidad, pero nadie puede discutir con una prueba ejecutable que muestra en rojo y verde cómo opera el código en un conjunto determinado de entradas.

Es difícil mantener actualizada la documentación estática, pero si no actualizamos nuestra pruebas automatizadas cuando el sistema cambia, las pruebas fallan. Necesitamos arreglarlos para mantener nuestro proceso de construcción "verde". Esto significa que las pruebas automatizadas siempre son una imagen precisa de cómo funciona nuestro código. Esa es sólo una de las formas en que nuestro La inversión en automatización vale la pena.

ROI y recuperación de la inversión

Todas las razones que acabamos de presentar contribuyen al resultado final y a la rentabilidad de la automatización. La automatización proporciona coherencia a un proyecto y da la oportunidad del equipo de probar de manera diferente y superar los límites de la aplicación. La automatización significa tiempo adicional para que los evaluadores y los miembros del equipo se concentren en lanzar al mercado el producto adecuado en el momento oportuno.

Un componente importante de la recuperación de la automatización de pruebas es la forma en que se solucionan los defectos fijado. Los equipos que dependen de pruebas manuales tienden a encontrar errores mucho tiempo después del código. que contiene el error está escrito. Se ponen en modo de arreglar el “error de el día”, en lugar de buscar la causa raíz del error y rediseñar el código en consecuencia. Cuando los programadores ejecutan el conjunto de pruebas automatizadas en su propia zona de pruebas, las pruebas de regresión automatizadas encuentran los errores antes de que se ejecute el código. registrado, así que hay tiempo para corregir el diseño. Esa es una recompensa mucho mayor y así es como se reduce la deuda técnica y se desarrolla un código sólido.

BARRERAS A LA AUTOMATIZACIÓN: COSAS QUE ESTORBAR

En 2001, Bret Pettichord [2001] enumeró siete problemas que afectan a la automatización. Siguen siendo aplicables, pero están destinados a equipos que no incorporan la automatización como parte de su desarrollo. Y por supuesto, porque estás haciendo ágil, estás haciendo eso, ¿verdad?

Nos gustaría pensar que todo el mundo ha incluido tareas de automatización como parte de cada historia, pero la realidad es que probablemente no estarías leyendo esto sección si lo tuvieras todo bajo control. Hemos incluido la lista de Bret para mostrar qué problemas que probablemente tenga si no incluye la automatización como parte del Entregables cotidianos del proyecto.

La lista de Bret

La lista de problemas de automatización de Bret se ve así:

Usar únicamente el tiempo libre para la automatización de pruebas no le da el enfoque que necesita.

Faltan objetivos claros.

Hay una falta de experiencia.

Hay mucha rotación, porque pierdes toda la experiencia que puedas tener.

Una reacción a la desesperación es a menudo la razón por la que se elige la automatización, en cuyo caso puede ser más un deseo que una propuesta realista.

Puede haber renuencia a pensar en realizar pruebas; La diversión está en la automatización, no en las pruebas.

Centrarse en resolver el problema tecnológico puede hacer que pierda de vista si el resultado satisface las necesidades de la prueba.

Creemos que hay otros problemas con los que se topan los equipos cuando intentan automatizar. Incluso si intentamos incluir la automatización en los entregables de nuestro proyecto, Hay otras barreras para el éxito. En la siguiente sección presentamos nuestra lista de Obstáculos para una automatización de pruebas exitosa.

Nuestra lista

Nuestra lista de barreras para la automatización de pruebas exitosa se basa en las experiencias Hemos tenido con nuestros propios equipos ágiles y con los de otros equipos que conocemos.

Actitud de los programadores

La "joroba del dolor"

Inversión inicial

Código que siempre está en constante cambio

Sistemas heredados

Miedo

viejos hábitos

Actitud de los programadores: "¿Por qué automatizar?"

Programadores que estén acostumbrados a trabajar en un entorno tradicional, donde algún equipo de control de calidad separado e invisible hace todas las pruebas y puede que ni siquiera proporcione Se ha pensado mucho en la automatización de pruebas funcionales. Algunos programadores no se molestan probar mucho porque tienen el equipo de control de calidad como red de seguridad para detectar errores antes del lanzamiento. Los largos ciclos de desarrollo en cascada hacen que las pruebas sean aún más remotas para los programadores. Para cuando los probadores invisibles estén haciendo su trabajo, el Los programadores han pasado a la siguiente versión. Los defectos entran en una cola para arreglarse después con grandes gastos y nadie es responsable de haberlos producido. Incluso los programadores que han adoptado el desarrollo basado en pruebas y están acostumbrados a automatizar pruebas a nivel de unidad, es posible que no piensen en cómo Se realizan pruebas de aceptación más allá del nivel unitario.

La historia de Lisa

Una vez me uní a un equipo XP de programadores expertos que practicaban el desarrollo basado en pruebas y que tenían un conjunto razonable de pruebas unitarias ejecutándose en un proceso de compilación automatizado. Nunca habían automatizado ninguna prueba empresarial, por lo que un día comencé una discusión sobre qué herramientas podrían usar para automatizar pruebas de regresión funcionales orientadas al negocio. Los programadores querían saber por qué necesitábamos automatizar estas pruebas.

Al final de la primera iteración, cuando todos estaban ejecutando las pruebas de aceptación a mano, señalé que todas estas pruebas se realizarían nuevamente en la siguiente iteración como pruebas de regresión, además de las pruebas para todas las nuevas historias. En la tercera iteración, habría tres veces más pruebas. Para un evaluador, parece ridículamente obvio, pero a veces los programadores necesitan realizar pruebas manuales antes de comprender la obligación de automatizarlas.

—Lisa

La educación es la clave para lograr que los programadores y el resto del equipo comprendan la importancia de la automatización.

La “joroba del dolor” (la curva de aprendizaje)

Es difícil aprender a automatizar pruebas, especialmente aprender a hacerlo de una manera que produce un buen retorno de los recursos invertidos en él. Un término que hemos escuchado Brian Marick usa para describir la fase inicial de automatización que los desarrolladores (incluidos los evaluadores) tienen que superar es la “JOROBAD DEL DOLOR” (ver Figura 13-1). Esta frase hace referencia a la lucha que atraviesan la mayoría de los equipos a la hora de adoptar automatización.

A menudo se espera que los nuevos equipos adopten prácticas como TDD y refactorización, que son difíciles de aprender. Sin un buen entrenamiento, hay mucho tiempo para dominar nuevas habilidades y un fuerte apoyo de la gerencia, se desaniman fácilmente.

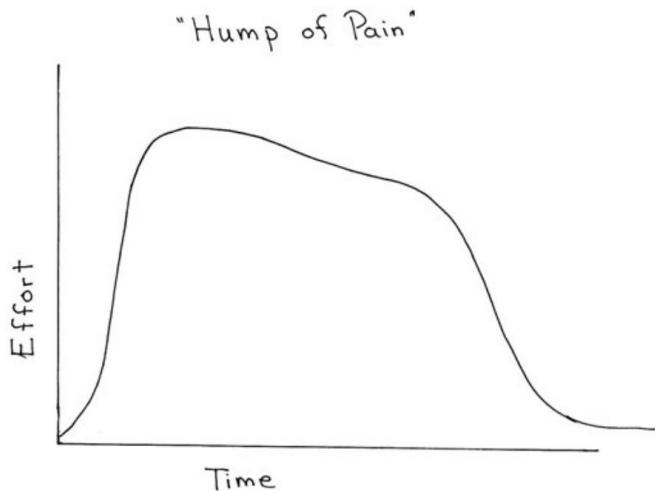


Figura 13-1 Joroba dolorosa de la curva de aprendizaje de la automatización

Envejecido. Si tienen obstáculos adicionales para aprender, como tener que trabajar con código heredado mal diseñado, puede parecer imposible alguna vez conseguir tracción automática de pruebas.

La historia de Lisa

Mi equipo en ePlan Services originalmente intentó escribir pruebas unitarias para un sistema heredado que definitivamente no fue escrito teniendo en cuenta las pruebas. Descubrieron que esta era una tarea difícil, si no imposible, por lo que decidieron codificar todas las historias nuevas en una arquitectura nueva y comprobable. Curiosamente, aproximadamente un año después, descubrieron que en realidad no era tan difícil escribir pruebas unitarias para el código antiguo. El problema era que no sabían en absoluto cómo escribir pruebas unitarias y era más fácil aprender con una arquitectura bien diseñada.

Escribir pruebas a nivel unitario se convirtió simplemente en una parte natural de escribir código.

—Lisa

La joroba de dolor puede ocurrir porque estás construyendo tu dominio específico marco de prueba o aprender su nueva herramienta de prueba funcional. Es posible que desee Trae a un experto para que te ayude a configurarlo correctamente.

Sabes que tu equipo ha superado el "problema" cuando la automatización se convierte en, Si no es fácil, al menos es un proceso natural y arraigado. Lisa ha trabajado en tres equipos que adoptaron con éxito TDD y la automatización de pruebas funcionales. Cada Al mismo tiempo, el equipo necesitó mucho tiempo, capacitación, compromiso y estímulo para lograr avances en las prácticas.

Inversión inicial

Incluso con todo el equipo trabajando en el problema, la automatización requiere una gran inversión, una que tal vez no dé sus frutos de inmediato. Se necesita tiempo e investigación para decidir qué marcos de prueba usar y si construirlos.

internamente o utilizar herramientas producidas externamente. Nuevo hardware y software están probablemente sea necesario. Los miembros del equipo pueden tardar un poco en aprender cómo Utilice arneses de prueba automatizados.

Muchas personas han experimentado esfuerzos de automatización de pruebas que no dieron resultado. Es posible que su organización haya comprado una herramienta de captura y reproducción de un proveedor, Se lo entregó al equipo de control de calidad y esperaba que resolviera todos los problemas de automatización. Estas herramientas suelen permanecer en un estante acumulando polvo. Puede que haya habido miles de líneas de scripts de prueba GUI generados, sin que quede nadie que sepa lo que hacen, o los scripts de prueba que son imposibles de mantener ya no están útil.

La historia de Janet

Entré a una organización como nuevo gerente de control de calidad. Una de mis tareas era evaluar los scripts de prueba automatizados actuales y aumentar la cobertura de la prueba. Unos años antes se había comprado una herramienta de proveedor y los evaluadores que habían desarrollado la suite inicial ya no estaban en la organización. Uno de los nuevos evaluadores contratados estaba intentando aprender a utilizar la herramienta y estaba agregando pruebas a la suite.

Lo primero que hice fue pedirle a este evaluador que realizara una evaluación en el conjunto de pruebas para ver cuál era realmente la cobertura. Pasó una semana intentando comprender cómo se organizaban las pruebas. También comencé a husmear y descubrí que las pruebas existentes estaban muy mal diseñadas y tenían muy poco valor.

Dejamos de agregar más pruebas y, en cambio, dedicamos un poco de tiempo a comprender cuál era el objetivo de nuestra automatización de pruebas. Al final resultó que, la herramienta del proveedor no podía hacer lo que realmente necesitábamos, por lo que cancelamos las licencias y encontramos una herramienta de código abierto que satisfizo nuestras necesidades.

Todavía teníamos que dedicar tiempo a aprender la nueva herramienta de código abierto, pero esa inversión se habría realizado si nos hubiéramos quedado con la herramienta original del proveedor de todos modos, porque nadie en el equipo sabía cómo usar la herramienta original.

—Janet

Las habilidades de diseño de pruebas tienen un gran impacto en si la automatización vale la pena lejos. Las malas prácticas producen pruebas que son difíciles de entender y mantener, y puede producir resultados difíciles de interpretar o fallos falsos que toman tiempo para investigación. Equipos con formación y habilidades inadecuadas podrían decidir el regreso en su inversión en automatización no vale la pena.

Las buenas prácticas de diseño de pruebas producen pruebas simples, bien diseñadas, continuamente refactorizadas y mantenibles. Las bibliotecas de módulos y objetos de prueba se acumulan tiempo y agilizar la automatización de nuevas pruebas. Consulte el Capítulo 14 para obtener algunos consejos y directrices para el diseño de pruebas para la automatización.

Sabemos que no es fácil capturar métricas. Por ejemplo, intentar capturar el tiempo que lleva escribir y mantener pruebas automatizadas versus el tiempo que lleva ejecutar las mismas pruebas de regresión manualmente es casi imposible. De manera similar, intentar para capturar cuánto cuesta reparar defectos a los pocos minutos de introducir ellos versus cuánto cuesta encontrar y solucionar problemas después del final de la iteración también es bastante difícil. Muchos equipos no hacen el esfuerzo de rastrear esto. información. Sin números que demuestren que automatizar requiere menos esfuerzo y proporciona más valor, es más difícil para los equipos convencer a la gerencia de que Vale la pena invertir en automatización. La falta de métricas que demuestren el retorno de la inversión de la automatización también hace que sea más difícil cambiar una hábitos del equipo.

Código que siempre está en constante cambio

Automatizar pruebas a través de la interfaz de usuario es complicado, porque las UI tienden a cambiar frecuentemente durante el desarrollo. Ésa es una de las razones por las que ese simple registro y las técnicas de reproducción rara vez son una buena opción para un proyecto ágil.

Si el equipo tiene dificultades para producir un buen diseño en el negocio subyacente.

acceso a la lógica y a la base de datos, y se realizan retrabajos importantes con frecuencia, podría ser

Es difícil mantenerse al día incluso con las pruebas automatizadas detrás de la GUI a nivel de API. Si

Si se presta poca atención a las pruebas al diseñar el sistema, puede resultar difícil y costoso encontrar una forma de automatizar las pruebas. Los programadores

Los evaluadores deben trabajar juntos para obtener una aplicación comprobable.



En el capítulo 14, "Un Estrategia ágil de automatización de pruebas", Veremos formas de organizar pruebas automatizadas.

Aunque el código y la implementación reales, como la GUI, tienden a cambiar

Con frecuencia, en el desarrollo ágil, la intención del código rara vez cambia. Organizar el código de prueba según la intención de la aplicación, en lugar de según su implementación.

le permite mantenerse al día con el desarrollo.

Código heredado

Según nuestra experiencia, es mucho más fácil impulsar la automatización si se escribe código nuevo en una arquitectura diseñada teniendo en cuenta las pruebas. Escribiendo

Las pruebas para código existente que tiene pocas o ninguna prueba es, en el mejor de los casos, una tarea desalentadora. Parece prácticamente imposible para un equipo nuevo en lo ágil y nuevo en la automatización de pruebas.

A veces es un callejón sin salida. Quieres automatizar pruebas para poder refactorizar parte del código heredado, pero el código heredado no está diseñado para la capacidad de prueba, por lo que Es difícil automatizar pruebas incluso a nivel unitario.

Si su equipo enfrenta este tipo de desafío y no tiene suficiente tiempo para

Si se piensa en cómo abordarlo, será difícil comenzar a automatizar las pruebas de manera efectiva. El Capítulo 14 ofrece estrategias para abordar estos problemas.

Miedo

La automatización de pruebas da miedo a quienes nunca la dominan, e incluso a algunos

quien tiene. Los programadores pueden ser buenos escribiendo código de producción, pero

Es posible que no tenga mucha experiencia en la redacción de pruebas automatizadas. Los probadores no pueden

Tienen una sólida experiencia en programación y no confían en su potencial.

Habilidades de automatización de pruebas.

Los evaluadores que no son de programación a menudo han recibido el mensaje de que han

Nada que ofrecer en el mundo ágil. Creemos lo contrario. Sin probador individual

Debería tener que preocuparse por cómo realizar la automatización. Es un problema de equipo y Normalmente hay muchos programadores en el equipo que pueden ayudar. El truco es aceptar el aprendizaje de nuevas ideas. Tómate un día a la vez.

Viejos hábitos

Cuando las iteraciones no se desarrollan sin problemas y el equipo no puede completar todas las tareas de programación y prueba al final de una iteración, los miembros del equipo puede entrar en pánico. Hemos observado que cuando las personas entran en modo de pánico, caen en viejos hábitos cómodos, incluso si esos hábitos nunca produjeron buenos resultados.

Entonces podemos decir: "Se supone que debemos entregar el 1 de febrero. Si queremos cumplir Esa fecha, no tenemos tiempo para automatizar ninguna prueba. Tendremos que hacer lo que sea Las pruebas manuales se pueden realizar en esa cantidad de tiempo y esperar lo mejor. Nosotros Siempre podemos automatizar las pruebas más adelante".

Este es el camino a la perdición. Se pueden realizar algunas pruebas manuales, pero tal vez no las importantes pruebas exploratorias manuales que habrían encontrado el error que le costó a la empresa cientos de miles de dólares en ventas perdidas. Luego, debido a que no terminamos con nuestras tareas de automatización de pruebas, esas tareas se trasladan a la siguiente iteración, reduciendo la cantidad de valor empresarial que podemos ofrecer. A medida que avanzan las iteraciones, la situación continúa deteriorándose.

¿ PODEMOS SUPERAR ESTAS BARRERAS?

Consulte el Capítulo 3,

"Desafíos culturales", para obtener algunas ideas sobre cómo realizar cambios en la cultura del equipo para facilitar las prácticas ágiles.

El enfoque ágil de todo el equipo es la base para superar los desafíos de la automatización. Los programadores que son nuevos en el mundo ágil probablemente estén acostumbrados a ser recompensados por entregar código, tenga errores o no, siempre y cuando cumplir con los plazos. El desarrollo basado en pruebas está más orientado al diseño. que las pruebas, por lo que es posible que las pruebas orientadas a los negocios aún no entren en su conciencia. Se necesita liderazgo y un compromiso de equipo con la calidad para que todos piensen en cómo escribir, usar y ejecutar pruebas tanto orientadas a la tecnología como a los negocios. Involucrar a todo el equipo en la automatización de pruebas puede ser una desafío cultural.

En el siguiente capítulo, mostramos cómo utilizar valores y principios ágiles para superar algunos de los problemas que hemos descrito en este capítulo.

RESUMEN

En este capítulo, analizamos algunos factores importantes relacionados con la automatización de pruebas:

Necesitamos que la automatización proporcione una red de seguridad, nos brinde retroalimentación esencial, mantenga la deuda técnica al mínimo y ayude a impulsar la codificación.

El miedo, la falta de conocimiento, las experiencias pasadas negativas con la automatización, el código que cambia rápidamente y el código heredado se encuentran entre las barreras comunes a la automatización.

Automatizar las pruebas de regresión, ejecutarlas en un proceso de compilación automatizado y solucionar las causas fundamentales de los defectos reduce la deuda técnica y permite el crecimiento de código sólido.

La automatización de las pruebas de regresión y las tediosas tareas manuales libera al equipo para realizar trabajos más importantes, como las pruebas exploratorias.

Los equipos con pruebas automatizadas y procesos de construcción automatizados disfrutan de una velocidad más estable.

Sin pruebas de regresión automatizadas, las pruebas de regresión manuales seguirán creciendo en alcance y, eventualmente, es posible que simplemente se ignoren.

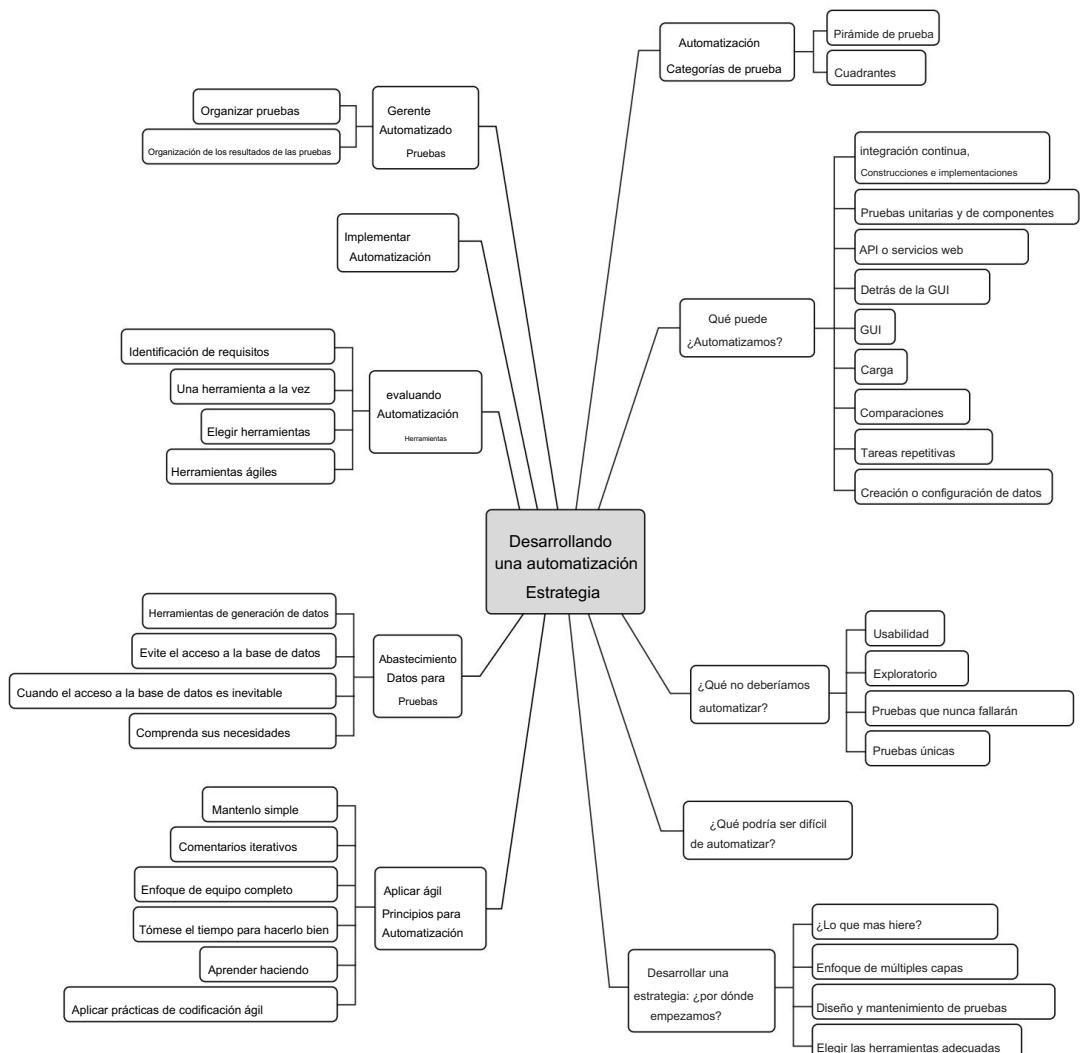
La cultura y la historia del equipo pueden hacer que sea más difícil para los programadores priorizar la automatización de las pruebas empresariales en lugar de codificar nuevas funciones.

El uso de principios y valores ágiles ayuda a todo el equipo a superar las barreras para la automatización de pruebas.

Esta página se dejó en blanco intencionalmente.

Capítulo 14

UNA PRUEBA ÁGIL ESTRATEGIA DE AUTOMATIZACIÓN



Mientras exploramos cada uno de los cuadrantes de pruebas ágiles en la Parte III, brindamos ejemplos de herramientas que pueden ayudar a que esos diferentes esfuerzos de prueba tengan éxito. Muchas de esas herramientas sirven para automatizar pruebas. Como describimos en el capítulo anterior, los equipos enfrentan muchos obstáculos en su búsqueda de una automatización de pruebas exitosa. Cada vez hay mejores herramientas disponibles, pero el truco consiste en elegir las herramientas adecuadas y aprender a utilizarlas de forma eficaz. La automatización de pruebas requiere una inversión cuidadosa y mejoras incrementales. En este capítulo, explicamos cómo puede aplicar valores y principios ágiles para lograr impulso al iniciar o mejorar sus esfuerzos de automatización.

UN ENFOQUE ÁGIL PARA LA AUTOMATIZACIÓN DE PRUEBAS

Aquí estás, leyendo este capítulo sobre cómo hacer que tu estrategia de automatización de pruebas funcione, tal vez esperando esa solución milagrosa o una respuesta a todas tus dudas. preguntas. Lamentamos decepcionarte, pero debemos decírtelo desde el principio. No hay bala de plata. No existe una respuesta única que funcione para todos los equipos. Pero no se desanime, porque tenemos algunas ideas para ayudarle a empezar.

Primero, sugerimos abordar sus problemas de automatización como lo haría con cualquier problema. Defina el problema que está tratando de resolver. Para ayudarte a descubrir eso Primero hablamos de algunos conceptos básicos de la automatización de pruebas y reintroducimos algunos términos.

Categorías de pruebas de automatización

En la Parte III, presentamos los cuadrantes de pruebas ágiles y hablamos de cada uno de ellos. cuadrante y el propósito de las pruebas en cada cuadrante. En esta sección, nosotros Mire los cuadrantes bajo una luz diferente. Miremos detenidamente los cuadrantes. (ver Figura 14-1).

Puedes ver que hemos etiquetado ambos cuadrantes que apoyan al equipo (Q1 y Q2) como el uso de la automatización. En el Cuadrante 4, las herramientas utilizadas para criticar la producto desde el punto de vista tecnológico también suelen requerir herramientas automatizadas. En el Capítulo 9, "Kit de herramientas para pruebas empresariales que respalden al equipo", explicamos discutimos algunas de las herramientas que se pueden utilizar para automatizar las operaciones de cara al negocio. pruebas en la búsqueda del apoyo al equipo. De hecho, el único cuadrante que no lo es La etiqueta que utiliza la automatización es el Cuadrante 3: las pruebas comerciales que critican el producto. Sin embargo, como comentamos en el Capítulo 10, "Recursos de cara al negocio". Pruebas que critican el producto", las herramientas pueden ser útiles para algunas de esas pruebas. Por ejemplo, la automatización puede ayudar a configurar datos de prueba y escenarios de usuario, y analizar la actividad registrada.

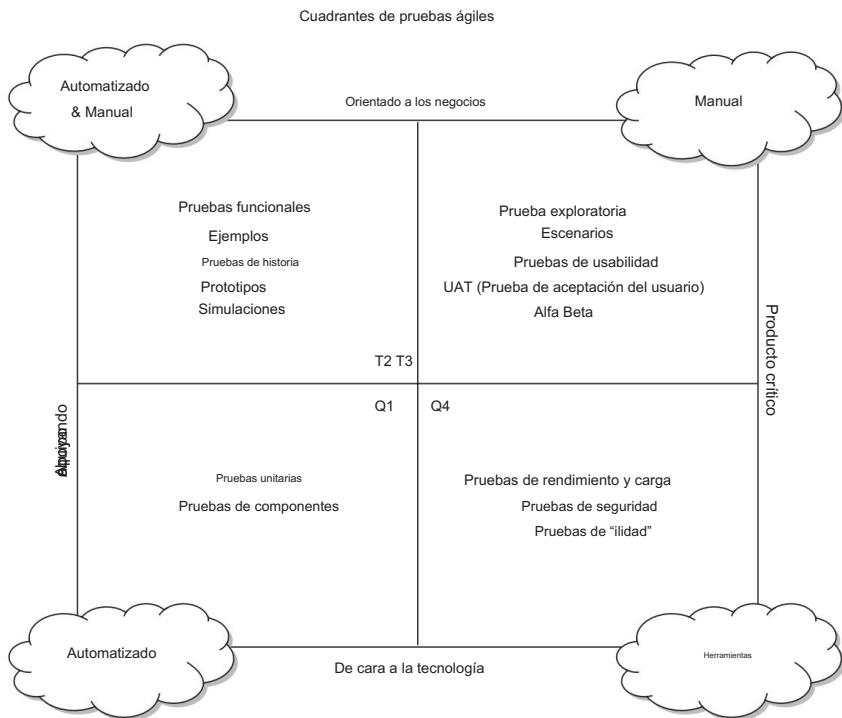


Figura 14-1 Cuadrantes de pruebas ágiles

Consulte el Capítulo 8, "Pruebas empresariales que respaldan al equipo", para obtener más información sobre las pruebas del Mago de Oz.

Utilice los cuadrantes para ayudarle a identificar los diferentes tipos de herramientas de automatización que podría necesitar para cada proyecto, incluso para cada iteración. Nos resulta útil revisar cada cuadrante y hacer una lista de verificación de las herramientas que podrían ser necesarias. Digamos que estamos a punto de rediseñar una interfaz de usuario. Nos fijamos en el cuadrante 1. ¿Cómo se puede codificar primero la prueba? ¿Sabemos cómo realizar una prueba unitaria de nuestra capa de presentación? ¿Necesitamos una nueva herramienta para ayudar con eso? Pasemos ahora al cuadrante 2.

Necesitaremos hacer algunos prototipos; ¿Deberíamos simplemente usar papel o deberíamos planificar una actividad tipo Mago de Oz? ¿Qué herramienta utilizaremos para crear pruebas ejecutables orientadas al negocio para guiar el desarrollo? ¿Tenemos scripts de prueba de regresión que necesitarán actualizarse o reemplazarse? Sabemos que una de nuestras actividades del Cuadrante 3 serán las pruebas de usabilidad. Eso requiere cierta planificación previa. Es posible que queramos herramientas que ayuden a rastrear las actividades de los usuarios para que podamos analizarlas más a fondo. Al pensar en el cuadrante 4, nos damos cuenta de que tenemos scripts de prueba de carga que utilizan la interfaz de usuario anterior, por lo que tenemos que dedicar tiempo a actualizarlos para la nueva.

Como enfatizamos en la Parte III, "Uso de los cuadrantes de pruebas ágiles", el orden de cuadrantes no se relaciona con el orden en el que hacemos las pruebas. Como nosotros Al hacer nuestra lista de verificación de herramientas necesarias para cada tipo de prueba, pensamos en cuándo Queremos probar para saber cuándo tener listas nuestras herramientas de automatización. Por ejemplo, un equipo que diseña una nueva arquitectura planearía hacer un pico y ejecutar prueba de escalabilidad lo antes posible. Necesitarán dedicar tiempo durante la primera iteración del proyecto, encontrar e implementar una herramienta de prueba de rendimiento.

Los cuadrantes nos ayudan a determinar qué herramientas podríamos necesitar, pero con tantas diferentes opciones de automatización en diferentes niveles, una estrategia sobre dónde hacer qué tipos de pruebas y cómo organizarlas es esencial. Para entregar valor Rápida y frecuentemente, nuestros esfuerzos de automatización necesitan un alto retorno de la inversión. La pirámide de pruebas nos ayuda a optimizar nuestra inversión en pruebas.

Pirámide de automatización de pruebas

La Figura 14-2 ilustra la "pirámide de automatización de pruebas". Nos gusta la versión que presentó Mike Cohn, que muestra la capa de base compuesta por Pruebas de unidades y componentes orientadas a la tecnología. Reconocemos que muchos equipos lucharán con esta idea, porque parece lo contrario de lo que muchos equipos tienen actualmente. A muchos equipos de prueba se les ha enseñado el modelo "V" de pruebas, donde se realizan actividades como pruebas de componentes, sistemas y versiones. hecho en secuencia después de las actividades de codificación. Otros equipos tienen una pirámide invertida, con la mayoría de las pruebas en la capa funcional o de presentación.

La pirámide ágil de automatización de pruebas muestra tres capas diferentes de automatización. pruebas. El nivel más bajo es la base que sustenta todo el resto. es principalmente compuesto por pruebas unitarias robustas y pruebas de componentes, la tecnología orientada pruebas que apoyan al equipo. Esta capa representa la mayor parte de la información automatizada. pruebas. Generalmente están escritos en el mismo idioma que el sistema bajo prueba, utilizando la familia de herramientas xUnit. Después de que un equipo haya dominado el arte de TDD, Estas pruebas son, con diferencia, las más rápidas y menos costosas de redactar. Ellos proveen la retroalimentación más rápida también, lo que los hace muy valiosos. Tienen, con diferencia, el mayor retorno de la inversión (ROI) de cualquier tipo de prueba.

En el desarrollo ágil, intentamos enviar tantas pruebas como sea posible a esta capa.

Si bien las pruebas empresariales tienden a realizarse en uno de los niveles superiores, las implementamos a nivel de unidad cuando tiene sentido. Si están probando a los clientes.

No es necesario saber leer y se pueden codificar mucho más rápidamente.

pruebas unitarias, es una buena opción. Otros tipos de pruebas orientadas a la tecnología, como

Las pruebas de desempeño también pueden ser posibles a nivel de unidad.

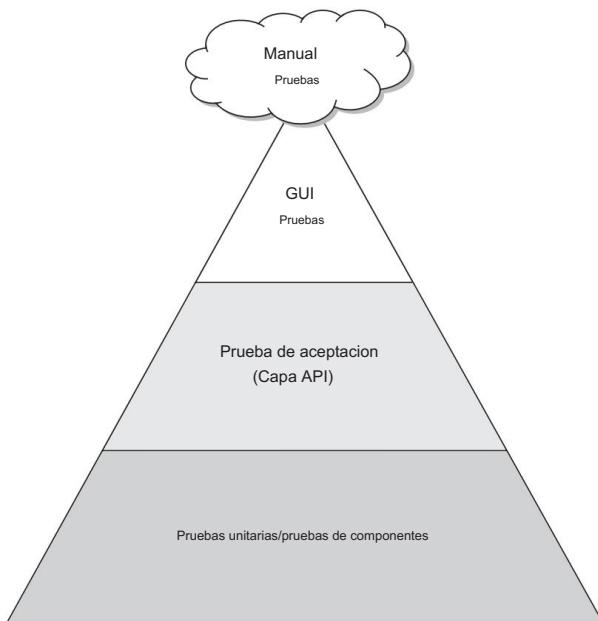


Figura 14-2 Pirámide de automatización de pruebas

Ver Capítulo 8,
"De cara al negocio
Pruebas que respaldan al equipo", para obtener más información sobre pruebas orientadas a los negocios que apoyan al equipo.

El nivel medio de la pirámide es la capa que incluye la mayoría de las pruebas empresariales automatizadas escritas para respaldar al equipo. Éstas son las pruebas funcionales que verifican que estamos "construyendo lo correcto". Las pruebas en este nivel pueden incluir pruebas de "historia", pruebas de "aceptación" y pruebas que cubren pruebas más amplias. conjuntos de funcionalidades que la capa de prueba unitaria. Estas pruebas operan en la API. nivel o "detrás de la GUI", probando la funcionalidad directamente sin tener que ir a través de la GUI. Escribimos casos de prueba que configuran entradas y accesorios que alimentan las entradas en el código de producción, aceptar las salidas y compararlas a los resultados esperados. Debido a que estas pruebas pasan por alto la capa de presentación, son menos costoso de escribir y mantener que las pruebas que utilizan la interfaz.

Intentamos escribirlos en un lenguaje específico de dominio que los clientes puedan entender, por lo que requieren más trabajo que las pruebas de nivel unitario. Ellos también generalmente ejecutar más lentamente, porque cada prueba cubre más terreno que una prueba unitaria y puede acceder a la base de datos u otros componentes. La retroalimentación que brindan es No es tan rápido como las pruebas a nivel unitario, pero aún así es mucho más rápido de lo que podríamos obtener. operando a través de la interfaz de usuario. Por lo tanto, su ROI no es tan alto como las pruebas que forman la base de la pirámide, pero es más alta que la capa superior.

tenemos mas
sobre estas pruebas

en el Capítulo 8,

“De cara al negocio

Pruebas que respaldan
al equipo” y

Capítulo 9, “Kit de
herramientas para empresas

Pruebas que respaldan
al equipo”, donde
analizamos la

situación empresarial
pruebas que apoyan
al equipo y al
herramientas que
capturar activamente
estas pruebas.

Fit y FitNesse son ejemplos de herramientas utilizadas para la capa media de la pirámide. También son comunes los instrumentos de prueba locales que utilizan hojas de cálculo u otros medios amigables para las empresas para definir casos de prueba.

El nivel superior representa lo que debería ser el esfuerzo de automatización más pequeño, porque las pruebas generalmente proporcionan el retorno de la inversión más bajo. Estas pruebas son las que hecho a través de la GUI, los que realmente operan y manipulan el Capa de presentación. Se escriben después de que se completa el código, al igual que Generalmente se escribe para criticar el producto e ir directamente a la suite de regresión.

Estas pruebas son tradicionalmente más caras de escribir, aunque hay nuevas herramientas que ayudan a reducir la inversión necesaria. Porque los componentes del La interfaz de usuario tiende a cambiarse con frecuencia, estas pruebas son mucho más frágiles. que las pruebas que funcionan a nivel funcional o unitario. Por ejemplo, simplemente cambiando el nombre Los elementos HTML pueden provocar que falle un script de prueba. Operando a través del usuario La interfaz también ralentiza estas pruebas, en comparación con las pruebas en los niveles más bajos de la pirámide que opera directamente sobre el código de producción. Las pruebas en la cima La capa proporciona comentarios importantes, pero un conjunto de pruebas de GUI puede llevar horas para ejecutarse en lugar de los pocos minutos necesarios para los conjuntos de pruebas a nivel de unidad. Nosotros desea minimizar el número de pruebas en esta capa, por lo que solo deben formar la punta de la pirámide.

No importa cuántas pruebas automatizadas tengan, la mayoría de los sistemas también necesitan actividades de prueba manuales, como pruebas exploratorias y pruebas de aceptación del usuario. No queremos olvidarnos de estos, por eso los hemos ilustrado con la pequeña nube en la punta de la pirámide. La mayor parte de nuestras pruebas de regresión deben automatizarse o nuestras pruebas manuales tampoco nos darán un buen retorno de la inversión.

Patrick Wilson-Welsh [2008] añade una dimensión descriptiva a la pirámide de automatización de pruebas con una metáfora de los “tres cerditos”. La base inferior La capa está hecha de ladrillos. Las pruebas son sólidas y no vulnerables a los resoplidos. y resoplidos del lobo feroz. La capa intermedia está hecha de palos. Ellos Necesita reorganizarse con más frecuencia que la capa de ladrillos para mantenerse fuerte. las pruebas en la capa superior está hecha de paja. Es difícil lograr que permanezcan en su lugar y el El lobo puede volarlos fácilmente. Si tenemos demasiadas pruebas hechas de paja, vamos a dedicar mucho tiempo a ponerlos nuevamente en forma.

La mayoría de los nuevos equipos ágiles no comienzan con esta forma de pirámide; generalmente está invertida, un resto de proyectos anteriores. Las herramientas de prueba de GUI suelen ser más fáciles de aprender, por lo que los equipos comienzan con muchas pruebas en su capa superior. Como nosotros Como se mencionó en el capítulo anterior, el “dolor” que la mayoría de los programadores tienen que superar para dominar la automatización de pruebas unitarias significa que el equipo



Consulte la bibliografía para obtener un enlace al análisis de Patrick Wilson-Welsh sobre "invertir la pirámide de pruebas".

Puede comenzar con sólo unos pocos ladrillos. Las luminarias que automatizan funcionales Las pruebas en la capa intermedia son fáciles de escribir si el sistema está diseñado con esas pruebas en mente, por lo que los palos podrían acumularse más rápido que los ladrillos. A medida que los equipos dominan TDD y la automatización de pruebas unitarias, la capa inferior comienza a crecer. Cuando ellos obtienen tracción, un equipo que utilice TDD construirá rápidamente la base de ladrillos de La pirámide de prueba.

La pirámide de pruebas es un buen lugar para empezar a ver cómo funciona la automatización de pruebas. puede ayudar a un equipo ágil. Los programadores tienden a centrarse en la base de la pirámide y necesitan mucho tiempo y capacitación para superar la "joroba de la pirámide". dolor" y llegar al punto en el que TDD es natural y rápido. En tradicional equipos, los evaluadores generalmente no tienen más remedio que automatizar las pruebas a nivel de GUI. El enfoque de equipo completo utilizado por los equipos ágiles significa que los evaluadores se asocian con programadores y ayudarlos a mejorar en la redacción de pruebas, lo que a su vez solidifica esa capa de ladrillos de la pirámide. Debido a que las pruebas impulsan el desarrollo, todo el equipo siempre está diseñando para lograr la máxima capacidad de prueba y el La pirámide puede crecer hasta alcanzar la forma correcta.

Los programadores se asocian con evaluadores para automatizar pruebas de nivel funcional, completando la capa media. Por ejemplo, un evaluador y un cliente pueden preparar un panel de 400 filas. Hoja de cálculo de casos de prueba para una aplicación de servicios web. El programador puede ayudar a encontrar una manera de automatizar esas pruebas. Diferentes miembros del equipo pueden tener experiencia en áreas como la generación de datos de prueba o el uso de herramientas como macros de Excel, y todo ese conocimiento se difunde por todo el equipo. Trabajando juntos, el equipo encuentra las mejores combinaciones de herramientas, casos de prueba y datos de prueba.

Involucrar a los programadores en la búsqueda de formas rentables de automatizar el Las pruebas de GUI de alto nivel tienen múltiples beneficios. Estos esfuerzos pueden brindar a los programadores una mejor comprensión del "panorama general" del sistema y los evaluadores pueden aprender cómo crear pruebas de GUI más flexibles y menos parecidas a pajitas.

Cuanto más pueda trabajar un equipo en conjunto y compartir conocimientos, más fuerte será equipo, la aplicación y las pruebas se convertirán. El lobo feroz no lo hará tener una oportunidad. Comencemos viendo qué tipo de pruebas podemos automatizar. y luego en lo que ni siquiera deberíamos intentar.

¿QUÉ PODEMOS AUTOMATIZAR ?

La mayoría de los tipos de pruebas que pueda imaginar se benefician de la automatización. unidad manual Las pruebas no ayudan mucho a prevenir fallas de regresión, porque realizar una Un conjunto de pruebas manuales antes de cada check-in simplemente no es práctico. Tampoco se puede diseñar código de prueba primero mediante pruebas unitarias manuales. Cuando los programadores

no pueden ejecutar pruebas rápidamente con solo tocar un botón, es posible que no estén motivados suficiente para ejecutar pruebas. Podríamos probar manualmente que diferentes unidades de código funcionan juntos correctamente, pero las pruebas automatizadas de componentes son una red de seguridad mucho más eficaz.

Las pruebas exploratorias manuales son una forma eficaz de encontrar defectos funcionales, pero Si no tenemos suficientes pruebas de regresión automatizadas orientadas al negocio, probablemente pasamos todo nuestro tiempo tratando locamente de mantenernos al día con la regresión manual. pruebas. Hablemos de los diferentes tipos de pruebas que se pueden realizar. bien con la automatización.

Para ejecutar pruebas automatizadas, necesita algún tipo de marco automatizado que permite a los programadores verificar el código con frecuencia, ejecutar pruebas en ese código y crear archivos desplegables. Consideremos esto primero.

Integración, compilaciones e implementaciones continuas

Consulte el Capítulo 7, "Pruebas tecnológicas que respaldan al equipo". para ver ejemplos de herramientas de automatización de compilación.

Cualquier tarea tediosa o repetitiva involucrada en el desarrollo de software es candidata para la automatización. Hemos hablado de la importancia de una construcción automatizada. proceso. No puedes construir tu pirámide de prueba automatizada sin esto. Su equipo necesita la retroalimentación inmediata de las pruebas a nivel de unidad para mantener el rumbo. Recibir correos electrónicos de compilación automatizados que enumeren todos los cambios registrados es de gran ayuda a los evaluadores porque saben cuándo una compilación está lista para probar sin tener que hacerlo. Molestar a los programadores.

Peligro: esperando la construcción del martes

En un entorno tradicional, es normal que los evaluadores esperen una compilación estable, incluso si eso significa esperar hasta el próximo martes. En un entorno ágil, si los evaluadores no siguen el ritmo de los desarrolladores, las historias se prueban más tarde en el juego. Si los desarrolladores no reciben comentarios, como sugerencias y errores, los evaluadores pueden perder credibilidad ante los desarrolladores. Los errores no se descubrirán hasta que los desarrolladores ya estén en otra historia y no quieran que los interrumpan para solucionarlos hasta más tarde.

Los errores se acumulan y la automatización se ve afectada porque no se puede completar. La velocidad se ve afectada porque una historia no se puede marcar como "terminada" hasta que se prueba. Esto hace que sea más difícil planificar la próxima iteración. Al final del ciclo de lanzamiento, las pruebas de tu historia llegan hasta el final del juego y es posible que no tengas un lanzamiento exitoso. Como mínimo, tendrás una liberación estresante.

Un proceso de implementación automatizado también acelera las pruebas y reduce los errores. De hecho, el día que Janet estaba editando este capítulo, arruinó el despliegue. porque fue un proceso manual. Era bastante simple, pero ella era nueva en el proyecto y movió el archivo al lugar equivocado. Implementar un proceso de implementación automatizado estaba en la lista de cosas que Janet debía hacer de inmediato. El equipo de Lisa implementó su marco de integración y construcción continua. Lo primero y lo encontré bastante fácil y rápido de hacer, aunque requiere cuidados y alimentación continuos. Otros equipos, especialmente aquellos con grandes y complejos enfrentan obstáculos mucho mayores.

Hemos hablado con equipos que tuvieron tiempos de construcción de dos horas o más. Este significaba que un programador tendría que esperar dos horas después de registrarse código para obtener la validación de que su registro no rompió ninguna funcionalidad preexistente. Es mucho tiempo de espera.

La mayoría de los equipos ágiles encuentran que una construcción continua dura más de ocho a diez minutos para ser inviable. Incluso 15 minutos es demasiado tiempo para esperar comentarios, porque los registros comenzarán a acumularse y los evaluadores esperarán mucho tiempo para recibirlas. la última y mejor construcción. ¿Te imaginas cómo trabajan los desarrolladores con una construcción que lleva dos horas se siente a medida que se acerca el final de una iteración o ciclo de lanzamiento? Si rompen alguna funcionalidad tendrán que esperar dos más horas para saber si lo habían arreglado o no.

Muchas veces, las compilaciones largas son el resultado de acceder a la base de datos o intentar prueba a través de la interfaz. Miles de pruebas ejecutadas en una gran base de código pueden agotar los recursos de la máquina que ejecuta la compilación. Haga un perfil de sus pruebas y vea dónde está el cuello de botella. Por ejemplo, si es el acceso a la base de datos que está causando la mayoría de los problemas, intente burlarse del base de datos real y utilice una en memoria en su lugar. Configure el proceso de compilación para distribuir las pruebas en varias máquinas. Vea si otro software podría ayudar a gestionar mejor los recursos. Traiga expertos externos a su equipo para que le ayuden si es necesario.

La clave para acelerar un proceso continuo de integración y construcción es tomar pasito a pasito. Introduzca los cambios uno a la vez para que pueda mida cada éxito por separado y sepa que está en el camino correcto. Para comenzar con, es posible que desee simplemente eliminar las pruebas más costosas (en términos de tiempo) para ejecutarse todas las noches en lugar de en cada compilación.

Un proceso de integración y construcción continuo y rápido proporciona el mayor retorno de la inversión (ROI) de cualquier esfuerzo de automatización. Es lo primero que todo equipo necesita automatizar.

Consulte la bibliografía para obtener enlaces para crear automatización. herramientas y libros con más información sobre cómo mejorar el proceso de construcción.

Capítulo 7, "Tecnología-Enfrentando pruebas que Apoyen el Equipo", entra en detalles sobre algunos de las herramientas que puede ser usado.

Cuando esté implementado, el equipo tiene una manera de obtener comentarios rápidos de las pruebas automatizadas. A continuación, analizamos los diferentes tipos de pruebas que deberían automatizarse.

Pruebas unitarias y de componentes

No podemos exagerar la importancia de automatizar las pruebas unitarias. Si tu los programadores utilizan TDD como mecanismo para escribir sus pruebas, luego no sólo están creando un excelente conjunto de regresión, sino que también los están utilizando para diseñar código robusto y de alta calidad. Si su equipo no está automatizando pruebas unitarias, es Las posibilidades de éxito a largo plazo son escasas. Realice la automatización de pruebas a nivel unitario y la integración continua su primera prioridad.

Pruebas de API o servicios web

Probar una API o una aplicación de servicios web es más fácil utilizando alguna forma de automatización. Janet ha estado en equipos que han utilizado Ruby con éxito para leer en un hoja de cálculo con todas las permutaciones y combinaciones de variables de entrada y comparar los resultados con los resultados esperados almacenados en las hojas de cálculo. Estas pruebas basadas en datos son fáciles de escribir y mantener.

Un cliente de Janet utilizó la función IRB (Interactive Ruby Shell) de Ruby para Probar los servicios web para las pruebas de aceptación. El equipo estaba dispuesto a compartir su guiones con el equipo del cliente, pero los evaluadores de negocios prefirieron observar vea qué sucedió si las entradas se cambiaron sobre la marcha. La ejecución de pruebas de forma interactiva de forma semiautomática lo permitió.

Pruebas detrás de la GUI

Consulte el Capítulo 9, "Kit de herramientas para empresas Pruebas que respaldan al equipo", para ver ejemplos de herramientas específicas.

Probar detrás de la GUI es más fácil de automatizar que probar la GUI misma. Porque las pruebas no se ven afectadas por los cambios en la capa de presentación y funcionan. en un código de lógica empresarial más estable, son más estables. Herramientas para este tipo de Las pruebas generalmente implican escribir pruebas en un formato declarativo, utilizando tablas. u hojas de cálculo. Los dispositivos que obtienen el código de producción para operar en el Las entradas de prueba y la devolución de los resultados generalmente se pueden escribir rápidamente. Esto es un Área privilegiada para escribir pruebas orientadas al negocio, comprensibles tanto para los clientes como para los desarrolladores que impulsan el desarrollo.

Probando la GUI

Incluso es necesario probar una GUI delgada con poca o ninguna lógica empresarial. El rápido El ritmo de desarrollo ágil, que ofrece nuevas funciones en cada iteración, actualiza algunas pruebas de regresión automatizadas a nivel de GUI para la mayoría de los proyectos.

La selección de herramientas es clave para una automatización exitosa de la GUI. Los scripts automatizados debe ser flexible y fácil de mantener. Janet ha usado Ruby y Watir muy con éxito cuando el marco se desarrolló utilizando buenas prácticas de codificación, como si fuera una aplicación de producción. Se dedicó tiempo a desarrollar el bibliotecas para que no haya mucha reelaboración o duplicación en el código, y Los cambios necesarios se podrían realizar en un solo lugar. Hacer que el código fuera fácil de mantener aumentó el retorno de la inversión en estas pruebas.



Consulte el Capítulo 9, "Kit de herramientas para

Orientado a los negocios
Pruebas que respaldan al equipo", para ejemplos de marcos de prueba de GUI.

Un punto sobre la capacidad de prueba aquí: asegúrese de que los programadores nombren sus objetos o les asignen ID. Si dependen de identificadores generados por el sistema, entonces Cada vez que se agrega un nuevo objeto a la página, las ID cambiarán, lo que requerirá cambios en las pruebas.

Mantenga las pruebas solo en la interfaz real. Verifique cosas como asegurarse de que Los botones realmente funcionan y hacen lo que se supone que deben hacer. No intentes intentar probar funcionalidad empresarial. Otros tipos de pruebas que se pueden automatizar fácilmente son comprobadores de enlaces. No es necesario que alguien revise manualmente cada enlace en cada página para asegurarse de que lleguen a la página correcta. Busque la fruta más fácil, automative primero las cosas que son simples de automatizar y Tendrás más tiempo para los desafíos más grandes.

Pruebas de carga



Consulte el Capítulo 11, "Crítica del producto mediante pruebas tecnológicas", para ver ejemplos de herramientas de automatización de pruebas de carga.

Algunos tipos de pruebas no se pueden realizar sin automatización. Pruebas de carga manuales Por lo general, no son factibles ni precisos, aunque todos lo hemos intentado alguna vez. otro. Las pruebas de rendimiento requieren tanto herramientas de monitoreo como una manera de impulsar acciones en el sistema bajo prueba. No se puede generar un ataque de gran volumen para verificar si un sitio web puede ser pirateado o si puede soportar una gran carga. sin algún marco de herramientas.

Comparaciones



Lea más sobre las herramientas de administración de código fuente y los IDE en el Capítulo 7, "Pruebas tecnológicas que respaldan al equipo".

Verificar visualmente la salida de un archivo ASCII mediante un proceso del sistema es mucho más fácil si Primero analiza el archivo y lo muestra en un formato legible por humanos. Un guión para comparar archivos de salida para asegurarse de que no se realizaron cambios involuntarios es una mucho más rápido y preciso que intentar compararlos manualmente. Archivo Abundan las herramientas de comparación, que van desde diferencias gratuitas hasta herramientas propietarias. como WinDiff. Las herramientas de gestión de código fuente y los IDE tienen los suyos propios. herramientas de comparación integradas. Estos son elementos esenciales en la caja de herramientas de todo evaluador. No se olvide de crear scripts para comparar tablas de bases de datos cuando realice pruebas para su almacén de datos o proyectos de migración de datos.

Tareas repetitivas

Mientras trabajamos con nuestros clientes para comprender mejor el negocio y aprender lo que es valioso para ellos, podríamos ver oportunidades para automatizar algunas de sus tareas. La empresa de Lisa necesitaba enviar varios formularios con una carta de presentación a todos sus clientes. Los programadores no sólo pudieron generar los formularios sino También podría concatenarlos con la carta de presentación y acelerar enormemente el proceso. esfuerzo de correo. El compañero de pruebas de Lisa, Mike Busse, escribió una macro de hoja de cálculo para hacer cálculos complejos para asignar fondos que los administradores del plan de jubilación habían estado haciendo manualmente. Se pueden reemplazar muchas listas de verificación manuales con un script automatizado. La automatización no es sólo para realizar pruebas.

Creación o configuración de datos

Otra área útil para la automatización es la creación o configuración de datos. Si estás configurando tus datos constantemente, automatiza el proceso. A menudo necesitamos repetir algo varias veces para poder recrear un error. Si eso se puede automatizar, se le garantizará que obtendrá los mismos resultados cada vez.

La historia de Lisa

Muchos de nuestros esquemas de prueba, incluidos los utilizados para conjuntos de regresión automatizados, utilizan datos canónicos. Estos datos canónicos o "semillas" se tomaron originalmente de la producción. Algunas tablas de la base de datos, como las tablas de búsqueda, no cambian, por lo que nunca es necesario actualizarlas con una copia nueva. Otras tablas, como las que contienen información sobre planes de jubilación, empleados y transacciones, deben comenzar desde la Zona Cero cada vez que se ejecuta un conjunto de regresión.

Nuestro desarrollador de bases de datos escribió un procedimiento almacenado para actualizar cada esquema de prueba a partir del esquema "semilla". Los evaluadores podemos especificar las tablas que queremos actualizar en una tabla especial llamada REFRESH_TABLE_LIST. Tenemos un objetivo ant para cada esquema de prueba para ejecutar el procedimiento almacenado que actualiza los datos. Las compilaciones automatizadas utilizan este objetivo, pero lo utilizamos nosotros mismos siempre que queremos limpiar nuestro esquema de prueba y empezar de nuevo.

Muchas de nuestras pruebas de regresión crean sus propios datos además de los datos "semilla". Nuestras pruebas Watir crean todos los datos que necesitan e incluyen una lógica que las hace volver a ejecutar sin importar qué datos estén presentes. Por ejemplo, el script que prueba a un empleado que solicita un préstamo de su plan de jubilación primero cancela cualquier préstamo existente para poder obtener uno nuevo.

Las pruebas de FitNesse que prueban la capa de la base de datos también crean sus propios datos. Usamos un esquema especial en el que hemos eliminado la mayoría de las restricciones, por lo que no tenemos que agregar todas las columnas de cada tabla. Las pruebas solo agregan los datos pertinentes a la funcionalidad que se está probando. Cada prueba elimina los datos que creó, por lo que las pruebas posteriores no se ven afectadas y cada prueba es independiente y se puede volver a ejecutar.

—Lisa

Limpiar los datos de prueba es tan importante como generarlos. Tu creación de datos El kit de herramientas debe incluir formas de derribar los datos de la prueba para que no afecten a un prueba diferente o evitar volver a ejecutar la misma prueba.

Hemos analizado áreas importantes donde la automatización es necesaria o al menos útil.

Nuestra opinión es que siempre que necesites hacer una prueba o alguna prueba relacionada actividad, primero decida si puede ser ayudada por la automatización. En algunos casos, la automatización no será apropiada. Veamos algunos de ellos.

¿QUÉ NO DEBEMOS AUTOMATIZAR ?

Algunas pruebas necesitan ojos, oídos e inteligencia humanos. Las pruebas de usabilidad y exploratorias son dos que caen en esa categoría. Otras pruebas que pueden no justificar la inversión en automatización son las pruebas únicas y las que nunca fallarán.

Pruebas de usabilidad

Analizamos algunas herramientas de registro y monitoreo en el Capítulo 10, "Presentación empresarial". Pruebas que critican el producto".

Las pruebas de usabilidad reales requieren que alguien utilice realmente el software. La automatización podría ser útil para configurar escenarios para examinar posteriormente nuestra capacidad. Observar a los usuarios en acción, informarles sobre sus experiencias y Juzgar los resultados es un trabajo para una persona que entiende que los aspectos de usabilidad del software no pueden automatizarse. Registrar las acciones del usuario es útil para pruebas de usabilidad.

La historia de Janet

Habíamos evaluado varias herramientas GUI pero decidimos usar Ruby con Watir. Mantuvimos nuestras pruebas limitadas únicamente a las funciones de la GUI. Una de nuestras pruebas fue verificar que se mostraran los mensajes de validación correctos en la pantalla. Estaba ejecutando las pruebas y casualmente estaba mirando la pantalla porque no había visto esta prueba en particular que creó uno de los otros evaluadores. Mis ojos captaron algo extraño, pero la prueba pasó, así que lo volví a reproducir. Uno de los programadores había agregado un "\$" a la pantalla y el mensaje de error se mostró desplazado debido a ello. Se mostró el mensaje correcto, pero no en el lugar correcto. En este caso, el valor de ver las pruebas fue enorme porque nos estábamos preparando para lanzarlas bastante pronto y probablemente no habríamos detectado ese problema en particular.

—Janet

Es posible automatizar pruebas que aseguren que la GUI nunca cambie, pero usted Necesitas preguntarte si vale la pena el costo. ¿Realmente te importa que un botón haya cambiado de posición un píxel? ¿Los resultados justifican el esfuerzo? Nosotros No creo que debas automatizar las pruebas de "apariencia", porque una prueba automatizada

El script sólo puede buscar lo que usted le dice que vea. La automatización perdería lo visual problemas que saltarían a la vista de un humano.

Prueba exploratoria

Consulte el Capítulo 10,
"Presentación empresarial
Pruebas que critican el
producto", para obtener
más información sobre
las pruebas exploratorias
y las herramientas que
pueden facilitarlas.

De manera similar, las pruebas exploratorias pueden acelerarse con scripts para crear pruebas. datos y saltar a través de algunos pasos de configuración, pero requiere un probador capacitado para diseñar y ejecutar las pruebas. Uno de los principales objetivos de las pruebas exploratorias es aprender más sobre el producto haciendo, y luego usar esa información para mejorar desarrollo futuro. Los scripts automatizados no harán eso por usted. Sin embargo, como Como hemos dicho antes, no tendrás tiempo para realizar pruebas exploratorias sin mucha otras pruebas automatizadas.

Pruebas que nunca fallarán

Hemos escuchado el argumento de que las pruebas que nunca fallarán no necesitan ser automatizadas. Si un requisito es tan obvio que sólo hay una forma de implementarlo y ningún programador mirará jamás ese código más tarde sin saber exactamente qué debe hacer, las posibilidades de que alguien introduzca un defecto en ese El código es casi nada. Digamos que tenemos un formulario con campos de dirección. Hacemos Necesita una prueba de regresión automatizada para verificar que la segunda línea de dirección ¿no es requerido? Después de haberlo verificado manualmente, ¿qué probabilidad hay de que alguien ¿Lo cambiará accidentalmente a un campo obligatorio más adelante? Incluso si alguien lo hiciera, No sería un evento catastrófico. Alguien más lo notaría y la gente Podría solucionarlo fácilmente hasta que se solucionara.

Por otra parte, sería fácil incluir una prueba. Y trucos de programador como los errores de copiar/pegar ocurren todo el tiempo. Si te sientes cómodo con eso La prueba manual única hace el trabajo y que el riesgo de fallas futuras no justifica la automatización de las pruebas de regresión, no las automatice. Si tu decisión resulta ser incorrecta, tendrás otra oportunidad de automatizarlas. más tarde. Si no está seguro y no es muy difícil automatizarlo, hágalo.

Consulte el Capítulo 18,
"Codificación y pruebas",
para obtener más
información sobre el análisis
de riesgos y su relación con
las pruebas.

Si está probando un sistema crítico para la vida, incluso existe un riesgo muy pequeño de regresión. el fracaso es demasiado. Utilice el análisis de riesgos para ayudar a decidir qué pruebas deben realizarse. automatizado.

Pruebas únicas

La mayoría de las veces, basta con ejecutar manualmente una prueba única. Si automatiza una prueba no tiene recompensa, ¿por qué hacerlo? A veces vale la pena realizar la automatización por un prueba única.

La historia de Lisa

Recientemente hicimos una historia para mostrar un cuadro de diálogo con un mensaje de advertencia al publicar una nómina, pero el mensaje solo debería aparecer durante las dos primeras semanas de enero. Automatizar una prueba para esta funcionalidad requeriría alguna forma de simular que la fecha actual era entre el 1 y el 15 de enero. Eso no es muy difícil de hacer, pero las consecuencias de una falla fueron bastante triviales y teníamos más historias críticas que contar. Entregar esa iteración. Automatizar esa prueba en ese momento simplemente no tenían suficiente valor para justificar el costo y el factor de riesgo era bajo. Decidimos probarlo manualmente.

Hay otros casos en los que realizar una prueba única parece lo más intuitivo, pero la automatización es una mejor opción. Alojamos sitios para diferentes socios comerciales y cada uno tiene contenido, apariencia y funcionalidad únicos. Los valores de la base de datos impulsan el comportamiento y el contenido correctos para cada marca. Algunos de los datos, como las tablas de tarifas basadas en el valor de los activos y el número de participantes, son muy complejos. Es mucho más fácil y preciso verificar estos datos mediante las pruebas de FitNesse. Tenemos un conjunto de dispositivos que nos permiten especificar claves para la "marca" del socio que queremos probar. Podemos ingresar fácilmente los resultados esperados apropiados de las hojas de cálculo que el personal de desarrollo comercial crea para cada nuevo socio.

Estas pruebas no forman parte de nuestro conjunto de regresión. Se utilizan una sola vez para validar la nueva marca.

—Lisa

Puede que valga la pena automatizar tareas tediosas, incluso si no las haces con frecuencia. Sopese el costo de la automatización con la cantidad de tiempo valioso consumido por haciendo la prueba manualmente. Si es fácil de hacer manualmente y la automatización no Sea rápido, solo manténgalo manual.

¿QUÉ PODRÍA SER DIFÍCIL DE AUTOMATIZAR?

Cuando el código no se escribe primero como prueba, o al menos teniendo en cuenta la automatización de pruebas, es mucho más difícil de automatizar. Los sistemas más antiguos tienden a caer en esta categoría, pero sin duda hay mucho código nuevo con las mismas características no comprobables. todavía en producción.

Si te enfrentas a trabajar en código existente que aún no tiene pruebas automatizadas, te espera una batalla cuesta arriba, pero que se puede ganar. Código heredado puede tener E/S, acceso a bases de datos, así como lógica de negocios y código de presentación entrelazados. Puede que no esté claro dónde conectar el código para automatizar un prueba. ¿Cómo se empieza a automatizar pruebas en un sistema de este tipo? ciertamente tu No puedo planear automatizar todo lo que está debajo de la GUI, porque gran parte de la La lógica está en la capa de presentación.

Hay al menos un par de enfoques diferentes que funcionan bien. El "joroba de dolor" de la que hablamos en el Capítulo 13, "Por qué queremos automatizar Las pruebas y lo que nos frena", es intimidante, pero se puede superar y entonces la automatización de pruebas será mucho más fácil. El trabajo de Michael Feathers Efectivamente con código heredado [2004] explica cómo construir un armés de prueba alrededor de bases de código existentes y refactorizarlas para adaptarse a la automatización. Incluso con código heredado, puede escribir pruebas para protegerse contra la introducción nuevos problemas. Este enfoque puede funcionar incluso en sistemas que carecen de estructura. o no están orientados a objetos.

Capítulo 7,
"Pruebas
tecnológicas que
respaldan al equipo",
entra en más
detalles sobre
los diferentes
enfoques ágiles
del código heredado.

El equipo de Lisa optó por un enfoque diferente pero igualmente eficaz. El equipo Los miembros comenzaron a "estrangular" el código heredado escribiendo todas las funciones nuevas en un Nueva arquitectura fácil de probar. Están reemplazando gradualmente todo el código antiguo. con código escrito primero como prueba. Cuando trabajan en código antiguo para corregir errores, o en En los casos en los que el código antiguo necesita actualización, simplemente agregan pruebas unitarias para todos del código que cambian. Un conjunto de pruebas de humo GUI cubre las funciones críticas del resto del sistema heredado que no tiene pruebas unitarias.

Al igual que con cualquier proyecto de automatización, acérquese al código difícil de automatizar. pieza por pieza y abordar primero las áreas de mayor riesgo. Resuelve la comprobabilidad problema y encontrar una manera de escribir pruebas a nivel unitario. El esfuerzo dará sus frutos.

DESARROLLO DE UNA ESTRATEGIA DE AUTOMATIZACIÓN : DONDE ¿ EMPEZAMOS ?

Un enfoque simple, paso a paso, parece incompatible con una automatización. estrategia, pero en las pruebas ágiles intentamos comprender el problema primero. decidir dónde y cómo empezar con la automatización requiere un poco de reflexión y discusión. A medida que su equipo analiza los desafíos de las pruebas, deberá considerar dónde La automatización es apropiada. Antes de comenzar a buscar una herramienta de automatización en particular, querrá identificar sus requisitos.

Debe comprender qué problema está tratando de resolver. Qué vas a tratando de automatizar? Por ejemplo, si no tiene ningún tipo de automatización de pruebas, y comienza comprando una costosa herramienta de prueba comercial pensando que automatizará todas sus pruebas funcionales, es posible que esté comenzando en el lugar equivocado.

Le sugerimos que comience por el principio. Busque su mayor ganancia. el mas grande La mejor inversión son definitivamente las pruebas unitarias que los programadores pueden realizar. En lugar de comenzar en la cima de la pirámide de prueba, es posible que desee comenzar en la abajo, asegurándose de que los conceptos básicos estén en su lugar. También es necesario considerar

los diferentes tipos de pruebas que necesita automatizar y cuándo deberá hacerlo
Tener herramientas listas para usar.

En esta sección, asumimos que ha implementado pruebas automatizadas de unidades y componentes del Cuadrante 1 y que está buscando automatizar sus pruebas comerciales en Los cuadrantes 2 y 3, o las pruebas tecnológicas del cuadrante 4 que critican el producto. Le ayudaremos a diseñar una buena estrategia para desarrollar sus recursos de automatización.

Piense en las habilidades y la experiencia de su equipo. ¿Quién necesita la automatización y por qué? ¿Qué objetivos estás tratando de lograr? Entendiendo algunos de Estos problemas pueden afectar su elección de herramientas y el esfuerzo que dedica. Hay una sección sobre evaluación de herramientas al final de este capítulo.

La automatización da miedo, especialmente si empiezas desde cero, entonces, ¿dónde podemos comenzar?

¿Dónde duele más?

Para saber dónde centrar sus esfuerzos de automatización a continuación, pregúntele a su equipo: "¿Cuál es el área de mayor dolor?" o, para algunos equipos, "¿Cuál es el mejor Área de aburrimiento? ¿Puedes incluso implementar el código para probarlo? Hacer Los miembros del equipo se sienten seguros al cambiar el código, o les falta alguna ¿Red de seguridad de pruebas automatizadas? Tal vez los miembros de su equipo sean más avanzados, dominen TDD y tengan un conjunto completo de pruebas unitarias. Pero ellos no tiene un buen marco para especificar pruebas orientadas al negocio, o no puede conseguirlo un truco para automatizarlos. Quizás tengas algunas pruebas de GUI, pero son extremadamente lentos y su mantenimiento cuesta mucho.

Peligro: intentar probar todo manualmente

Si pasa todo su tiempo volviendo a probar funciones que ha probado antes, sin acceder a nuevas funciones y necesita agregar más y más pruebas, está sufriendo una grave falta de automatización de pruebas. Este peligro significa que los evaluadores no tienen tiempo para participar en las discusiones sobre diseño e implementación, los errores de regresión pueden pasar desapercibidos, las pruebas ya no pueden seguir el ritmo del desarrollo y los evaluadores se quedan estancados. Los desarrolladores no se involucran en las pruebas empresariales y los evaluadores no tienen tiempo para encontrar una mejor manera de resolver los problemas de las pruebas.

Su equipo puede solucionar este problema desarrollando una estrategia de automatización, como describimos en este capítulo. El equipo comienza a diseñar para la capacidad de prueba y elige e implementa herramientas de automatización apropiadas. Los evaluadores tienen la oportunidad de desarrollar sus habilidades técnicas.

Capítulo 18, "Codificación y pruebas", tiene más información sobre un enfoque simple para el análisis de riesgos.

Donde más duele, ese es el lugar para comenzar tus esfuerzos de automatización.

Por ejemplo, si su equipo tiene dificultades incluso para entregar código implementable, usted Es necesario implementar un proceso de construcción automatizado. No hay nada peor que jugar con los pulgares mientras esperas a que se pruebe algún código.

Pero, si el desempeño pone en peligro la existencia de su organización, las pruebas de desempeño deben ser la máxima prioridad. Se trata de volver a comprender lo que problema que estás tratando de resolver. El análisis de riesgos es tu amigo aquí.

La historia de Janet

Trabajé en un sistema heredado que intentaba abordar algunos problemas de calidad y agregar nuevas funciones para nuestro cliente principal. No había ninguna unidad automatizada ni pruebas funcionales para la aplicación existente, pero necesitábamos refactorizar el código para abordar los problemas de calidad. Los miembros del equipo decidieron abordarlo pieza por pieza. Cuando eligieron una parte de la funcionalidad para refactorizar, los programadores escribieron pruebas unitarias, se aseguraron de que pasaran y luego reescribieron el código hasta que las pruebas pasaron nuevamente. Al final de la refactorización, tenían un código bien escrito y comprobable y las pruebas correspondientes. Los evaluadores escribieron las pruebas funcionales de nivel superior al mismo tiempo. En un año, la mayor parte del código heredado de mala calidad se había reescrito y el equipo había logrado una buena cobertura de pruebas simplemente abordando un fragmento a la vez.

—Janet

La automatización de pruebas no dará sus frutos a menos que se implementen otras buenas prácticas de desarrollo. lugar. La integración continua ejecutando un conjunto sólido de pruebas unitarias es un primer paso hacia la automatización de otras pruebas. El código que se refactoriza continuamente para lograr mantenimiento y un buen diseño ayudará a aumentar el retorno de la inversión en la automatización. La refactorización no puede realizarse sin esa buena cobertura de pruebas unitarias. Estos desarrollos Las prácticas también deben aplicarse a los scripts de prueba funcionales automatizados.

Enfoque de múltiples capas

Si bien recomendamos dominar una herramienta a la vez, no espere demasiado de cualquier herramienta. Utilice la herramienta adecuada para cada necesidad. La herramienta que mejor funciona para pruebas unitarias puede o no ser apropiado para automatizar pruebas funcionales.

Las pruebas de GUI, carga, rendimiento y seguridad pueden requerir un proceso diferente. herramienta o herramientas.

El concepto de pirámide de pruebas de Mike Cohn (ver Figura 14-2) ha ayudado a nuestros equipos a poner sus esfuerzos de automatización donde hacen más bien. queremos maximizar las pruebas que tienen el mejor ROI. Si la arquitectura del sistema está diseñada para la capacidad de realizar pruebas, la automatización de las pruebas será menos costosa, especialmente a nivel unitario.

Las pruebas que pasan por la interfaz de usuario suelen tener el retorno de la inversión más bajo, porque Son costosos de mantener, pero los necesitamos. Ellos forman el pequeño punta de nuestra pirámide Podemos optar por automatizar algunas de estas pruebas, pero el La mayoría de las pruebas de GUI se definen en términos comerciales y probablemente sea mejor dejarlas. como pruebas de interacción humana (es decir, pruebas manuales).

La capa intermedia representa las pruebas funcionales que funcionan directamente con el código de producción, sin una GUI u otra capa intermedia. Si bien no son tan baratos de automatizar como las pruebas a nivel unitario y brindan retroalimentación un poco más lentamente, las herramientas adecuadas les permiten obtener un buen retorno de la inversión. El hecho de que estas pruebas puedan ser escritas en un lenguaje que los expertos en negocios entiendan aumenta su valor.

Hay muchas capas diferentes en la aplicación que se pueden probar de forma independiente. En su libro *xUnit Test Patterns* [2007], Gerard Meszaros se refiere a esto como patrón de prueba de capa. Advierte que al intentar probar todas las capas de la aplicación por separado, aún nos queda comprobar que las capas están enganchadas correctamente, y esto puede requerir al menos una prueba de la lógica empresarial a través de la capa de presentación.

La historia de Lisa

A medida que mi equipo construía nuestro marco de automatización paso a paso, reunimos un arsenal de herramientas. Después de implementar un marco de construcción continua con Ant y CruiseControl, dominamos JUnit para pruebas unitarias. Sabíamos que la automatización de pruebas unitarias es la forma más rápida y económica de automatizar, y proporciona la retroalimentación más rápida a los programadores.

Nuestro sistema heredado no tenía pruebas automatizadas, por lo que creamos un conjunto de pruebas de regresión GUI con Canoo WebTest. Esto proporcionó una buena recompensa porque los scripts de WebTest fueron especificados, no programados. Fueron rápidos de escribir y fáciles de mantener.

Después de implementar JUnit y WebTest, experimentamos con FitNesse y descubrimos que funcionaba bien para las pruebas funcionales detrás de la GUI. Descubrimos que la automatización con FitNesse es relativamente rápida. Aunque las pruebas de FitNesse son significativamente más caras de producir y mantener que las pruebas unitarias, su valor para impulsar el desarrollo y promover la colaboración entre clientes, programadores y evaluadores mantuvo alto el retorno de la inversión.

Todas estas herramientas fueron fáciles de aprender, implementar e integrar con el proceso de construcción y proporcionaron comentarios continuos sobre nuestros problemas de regresión. Fueron consideraciones importantes a la hora de decidir nuestra estrategia de automatización de pruebas.

—Lisa

Al evaluar la rentabilidad de sus esfuerzos de automatización, considere aspectos menos tangibles, como si la herramienta promovió la colaboración entre los

equipos técnicos y de atención al cliente. Una razón principal para escribir exámenes es ayudar desarrollo de guías. Si el proceso de redacción de sus pruebas de aceptación automatizadas resulta en una comprensión profunda de los requisitos del negocio, eso es suficiente de recuperación, incluso si las pruebas nunca encuentran un solo error de regresión más adelante.

Piense en el diseño y el mantenimiento de las pruebas

Piensa en todos los guiones de prueba manuales que has escrito en tu vida. No
¿Solo desearías que todo eso hubiera sido automatizado? ¿No habría tu vida
¿Ha sido mucho más fácil? Creemos que todas las pruebas programadas deben automatizarse. vamos comience a convertir esas pruebas escritas manualmente.

Una vez que haya comenzado, puede resultar bastante fácil automatizar las pruebas. Por ejemplo, cuando Si tiene un dispositivo FitNesse en funcionamiento, agregar más casos de prueba requiere poco esfuerzo. Esto es genial cuando tienes muchas permutaciones diferentes para probar. usted Probablemente pruebe más condiciones de las que probaría si se hubieran realizado todas las pruebas. a mano. Cuando los miembros del equipo de Lisa reescribieron el sistema de préstamos de su plan de jubilación, pudieron probar cientos de posibilidades diferentes para el procesamiento de pagos de préstamos mediante pruebas de FitNesse. ¿Qué sucede cuando se realizan tres pagos del préstamo? ¿Procesado el mismo día? Si alguien no hace ningún pago durante tres meses y luego envía un pago grande, ¿se calcula y aplica correctamente el interés? Fue fácil escribir pruebas automatizadas para descubrirlo.

Esa es una gran ventaja, pero tiene un inconveniente. Ahora el equipo tiene docenas, o incluso cientos de casos de prueba que mantener. ¿Qué pasa si las reglas sobre el cálculo? ¿El monto de interés para el pago de un préstamo cambiará dentro de un año? Este podría requerir la actualización de cada prueba. Si su herramienta de prueba no permite realizar cambios fácilmente en las pruebas existentes, su gran conjunto de pruebas automatizadas puede convertirse en un dolor de cabeza.

Las pruebas de un extremo a otro son particularmente difíciles de automatizar porque tienen la Es más probable que necesite mantenimiento a medida que cambian las reglas comerciales. Cómo podemos ¿Equilibrar la necesidad de automatización con el costo?

Diseño de prueba

Capítulo 8,
"Presentación empresarial
Pruebas que respaldan al equipo", explica más sobre lonchas finas.

Recuerde comenzar con la rebanada fina o el hilo de acero de la característica que está probando. Accérquese a la automatización del mismo modo que los programadores abordan la codificación. Conseguir una pequeña unidad del hilo de acero funcionando, y luego pasar a la siguiente. Después has cubierto toda la rebanada fina, regresa y sácale cuerpo.

Consulte el Capítulo 9, "Kit de herramientas para Orientado a los negocios Pruebas que respaldan al equipo", para obtener más información sobre el diseño de pruebas eficaces.

Elija cuidadosamente su patrón de prueba. Automatiza todos los casos de prueba que necesita, pero no más, y automaticelos al nivel más bajo que pueda. Límite el alcance de cada caso de prueba a una condición de prueba o una regla de negocio. Comprender el propósito de la prueba. Evite dependencias entre pruebas, porque aumentar rápidamente la complejidad y los gastos de mantenimiento.

Considera opciones

Como mencionamos antes, cuanto menor sea el nivel en el que automatizas una prueba, mejor será el retorno de la inversión. Empuje la automatización de pruebas lo más abajo posible en la pirámide. Si tiene una buena cobertura en sus pruebas unitarias y de integración de código, no necesita automatizar tantas pruebas funcionales. Con cobertura sólida en la parte inferior niveles, podría ser suficiente realizar pruebas de extremo a extremo manualmente para verificar el comportamiento del sistema. Utilice el análisis de riesgos para ayudarle a decidir.

Interfaz de usuario

Es necesario probar la interfaz de usuario. En algunas situaciones, prueba la automatización a nivel de GUI es fundamental. Quizás su equipo esté utilizando controles GUI de terceros y no esté seguro de cómo se comportarán. Si su análisis de riesgo y ROI admite mucha automatización a nivel de GUI, haga la inversión.

Si automatiza en los niveles superiores, no se exceda y automative todos los caminos posibles a través del sistema. No es necesario conservar todas las pruebas automatizadas creadas durante la fase de desarrollo en el conjunto de regresión; considere las ventajas y desventajas del tiempo de construcción y la posibilidad de encontrar defectos. Enfoca tu esfuerzo para cubrir todos los caminos importantes a través del código en la unidad, código integración y niveles funcionales. Obtendrás una recompensa mucho mejor.

Establecer un equilibrio

Lograr un equilibrio no es un principio ágil, es simplemente sentido común. Tu necesitas una solución bastante buena en este momento, pero no tiene por qué ser perfecta. ¿El resultado proporciona los resultados que necesita ahora mismo? ¿Proporciona un rendimiento adecuado de los recursos necesarios para utilizarlo para la automatización? Si es así, sigue adelante y usa y reserva tiempo para buscar alternativas más adelante. Puede mejorar su marco de automatización con el tiempo. El factor más importante es si sus herramientas de automatización se adaptan a su situación particular en este momento.

No te deslices hacia el otro lado y pienses: "Está bien, podemos generar un montón de scripts". con esta herramienta de registro, realice nuestras pruebas inmediatas y refactorice los scripts más tarde para que sean mantenibles". Si bien no es necesario que sigas buscando la solución de automatización perfectamente ideal, necesita una solución que no

aumente la deuda técnica de su equipo. Encuentre un equilibrio entre "Encuentra los errores necesitamos saber y no cuesta demasiado mantener" y "Esto es la solución más elegante y genial que podemos encontrar".

Elegir las herramientas adecuadas

Es genial que tengamos tantas herramientas disponibles para ayudarnos a resolver nuestra automatización. problemas. No busque más sofisticación de la que necesita. compañeros de trabajo de lisa He descubierto que una hoja de cálculo que recupera datos de la base de datos y realiza cálculos independientemente del sistema es una herramienta poderosa, tanto para impulsar el desarrollo y verificar los cálculos de la aplicación.

Generalmente minimizamos la automatización de pruebas en la capa de la GUI, pero hay situaciones en las que es apropiada una mayor automatización de la GUI. Si el usuario hace un cambio en X, ¿qué más cambia? Algunos problemas sólo se manifiestan en la GUI nivel. Lisa probó una solución de error que solucionaba un problema de back-end cuando los participantes del plan de jubilación solicitaban una distribución de dinero de sus cuentas. El cambio estuvo rodeado de pruebas unitarias, pero fue una regresión GUI prueba que falló cuando el formulario de distribución no apareció a pedido. Nadie anticipó que un cambio de back-end pudiera afectar la GUI, por lo que Probablemente no me habría molestado en probarlo manualmente. Por eso necesitas Pruebas de regresión GUI también.

Hemos hablado de algunas desventajas de las herramientas de grabación/reproducción, pero son apropiado en la situación adecuada. Es posible que esté utilizando una herramienta de grabación/reproducción. por una buena razón: tal vez su código heredado ya tenga un conjunto de funciones automatizadas pruebas creadas en esa herramienta, su equipo tiene mucha experiencia en la herramienta, o su La gerencia quiere que lo use por cualquier motivo. Puedes usar grabado scripts como punto de partida, luego divida los scripts en módulos, reemplace los datos codificados con parámetros cuando corresponda y ensamble las pruebas utilizando el módulos como bloques de construcción. Incluso si no tienes mucha experiencia en programación, no es difícil identificar los bloques de script que deberían estar en un módulo. Iniciar sesión, por ejemplo, es una opción obvia.

La grabación/reproducción también puede ser apropiada para sistemas heredados que están diseñados de tal manera que dificultan las pruebas unitarias y las pruebas de escritura manual. desde cero demasiado costoso. Es posible crear una capacidad de grabación y reproducción. en la aplicación, incluso una aplicación heredada. Con el diseño correcto y el uso de algún formato legible por humanos para la interacción grabada, es incluso Es posible crear pruebas de reproducción antes de crear el código.

Automatización de pruebas de GUI: de la Edad Media a la automatización exitosa en un entorno ágil

Pierre Veragen, líder de SQA en iLevel by Weyerhaeuser, explica cómo su equipo utilizó una herramienta que proporcionaba capacidad de grabación/reproducción y secuencias de comandos de manera productiva en un entorno en cascada, y luego la aprovechó cuando la empresa adoptó el desarrollo ágil.

En nuestros días de desarrollo en cascada, en el año 2000, comenzamos a automatizar las pruebas de GUI utilizando un enfoque de reproducción de grabaciones. Rápidamente acumulamos decenas de miles de líneas de guiones grabados que no satisfacían nuestras necesidades de prueba. Cuando asumí el cargo, 18 meses después, rápidamente me convencí de que el enfoque de reproducción de discos era para los dinosaurios.

Cuando tuvimos la oportunidad de obtener una nueva herramienta de automatización de pruebas a finales de 2003, evaluamos cuidadosamente las herramientas teniendo en cuenta estos criterios: capacidad de registro para ayudarnos a comprender el lenguaje de secuencias de comandos y la capacidad de crear una biblioteca orientada a objetos para cubrir la mayoría de nuestras necesidades, incluidos los informes de pruebas. En aquel momento TestPartner de CompuWare cumplió todos nuestros requisitos.

Comenzamos a utilizar TestPartner en una aplicación CAD con ingeniería altamente compleja, construida en Visual Basic 6, que aún utiliza un proceso en cascada. Antes de comenzar a automatizar las pruebas, a nuestros lanzamientos les seguían rápidamente uno o más parches. Centramos nuestros esfuerzos de automatización en verificar los cálculos de ingeniería a través de la GUI y, posteriormente, la posición real de los detalles CAD. Estas pruebas incluyeron cientos de miles de puntos de verificación individuales, que nunca se podrían haber realizado a mano. Al cabo de un año, después de haber agregado un sólido conjunto de pruebas manuales de la interacción del usuario, además de nuestras pruebas automatizadas, estábamos lanzando un software robusto sin los parches de seguimiento habituales. Nos sentíamos seguros de nuestra combinación de pruebas manuales y automatizadas, que no incluían una sola línea de guiones grabados.

En 2004, nuestro grupo pasó a Visual Basic .NET. Pasé varios meses adaptando nuestra biblioteca TestPartner para activar controles .NET. En 2006, adoptamos una metodología ágil. Aprovechando las lecciones aprendidas anteriormente en el mundo no ágil, logramos resultados sorprendentes con la automatización de pruebas. A finales de 2006, los miembros del equipo pudieron producir scripts de prueba de GUI y componentes de biblioteca que se pudieran mantener después de unos pocos días de capacitación. Al mismo tiempo, el equipo adoptó pruebas unitarias con NUnit y pruebas de aceptación de usuarios con FitNesse.

Al momento de escribir este artículo, los problemas se detectan en los tres niveles de nuestras pruebas automatizadas: Unidad, FitNesse y GUI. Los problemas encontrados en cada uno de los tres niveles de prueba son de naturaleza diferente. Debido a que todo está automatizado y se activa automáticamente, los problemas se detectan muy rápido, de manera verdaderamente ágil. Cada parte de nuestra automatización de pruebas aporta valor.

Algunas personas creen que sería mejor gastar los recursos en arquitectura y diseño, de modo que no sea necesaria la automatización de las pruebas de GUI. En nuestro grupo de desarrollo, cada equipo tomó su propia decisión sobre si automatizar las pruebas de GUI.

En caso de que decida utilizar la automatización de pruebas de GUI, aquí tiene algunos consejos: manténgase alejado de los scripts grabados, invierta en mantenibilidad y minimice las pruebas de GUI requeridas con una buena arquitectura de la aplicación. Según mi experiencia, invertir en buenas prácticas de automatización de pruebas de GUI siempre dará sus frutos.

El consejo de Pierre refleja bien cómo las buenas prácticas de desarrollo, especialmente aquellas seguidas en proyectos de desarrollo ágil, se aplican al desarrollo de pruebas automatizadas así como al desarrollo de código de producción.

Grabación y reproducción integradas

Gerard Meszaros, entrenador ágil y autor de [2007], describe una situación en la que el enfoque más simple resultó ser grabar/reproducción. Hemos mencionado los inconvenientes de las herramientas de grabación/reproducción, pero si diseña su código para admitirlas, pueden ser el mejor enfoque.

Me pidieron que ayudara a un equipo que estaba migrando una aplicación "sensible a la seguridad" de OS2 a Windows. La empresa estaba muy preocupada por la cantidad de tiempo que llevaría volver a probar el sistema portado y la probabilidad de que el equipo pasara por alto errores importantes. El sistema fue diseñado para ofrecer al usuario únicamente opciones válidas que no comprometan la seguridad. Consideraron usar una herramienta de grabación de pruebas para grabar pruebas en el sistema antiguo y reproducirlas en el nuevo sistema, pero no había herramientas de grabación de pruebas disponibles tanto para OS2 como para Windows que pudieran manejar ventanas dibujadas con caracteres ASCII. Después de revisar la arquitectura del sistema, determinamos que escribir pruebas de xUnit no sería una forma rentable de probar el sistema porque gran parte de la lógica de negocios estaba integrada en la lógica de la interfaz de usuario, y refactorizar el código para separarlas sería demasiado arriesgado y llevar mucho tiempo. En cambio, propusimos crear una capacidad de prueba de grabación y reproducción directamente en el sistema antes de migrarlo.

Aunque el resto del proyecto se basó en hitos, desarrollamos el mecanismo de prueba integrado de una manera muy ágil. Cada pantalla requería al menos un gancho nuevo y, a veces, varios. Comenzamos con las pantallas más utilizadas, agregando los ganchos necesarios para registrar las acciones del usuario, acciones y las respuestas del sistema a ellas en un archivo XML. También agregamos enlaces para reproducir el XML y determinar los resultados de la prueba. Inicialmente, centramos nuestros esfuerzos en probar el concepto enganchando sólo

las pantallas que necesitábamos para grabar y luego reproducir una prueba simple pero realista. Después de que todos estuvieron convencidos de que el enfoque funcionaría, priorizamos las pantallas con respecto al beneficio que proporcionaría. Implementamos los ganchos uno por uno hasta que pudimos automatizar una parte importante de las pruebas. También creamos una hoja de estilo XSLT que formatearía el XML de forma similar a Fit, con celdas verdes que indican resultados aceptables y celdas rojas que indican un paso de prueba fallido.

Mientras tanto, el cliente identificaba los escenarios de prueba que necesitaban casos de prueba. Cuando terminábamos suficientes pantallas para grabar una prueba en particular, el cliente hacía una "prueba de aceptación" de nuestros ganchos grabando y reproduciendo (aún en OS2) las pruebas que estaban esperando esos ganchos. Cuando todos los enlaces estuvieron en su lugar, pudimos continuar y transferir el código, incluidos los enlaces de prueba, de OS2 a Windows. Después de verificar la reproducción exitosa en OS2, el cliente movería los archivos de prueba XML a Windows y ejecútelos con la versión portada del código.

Al cliente le resultó bastante fácil hacer esto y pudo registrar una gran cantidad de pruebas en un período de tiempo relativamente corto. Debido a que las pruebas registraban acciones y respuestas en términos comerciales, fueron bastante fáciles de entender. Al cliente le encantó la capacidad y todavía elogia cuánto esfuerzo ahorró y cuánta más confianza tiene en el producto. "Esto no solo ahorró decenas de años-hombre de esfuerzo de prueba, sino que incluso descubrió errores desconocidos ocultos en el sistema heredado, que habíamos considerado el estándar de oro".

En la historia de Gerard, el equipo trabajó en conjunto para adaptar la capacidad de prueba a un sistema que no fue diseñado para dicha capacidad. Brindaron a sus clientes una manera de capturar sus escenarios de prueba en una plataforma y reproducirlos en ambas plataformas para verificar el puerto exitoso. Este es un ejemplo estelar del enfoque de todo el equipo. Cuando todos los miembros del equipo colaboran en una solución de automatización de pruebas, hay muchas más posibilidades de que tenga éxito.

Ver más ejemplos
de
herramientas para
pruebas de cara al
negocio en el Capítulo
9, "Kit de herramientas
para pruebas de cara
al negocio que
apoyan al equipo".

Algunos equipos ágiles obtienen valor de las herramientas de prueba comerciales o de código abierto, mientras que otros prefieren un enfoque completamente personalizado. Muchos evaluadores encuentran valioso escribir scripts simples en un lenguaje de scripting como Ruby, o un shell, para automatizar tareas mundanas pero necesarias, generar datos de prueba o manejar otras herramientas. Libros como *Everyday Scripting with Ruby for Teams, Testers and You* brindan una hoja de ruta para este enfoque. Si es un evaluador sin una sólida experiencia en programación, le recomendamos que lea un libro, busque un tutorial en línea o realice una prueba.

clase sobre un lenguaje de secuencias de comandos y vea lo fácil que puede ser escribir secuencias de comandos útiles.

Lo que intentamos decirte es que puedes utilizar muchas herramientas diferentes. Mira a el problema que intentas resolver y decidir en equipo la forma más fácil y más forma eficaz de solucionarlo. De vez en cuando, da un paso atrás y echa un vistazo a las herramientas.

estás usando. ¿Están todos los miembros del equipo contentos con ellos? ¿Se le escapan problemas porque no tiene las herramientas adecuadas? Dedique tiempo a explorar nuevas herramientas y ver si podrían llenar vacíos o reemplazar una herramienta que no está dando sus frutos.

Si su equipo es nuevo en el desarrollo ágil o está trabajando en un proyecto nuevo, Es posible que tenga que elegir herramientas y configurar entornos de prueba. durante las primeras iteraciones, cuando también podría estar trabajando en historias de alto riesgo. No espere poder ofrecer mucho valor empresarial si todavía está creando su infraestructura de prueba. Planifique con suficiente tiempo para evaluar herramientas, establecer desarrollar procesos de construcción y experimentar con diferentes enfoques de prueba.

APLICACIÓN DE PRINCIPIOS ÁGILES A LA AUTOMATIZACIÓN DE PRUEBAS

Cada equipo, cada proyecto y cada organización tiene una situación única con desafíos de automatización únicos. Cada uno tiene su propia cultura, historia, recursos, presiones comerciales, productos y experiencia. No importa cuál sea tu equipo situación, puede utilizar los principios y valores ágiles discutidos en el Capítulo 2 para ayudarte a encontrar soluciones. Conceptos como coraje, retroalimentación, sencillez, La comunicación, la mejora continua y la respuesta al cambio no son solo ideas ágiles: son cualidades comunes a todos los equipos exitosos.

Mantenlo simple

La máxima ágil de “hacer lo más simple que pueda funcionar” se aplica a pruebas y código. Mantenga el diseño de la prueba simple, mantenga el alcance mínimo y Utilice la herramienta más sencilla que haga el trabajo.

La simplicidad es un valor ágil fundamental por una buena razón. El mejor lugar para empezar es el enfoque más simple que puedas imaginar. Sin embargo, hacer lo más simple no significa hacer lo más fácil. Implica pensar realmente en lo que necesitas ahora y dando pequeños pasos para llegar allí. Manteniendo las cosas simples, si lo haces Si haces una mala elección, no te desviará demasiado antes de darte cuenta del error de tus caminos.

Es fácil involucrarse en una tarea y alejarse de lo básico hacia algún desafío intrigante. Evalúe el retorno de la inversión de cada tarea de automatización antes de realizarla. La automatización es divertida (cuando superas la parte aterradora de empezar). Es Es tentador intentar algo difícil sólo porque puedes. Como todos los demás aspectos de las pruebas en un proyecto de desarrollo ágil, la única forma de mantenerse al día es hacer sólo lo mínimo requerido.

Utilice la herramienta más sencilla que pueda utilizar. Recuerda la pirámide de pruebas. si un La prueba de cara al cliente se puede automatizar más fácilmente a nivel de unidad, hágalo

allá. Lisa a veces escribe casos de prueba en FitNesse, solo para descubrir que los programadores pueden automatizarlos mucho más rápido que las pruebas JUnit. Por el contrario, a veces los programadores usan FitNesse para TDD en lugar de JUnit, porque el código que están escribiendo se presta para ser probado en uno de los formatos de accesorios de FitNesse.

Comentarios iterativos

Las iteraciones cortas nos permiten experimentar con varios enfoques de automatización, evaluar los resultados y cambiar de rumbo tan rápido como sea necesario. Comprométase con un esfuerzo de automatización, como desarrollar un marco de prueba interno o implementar una herramienta de código abierto durante al menos un par de iteraciones. Despues de cada iteración, Mire lo que funciona y lo que no. Piensa en ideas para superar problemas y pruébelos en la siguiente iteración. Si no es la solución correcta, intente algo más durante algunas iteraciones. No se deje atrapar por un atolladero donde has invertido tantos recursos en una herramienta y tienes tantas pruebas que la utilizan, que sientes que no puedes cambiar de herramienta. Entre las muchas herramientas comerciales y de código abierto, además de la capacidad de los programadores para escribir herramientas de prueba locales, no hay razón para conformarse con menos de la herramienta óptima.

La historia de Lisa

Uno de mis primeros equipos de XP luchó por encontrar una buena manera de automatizar las pruebas de aceptación de cara al cliente para una aplicación web basada en Java. Esto fue cuando había muchas menos opciones de herramientas para los equipos ágiles. Primero, probamos una herramienta de código abierto que simulaba un navegador, pero carecía de las funciones que necesitábamos. Simplemente no fue bastante robusto. Hablamos de esto en la próxima retrospectiva.

Decidimos intentar utilizar la herramienta de prueba unitaria para realizar pruebas detrás de la GUI durante las siguientes dos iteraciones. Al comprometernos con dos iteraciones, sentimos que nos estábamos dando tiempo suficiente para probar la herramienta, pero no tanto tiempo como para haber invertido demasiado si no fuera la solución correcta. Los clientes encontraron que las pruebas unitarias eran difíciles de leer y había una lógica en la GUI que no pudimos probar con esta herramienta.

Después de otra discusión durante nuestra retrospectiva, nos comprometimos a dos iteraciones de uso de una herramienta de prueba de GUI del proveedor que había usado ampliamente en proyectos anteriores. A los programadores de Java les resultó lento porque la herramienta utilizaba un lenguaje de secuencias de comandos propietario, pero funcionó lo suficientemente bien como para realizar la automatización mínima necesaria. Después de dos iteraciones, decidimos que no era lo ideal, pero en ese momento no había muchas otras opciones y era la mejor que teníamos.

En retrospectiva, deberíamos haber seguido buscando una mejor opción. Quizás podríamos haber desarrollado nuestro propio arnés de prueba. Pudimos automatizar alrededor del 60 % de las pruebas de regresión por encima del nivel unitario utilizando la herramienta del proveedor, lo que parecía excelente en ese momento. Si nos hubiéramos esforzado un poco más, podríamos haberlo hecho mucho mejor.

—Lisa

Utilice las iteraciones a su favor. Facilitan un enfoque gradual. Si tu idea es un fracaso, lo sabrás rápidamente y tendrás la oportunidad de probar otra diferente. No tenga miedo de seguir buscando, pero no siga buscando la solución perfecta si una que prueba funciona adecuadamente.

Enfoque de equipo completo

El desarrollo ágil no puede funcionar sin automatización. Afortunadamente, el enfoque de equipo completo, que exploramos en el Capítulo 1, significa que una gama más amplia de habilidades y recursos disponibles para encontrar e implementar una automatización útil. Hay habilidades y recursos disponibles para encontrar e implementar una automatización útil. estrategia. Atacar el problema en equipo significa que es más probable que el código funcione. estar diseñados para ser comprobables. Programadores, probadores y otros miembros del equipo. colaborará para automatizar pruebas, aportando múltiples puntos de vista y conjuntos de habilidades al esfuerzo.

El enfoque de todo el equipo ayuda a superar la barrera del miedo. Tareas de automatización Puede resultar abrumador para empezar. Saber que hay otras personas con diferentes habilidades y experiencia que pueden ayudarnos nos da valor. Ser capaz de pedir y recibir ayuda nos da confianza de que podemos lograr una cobertura adecuada con nuestras pruebas automatizadas.

La historia de Lisa

Mi equipo actual se comprometió a automatizar las pruebas de regresión en todos los niveles donde tuviera sentido. A continuación se muestran algunos ejemplos de casos en los que pedí ayuda para tener éxito con la automatización.

Al principio, cuando no teníamos ninguna prueba automatizada y los desarrolladores intentaban dominar el desarrollo basado en pruebas, nos decidimos por Canoo WebTest para las pruebas de humo de GUI. Necesitaba un poco de ayuda para comprender cómo configurar WebTest para que se ejecute en nuestro entorno y necesitaba mucha ayuda para ejecutar las pruebas desde el proceso de compilación automatizado. Le pedí ayuda a nuestro administrador del sistema (que también era uno de los programadores). Rápidamente obtuvimos un conjunto de pruebas ejecutándose en la compilación.

Más tarde, tenía muchas ganas de probar FitNesse para realizar pruebas funcionales detrás de la GUI. Tuve que ser paciente mientras los programadores todavía estaban ganando terreno con las pruebas unitarias automatizadas. El equipo acordó probar la herramienta, pero fue difícil encontrar tiempo para empezar a utilizarla. Elegí una historia que parecía adecuada para las pruebas de FitNesse y le pregunté al programador que trabajaba en la historia si podía emparejarme con él para probar algunas pruebas de FitNesse. Estuve de acuerdo y automatizamos algunas pruebas en FitNesse. Al programador le resultó fácil y útil, y dio un buen informe al resto del equipo.

Después de eso, no fue difícil acercarse a cada programador, sugerirle que escribiera pruebas de FitNesse para la historia en la que estaba trabajando y dejarle ver los resultados. Las pruebas de FitNesse encontraron casos de prueba en los que el programador no había pensado y vieron el beneficio de inmediato. Cuando todos los miembros del equipo tuvieron algo de experiencia con la herramienta, no solo estuvieron felices de automatizar las pruebas, sino que comenzaron a diseñar código de una manera que facilitara la escritura de accesos de FitNesse.

Cuando nuestro experto en Ruby, que diseñó la mayor parte de nuestro conjunto de pruebas Watir, dejó la empresa, estaba bastante preocupado por mantener nuestro enorme conjunto de pruebas y por poder codificar otras nuevas. Mi experiencia en Ruby no era tan buena como la suya (además, solo nos quedaba un evaluador, por lo que el tiempo era un problema). Todos los programadores del equipo salieron, compraron un libro sobre Ruby y me ayudaron cuando tuve problemas para actualizar los scripts para que funcionaran cuando el código cambiaba. Un programador incluso escribió un nuevo guion para probar una nueva historia cuando no tenía tiempo para esa tarea. Cuando contratamos a un nuevo evaluador, él y yo pudimos encargarnos del cuidado y la alimentación de los scripts de Watir, por lo que los programadores ya no necesitaron asumir esas tareas.

Sé que puedo pedir ayuda a mis compañeros de equipo con problemas de automatización y todo el equipo ve la automatización como una prioridad, por lo que los programadores siempre piensan en la capacidad de prueba al diseñar el código. Este es un ejemplo del enfoque de trabajo de todo el equipo.

—Lisa

El capítulo 11, "Crítica del producto mediante pruebas tecnológicas", habla sobre pruebas tecnológicas como estas y diferentes enfoques para manejarlas.

Las pruebas tecnológicas especializadas, como las pruebas de seguridad o de carga, pueden requerir la incorporación de expertos externos al equipo. Algunas empresas cuentan con equipos de especialistas que están disponibles como recursos compartidos para los equipos de productos. Incluso

Si bien aprovechan estos recursos, los equipos ágiles aún deben asumir la responsabilidad de garantizar que se realicen todos los tipos de pruebas. También pueden ser

Me sorprende descubrir que los miembros del equipo pueden tener las habilidades necesarias si toman un enfoque creativo.

Algunas organizaciones tienen equipos de prueba independientes que realizan tareas de posdesarrollo. pruebas. Es posible que estén realizando pruebas para garantizar que el software se integre con otros sistemas o realizando otras pruebas especializadas, como pruebas de rendimiento a gran escala.

pruebas. Los equipos de desarrollo deben trabajar estrechamente con estos otros equipos, utilizando la retroalimentación de todos los esfuerzos de prueba para mejorar el diseño del código y facilitar automatización.

Tomarse el tiempo para hacerlo bien

Resolver problemas e implementar buenas soluciones lleva tiempo. debemos ayudar Nuestra gerencia entiende que sin tiempo suficiente para hacer las cosas bien

De esta manera, nuestra deuda técnica crecerá y nuestra velocidad se ralentizará. Implementar Soluciones de la manera "correcta" requieren tiempo desde el principio, pero ahorrarán tiempo a largo plazo. término. Considere el tiempo que lleva generar ideas, soluciones y capacitación y aprendizaje en el trabajo.

Es comprensible que la dirección de su organización esté interesada en producir resultados lo más rápido posible. Si la dirección se muestra reacia a darle tiempo al equipo Para implementar la automatización, explique claramente las compensaciones. Ofrecer algunas funciones a corto plazo sin pruebas de regresión automatizadas para garantizar que

Seguir trabajando tendrá un gran coste en el futuro. A medida que tu equipo acumula deuda técnica, será menos capaz de ofrecer la gestión del valor empresarial necesidades. Trabaje para llegar a un compromiso. Por ejemplo, recortar el alcance de una característica, pero mantenga el valor esencial y utilice la automatización para entregar y mantener un mejor producto.

Siempre tenemos plazos y siempre nos sentimos presionados por el tiempo. La tentación de volver a hacer las cosas como siempre las hemos hecho, como ejecutar pruebas de regresión manualmente y esperar lo mejor, siempre está ahí, aunque Sabemos que eso no funciona. Nunca hay tiempo suficiente para volver atrás y arreglar cosas. Durante su próxima reunión de planificación, reserve tiempo para tomar decisiones significativas. progreso en sus esfuerzos de automatización.

La historia de Lisa

Nuestro equipo se enfoca en tomarse el tiempo para un buen diseño, un sólido conjunto de pruebas automatizadas y tiempo suficiente para pruebas exploratorias. La calidad, no la velocidad, siempre ha sido nuestro objetivo. Nuestros problemas de producción cuestan mucho solucionar, por lo que toda la empresa está a bordo para tomarse el tiempo para evitarlos. A veces no elegimos el diseño correcto y no tenemos miedo de arrancarlo y reemplazarlo cuando nos damos cuenta.

Naturalmente, existen compensaciones comerciales y la empresa decide si procede con riesgos conocidos. Trabajamos para explicar todos los riesgos con claridad y dar ejemplos de escenarios potenciales.

Aquí hay un par de ejemplos recientes de cómo tomarse el tiempo para hacer las cosas bien. Iniciamos un tema para hacer cambios importantes en los estados de cuenta de los planes de retiro. Uno de los programadores, Vince Palumbo, asumió la tarea de recopilar datos adicionales para utilizarlos en las declaraciones. Decidió escribir pruebas unitarias sólidas para la funcionalidad de recopilación de datos, aunque esto significaba que la historia tendría que continuar en la siguiente iteración. Escribir las pruebas unitarias requirió mucho tiempo y esfuerzo, e incluso con las pruebas, el código era extremadamente complejo y difícil de hacer. Un par de iteraciones más tarde, otro programador, Nanda Lankala-palli, conoció otra historia relacionada con la recopilación de datos y quedó gratamente sorprendido al encontrar nuevas pruebas unitarias. Pudo realizar sus cambios rápidamente y el esfuerzo de prueba se redujo considerablemente porque las pruebas unitarias estaban implementadas.

Más tarde, descubrimos que nos habíamos perdido un caso extremo en el que algunos cálculos para el cambio en el valor de la cuenta eran incorrectos. La combinación de pruebas unitarias automatizadas y una gran cantidad de pruebas exploratorias no fueron suficientes para detectar todos los escenarios. Aún así, tener las pruebas significó que Vince pudo escribir su código corregido primero como prueba y sentirse más seguro de que el código ahora era correcto.

Otro ejemplo reciente se refería al procesamiento de cheques entrantes. La empresa quería acortar el proceso de dos pasos a uno, lo que significaba que el dinero se invertiría en las cuentas del plan de jubilación dos días antes de lo que era posible en ese momento. Todo el proceso existente fue escrito en código heredado, sin pruebas unitarias.

Discutimos si reescribir el procesamiento en la nueva arquitectura. El propietario de nuestro producto estaba preocupado por la cantidad de tiempo que esto podría llevar. Nosotros sentíamos

Tomaría tanto tiempo cambiar el código existente como reescribirlo por completo, porque el código antiguo era difícil de entender y no tenía ninguna prueba unitaria. Nos decidimos por la reescritura, que no sólo redujo el riesgo de problemas en esta funcionalidad crítica sino que también nos dio la oportunidad de proporcionar un par de características adicionales con un pequeño costo adicional. Hasta ahora, esta estrategia ha demostrado que vale la pena.

—Lisa

Permitete tener éxito. Trabajar a un ritmo sostenible. Tómese el tiempo para refactorizar a medida que avanza o eventualmente terminará con un desastre. Como probadores, siempre tener muchas tareas diferentes que hacer. Si está aprendiendo una nueva herramienta o intentando automatizar nuevas pruebas, no realice múltiples tareas. Encuentre un gran bloque de tiempo y concéntrese. Esto es Es difícil, pero cambiar de marcha constantemente es más difícil.

Si las partes interesadas del negocio están impacientes por que su equipo “simplemente lo haga”, analice el problema con ellos. ¿Cuáles son los riesgos? ¿A cuánto ascenderá una producción? costo del problema? ¿Cuáles son los beneficios de lanzar un truco rápido? Cuánto ¿Deuda técnica sumará? ¿Cuál es el retorno de la inversión a largo plazo de una ¿Diseño sólido respaldado con pruebas automatizadas? ¿Cómo afectará cada enfoque? ¿La rentabilidad de la empresa y la satisfacción del cliente? ¿Qué pasa con los intangibles? costos, como el efecto que tiene en la moral del equipo hacer un trabajo de mala calidad? A veces el negocio saldrá bien, pero apostamos a que normalmente lo encontrarás esa inversión inicial vale la pena.

Aprender haciendo

Cada uno aprende de diferentes maneras, pero cuando has decidido cómo vas para automatizar una prueba, salte y comience a hacerlo. En secuencias de comandos cotidianas con Ruby para Teams, Testers, and You [2007], Brian Marick aconseja aprender a programar escribiendo un programa. ¡Cometer errores! Cuantos más problemas tengas, más más aprenderás. Conseguir que alguien se asocie con usted ayudará a acelerar el aprendizaje, incluso si ninguno de los dos está familiarizado con la herramienta o el idioma.

Si no tienes con quién emparejarte, habla con el “patito de goma”: Imagínate le estás describiendo el problema a un compañero de trabajo. El proceso de explicación puede a menudo hacen que la causa del problema salte a la vista. Simplemente leyendo una prueba Hablarlo en voz alta puede ayudarte a encontrar sus debilidades.

Aplicar prácticas de codificación ágil a las pruebas

Las pruebas son tan valiosas como el código de producción. De hecho, el código de producción no es mucho sin pruebas que lo respalden. Trate sus pruebas de la misma manera que trata a todos código. Guárdelo en la misma herramienta de control de código fuente que su código de producción.

Siempre debería poder identificar las versiones de los scripts de prueba que van con un versión particular del código.

Emparejamiento, refactorización, diseño simple, diseño modular y orientado a objetos, bueno estándares, manteniendo las pruebas lo más independientes posible: todas las cualidades del buen El código también son cualidades de buenas pruebas automatizadas. A veces, quienes no están informados perciben el desarrollo ágil como caótico o laxo, cuando en realidad es altamente disciplinado. Emprende tus tareas de automatización con la mayor disciplina, procediendo en pequeños pasos, comprobando cada paso exitoso. Si está programando scripts automatizados, escríbalos primero como prueba, como cualquier programador ágil.
escribiría código de producción. Sin embargo, tenga en cuenta la simplicidad. no escribas scripts de prueba sofisticados con mucha lógica a menos que haya un buen retorno de la inversión. Esas pruebas necesitan pruebas y su mantenimiento cuesta más. Especifique pruebas cuando pueda en lugar de codificar ellos, y siempre opte por el enfoque más simple posible.

No podemos enfatizarlo lo suficiente: la automatización de pruebas es un esfuerzo de equipo. La variación La experiencia, las habilidades y las perspectivas de diferentes miembros del equipo pueden trabajar juntos para encontrar el mejor enfoque para la automatización. Innovar: ser creativo. Haga lo que funcione para su situación particular, sin importar cuál sea el "concepto común".
sabiduría", dice.

Las herramientas de automatización son sólo una pieza del rompecabezas. Entornos de prueba y prueba. Los datos son componentes esenciales. Veamos los datos de prueba a continuación.

SUMINISTRO DE DATOS PARA PRUEBAS

No importa qué herramienta utilicemos para automatizar las pruebas, las pruebas necesitan datos para procesar. Idealmente, necesitan datos realistas que coincidan con los datos de producción. Sin embargo, las bases de datos de producción suelen contener muchísimos datos y pueden altamente complejo. Además, el acceso a la base de datos ralentiza exponencialmente las pruebas. Al igual que En gran parte de las pruebas ágiles, es un acto de equilibrio.

Herramientas de generación de datos

Mientras escribimos este libro, hay varias herramientas interesantes disponibles para generar pruebas. datos para todo tipo de campos de entrada y condiciones de entorno. Código abierto y herramientas comerciales como Data Generator, databene benerator, testgen, Datatect y Turbo Data están disponibles para generar archivos planos o generar datos directamente a las tablas de la base de datos. Estas herramientas pueden generar enormes variedades de diferentes tipos de datos, como nombres y direcciones.

También es bastante fácil generar datos de prueba con un script propio, utilizando un lenguaje de programación como Ruby o Python, una herramienta como Fit o FitNesse, o un script de shell.

La historia de Lisa

Nuestros scripts Watir crean entradas de datos de prueba aleatorias, tanto para garantizar que se puedan volver a ejecutar (es poco probable que creen un empleado con el mismo SSN dos veces) como para proporcionar una variedad de datos y escenarios. El guión que crea nuevos planes de jubilación produce planes con alrededor de 200 combinaciones diferentes de opciones. El script que prueba la obtención de un préstamo genera aleatoriamente la frecuencia, el motivo y el plazo del préstamo, y verifica que el pago esperado sea correcto.

Contamos con scripts de utilidad para crear archivos separados por comas para probar las cargas. Por ejemplo, hay varios lugares en el sistema que cargan archivos del censo con información de nuevos empleados. Si necesito un archivo de prueba con 1000 nuevos empleados con asignaciones aleatorias de inversiones a un plan de jubilación, simplemente puedo ejecutar el script y especificar la cantidad de empleados, los fondos mutuos en los que están invirtiendo y el nombre del archivo. Cada registro tendrá un número de seguro social generado aleatoriamente, nombre, dirección, beneficiarios, montos de aplazamiento de salario y asignaciones de fondos de inversión. Aquí hay un fragmento del código para generar los cálculos de inversión.

```
# El 33% de las veces maximiza la cantidad de fondos elegidos, el 33% de las veces.
# seleccione un solo fondo y el 33% de las veces seleccione entre 2 y 4 fondos
fund_hash = caso rand(3)
    cuando 0: a.get_random_allocations(@fund_list.clone)
    cuando 1: a.get_random_allocations(@fund_list.clone, 1)
cuando 2: a.min_percent = 8;
a.get_random_allocations(@fund_list.clone, rand(3) + 2)
fin
emp['fund_allocations'] = fund_hash_to_string(fund_hash)
```

Scripts como estos tienen usos duales, como pruebas de regresión que cubren muchos escenarios diferentes y como herramientas de prueba exploratorias que crean datos de prueba y construyen escenarios de prueba. No es difícil aprender a escribir (consulte la sección "Aprender haciendo" anteriormente en este capítulo).

—Lisa

Los scripts y las herramientas para generar datos de prueba no tienen por qué ser complejos. Por ejemplo, PerlClip simplemente genera texto en el portapapeles de Windows para poder pegarlo en donde sea necesario. Vale la pena probar cualquier solución que elimine el tedio suficiente como para permitirle descubrir posibles problemas relacionados con la aplicación. "Lo más simple algo que posiblemente podría funcionar" definitivamente se aplica a la creación de datos para pruebas. Quiere que sus pruebas sean lo más simples y rápidas posible.

Evite el acceso a la base de datos

Su primera opción para realizar pruebas debe intentar tener pruebas que puedan ejecutarse completamente en memoria. Aún necesitarán configurar y eliminar datos de prueba, pero el Los datos no se almacenarán en una base de datos. Cada prueba es independiente y se ejecuta tan rápido como cualquier prueba podría. El acceso a la base de datos significa que la E/S y los discos son inherentemente lentos. Cada lectura de la base de datos ralentiza la ejecución de la prueba. Si tu objetivo es dar rápido comentarios al equipo, entonces deseas que sus pruebas se ejecuten lo más rápido posible. Un objeto falso, como una base de datos en memoria, permite que la prueba haga lo que necesita. hacer y aún así dar retroalimentación instantánea.

La historia de Lisa

Integral

Se pueden encontrar explicaciones y ejemplos de varios tipos de dobles de prueba xUnidadPrueba Patronen . Consulte la bibliografía para obtener más información al respecto y herramientas para trabajar con resguardos de prueba y con objetos simulados y falsos.

Uno de nuestros procesos de compilación ejecuta solo pruebas a nivel de unidad e intentamos mantener su tiempo de ejecución en menos de ocho minutos para obtener una respuesta óptima. Las pruebas sustituyen la base de datos real por objetos falsos en la mayoría de los casos. Las pruebas que en realidad prueban la capa de la base de datos, como la persistencia de datos en la base de datos, utilizan un esquema pequeño con datos canónicos copiados originalmente de la base de datos de producción. Los datos son realistas, pero la pequeña cantidad hace que el acceso sea más rápido.

A nivel de prueba funcional, nuestros dispositivos de prueba FitNesse crean datos en memoria siempre que sea posible. Estas pruebas se ejecutan rápidamente y los resultados aparecen casi instantáneamente. Cuando necesitamos probar la capa de base de datos, o si necesitamos probar código heredado al que no se puede acceder independientemente de la capa de base de datos, normalmente escribimos pruebas de FitNesse que configuran y eliminan sus propios datos utilizando un dispositivo de datos propio. Estas pruebas son necesarias, pero se ejecutan lentamente y su mantenimiento es costoso, por lo que las mantenemos en el mínimo absoluto necesario para darnos confianza.

Queremos que nuestra versión que ejecuta pruebas empresariales proporcione comentarios en un par de horas para mantenernos productivos.

—Lisa

Herramientas como DbFit y NdbUnit pueden simplificar las pruebas de bases de datos y permitir el desarrollo de bases de datos basadas en pruebas; consulte la bibliografía para obtener más recursos.

Debido a que es tan difícil impulsar la automatización de pruebas, sería fácil decir "Está bien, tenemos algunas pruebas y tardan horas en ejecutarse, pero es mejor que ninguna prueba". El acceso a la base de datos es un factor importante que contribuye a la lentitud de las pruebas. Continúe tomando pequeños pasos para falsificar la base de datos donde pueda y pruebe toda la lógica que pueda, posible sin involucrar la base de datos. Si esto le resulta difícil, reevalúe su arquitectura del sistema y ver si se puede organizar mejor para las pruebas.

Si está probando lógica de negocios, algoritmos o cálculos en código, le interesa el comportamiento del código en sí dadas ciertas entradas; no te importa de dónde provienen los datos, siempre que representen con precisión datos reales. Si esto es el caso, cree datos de prueba que formen parte de la prueba y a los que se pueda acceder en la memoria, y deje que el código de producción opere a partir de ahí. Simular el acceso a la base de datos y objetos, y centrarse en el propósito de la prueba. No sólo se realizarán las pruebas más rápido, pero serán más fáciles de escribir y mantener.

Al generar datos para una prueba, utilice valores que reflejen la intención de la prueba, donde sea posible. A menos que esté completamente seguro de que cada prueba es independiente, genere valores de prueba únicos para cada prueba. Por ejemplo, use marcas de tiempo como parte de los valores del campo. Los datos únicos son otra red de seguridad para evitar que se realicen pruebas. infectándose unos a otros con datos perdidos. Cuando necesite grandes cantidades de datos, Intente generar los datos aleatoriamente, pero siempre límpielos al final de la prueba. para que no sangre en la siguiente prueba. Reconocemos que a veces Es necesario probar tipos de datos muy específicos. En estos casos, generado aleatoriamente datos anularían el propósito de la prueba. Pero es posible que puedas usar suficiente aleatorización para garantizar que cada prueba tenga entradas únicas.

Cuando el acceso a la base de datos es inevitable o incluso deseable

Si el sistema bajo prueba depende en gran medida de la base de datos, esto naturalmente debe ser probado. Si el código que está probando lee y/o escribe en la base de datos, en algún momento necesitarás probar eso, y probablemente querrás al menos algunas pruebas de regresión que verifiquen la capa de código de la base de datos.

Datos de configuración/desmontaje para cada prueba

Nuestro enfoque preferido es hacer que cada prueba agregue los datos que necesita a una prueba. esquema, operar con los datos, verificar los resultados en la base de datos y luego eliminar todos los datos de la prueba para que la prueba pueda volver a ejecutarse sin afectar otras pruebas posteriores. Esto apoya la idea de que las pruebas son independientes entre sí.

La historia de Lisa

Usamos un dispositivo de datos genérico que permite a la persona que escribe la prueba especificar la tabla de la base de datos, las columnas y los valores de las columnas para agregar datos. Otro dispositivo de búsqueda de datos genéricos nos permite ingresar un nombre de tabla y una cláusula SQL donde para verificar los datos persistentes reales. También podemos usar el dispositivo de datos genérico para eliminar datos usando el nombre de la tabla y un valor clave. La Figura 14-3 muestra un ejemplo de una tabla que utiliza un dispositivo de datos para generar datos de prueba en la base de datos. Llena la tabla

Data Fixture	all_fund			
ticker	name	share_price	class_id	addRow?
BOGUS	Fund ABC	15.786	6	true
BOGEY	Fund DEF	2.413	2	true
BENNY	Fund BEN	1	2	true
WHAAT	Fund What	27.4	3	true
WHYYY	Fund Why	1	6	true
WHERE	Fund Where	1.431	1	true

Figura 14-3 Ejemplo de una tabla que utiliza un dispositivo de datos para generar datos de prueba en la base de datos

“todos los fondos” con las columnas y valores especificados. Es fácil para quienes escribimos casos de prueba completar las tablas con todos los datos que necesitamos.

Tenga en cuenta que a los esquemas que utilizamos para estas pruebas se les han eliminado la mayoría de sus restricciones, por lo que solo tenemos que completar las tablas y columnas pertinentes a la funcionalidad que se está probando. Esto también facilita un poco el mantenimiento. La desventaja es que la prueba es un poco menos realista, pero las pruebas que utilizan otras herramientas verifican la funcionalidad en un entorno realista.

La desventaja de crear datos de prueba de esta manera es que cada vez que se realiza un cambio en la base de datos, como una nueva columna con un valor requerido, todos los datos fijos Las tablas en las pruebas que pueblan esa tabla tendrán que cambiarse. Estas pruebas pueden resultar engorrosas de escribir y mantener, por lo que solo las utilizamos cuando es absolutamente necesario. Intentamos diseñar las pruebas para mantener bajos los costos de mantenimiento. Por ejemplo, el elemento de datos de la Figura 14-3 está en una biblioteca de “inclusión” y puede incluirse en las pruebas que lo necesiten. Digamos que agregamos una nueva columna, “categoría_fondo”. Sólo necesitamos agregarlo a esta tabla de “inclusión”, en lugar de 20 pruebas diferentes que lo usan.

—Lisa

Datos canónicos

Otra alternativa es tener esquemas de prueba que puedan actualizarse rápidamente con datos de una base de datos canónica o de semillas. La idea es que estos datos de semillas sean una muestra representativa de los datos de producción real. Debido a que es una pequeña cantidad de datos, se puede reconstruir rápidamente cada vez que es necesario ejecutar un conjunto de pruebas de regresión.

Este enfoque también aumenta el tiempo que lleva ejecutar las pruebas, pero son sólo unos pocos minutos al inicio de la serie de regresión en lugar de tomar tiempo de cada prueba individual. Las pruebas seguirán siendo más lentas que las pruebas que no acceden a la base de datos, pero serán más rápidas que las pruebas que tienen que completar laboriosamente cada columna en cada tabla.

Los datos canónicos tienen muchos usos. Los probadores y programadores pueden tener los suyos propios. esquema de prueba para actualizar a voluntad. Pueden realizar tanto manuales como automatizados. pruebas sin pisar las pruebas de nadie más. Si los datos se eligen cuidadosamente, los datos serán más realistas que la cantidad limitada de datos que cada prueba puede construir por sí mismo.

Por supuesto, como ocurre con prácticamente todo, hay una desventaja. Datos canónicos Puede ser complicado mantener el ritmo. Cuando necesita nuevos escenarios de prueba, debe identificar los datos de producción que funcionarán, o inventar los datos que necesita y agregarlos. al esquema semilla. Tienes que borrar los datos, enmascarar la identificación de personas reales. características, haciéndolo inocuo por razones de seguridad. Cada vez que agregas

una tabla o columna a la base de datos de producción, debe actualizar sus esquemas de prueba en consecuencia. Es posible que tenga que actualizar los datos sensibles a la fecha cada año o realizar otro tipo de mantenimiento a gran escala. Debe seleccionar cuidadosamente qué tablas deben actualizarse y cuáles no, como las tablas de búsqueda. Si tiene que agregar datos para aumentar la cobertura de la prueba, la actualización tardará más en realizarse, lo que aumentará el tiempo del proceso de compilación que la desencadena. Como hemos estado enfatizando, es importante que sus compilaciones automatizadas brinden retroalimentación de manera oportuna, de modo que las actualizaciones de la base de datos cada vez más largas alarguen su ciclo de retroalimentación. También se pierde la independencia de la prueba con datos canónicos, por lo que si una prueba falla, otras pueden seguir su ejemplo.

Los miembros del equipo de Lisa ejecutaron sus conjuntos de pruebas de GUI y algunas de sus pruebas de regresión funcional contra esquemas actualizaron cada ejecución con datos canónicos. En raras ocasiones, las pruebas fallan inesperadamente debido a una actualización errónea de los datos iniciales. Decidir si "avanzar" los datos, de modo que, por ejemplo, las filas de 2008 se conviertan en filas de 2009, llega a ser un dolor de cabeza. Hasta ahora, el ROI del uso de datos canónicos ha sido aceptable para el equipo. El equipo actual de Janet también utiliza datos iniciales para sus pruebas de "capa intermedia" en compilaciones locales. Funciona bien para obtener comentarios rápidos durante el ciclo de desarrollo. Sin embargo, el entorno de prueba y los entornos de ensayo utilizan una copia migrada de los datos de producción. La desventaja es que las pruebas de regresión sólo se pueden ejecutar en copias locales de la compilación. El riesgo es bajo porque practican "construir una vez, implementar en muchas".

Datos similares a los de producción

La capacidad de probar un sistema que se parezca lo más posible a la producción es esencial para la mayoría de los equipos de desarrollo de software. Sin embargo, ejecutar un conjunto de pruebas de regresión automatizadas contra una copia de una base de datos de producción probablemente sería demasiado lento para ser una retroalimentación útil. Además, no puedes depender de que los datos permanezcan estables cuando traes nuevas copias para mantenerte actualizado.

Generalmente, cuando se habla de pruebas funcionales o de un extremo a otro, un clon de la base de datos de producción es más útil para las pruebas exploratorias manuales.

Las pruebas de estrés, rendimiento y carga, que requieren mucha automatización, necesitan un entorno que simule estrechamente la producción para proporcionar resultados que puedan traducirse en operaciones reales. La usabilidad, la seguridad y la confiabilidad son otros ejemplos de pruebas que necesitan un sistema similar a la producción, aunque es posible que no impliquen mucha automatización.

Siempre hay una compensación; Su base de datos de producción puede ser enorme, por lo que es costosa y lenta, pero proporciona los datos de prueba más precisos disponibles. Si su organización puede permitirse hardware y software para almacenar múltiples copias de datos de producción con fines de prueba, esto es ideal. Las pequeñas empresas pueden

tener limitaciones de recursos que podrían limitar la cantidad de datos que se pueden almacenados en entornos de prueba y preparación. En este caso, tendrás que decidir ¿Cuántos datos de prueba puede admitir y planificar cómo copiar datos suficientemente relevantes? datos para que la prueba sea representativa de lo que se utiliza en la "vida real". O puedes Considera hacer la inversión en hardware, que cada vez es menos costoso. todos los días, para respaldar un entorno de estilo de producción real. De lo contrario, su Los resultados de las pruebas pueden ser engañosos. Como mencionamos con los datos canónicos, es posible que tengas que borrar los datos antes de usarlos.

Migración de datos

La migración de datos debe probarse con una base de datos real. Los scripts de actualización de la base de datos deben ejecutarse con datos reales y con la última versión conocida del esquema de la base de datos.

Probar una migración de base de datos

Paul Rogers, un arquitecto de pruebas de automatización, cuenta esta historia de prueba de una migración de base de datos reveladora [2008]:

Ayer mismo ejecuté una migración de Rails en mi base de datos de prueba. Los desarrolladores lo escribieron, lo probaron y lo verificaron utilizando sus bases de datos de desarrollo. Mi base de datos de prueba era probablemente 20.000 veces más grande. La migración para ellos tomó unos segundos. Para mí, bueno, lo dejé después de tres horas, probablemente al 10% de su finalización. Los programadores necesitaban rehacer su estrategia de migración.

Dudo que esto hubiera aparecido en una base de datos en memoria, por lo que para mí, una base de datos real en este caso fue definitivamente la elección correcta. De hecho, es probable que esto influya en cosas que debemos considerar antes del lanzamiento, como cuánto tiempo lleva una implementación o cuánto tiempo lleva la actualización de la base de datos. Luego podemos usar esto para estimar cuánto tiempo de inactividad necesitaremos para la actualización real.

Este es otro ejemplo de cómo debemos lograr un equilibrio entre pruebas que brinden retroalimentación rápida y pruebas que reflejen de manera realista eventos que podrían ocurrir en producción.

Comprenda sus necesidades

Si comprende el propósito de sus pruebas, podrá evaluar mejor sus necesidades. Por ejemplo, si no necesita probar la velocidad de procedimientos almacenados o consultas SQL directamente, considere herramientas como las bases de datos en memoria, que Funciona como bases de datos reales pero acelera enormemente tus pruebas. Cuando lo necesites Para simular el entorno de producción real, haga una copia de todo el

base de datos de producción, si es necesario. El objetivo es obtener retroalimentación rápida, así que equilibre las pruebas en escenarios realistas con la búsqueda de defectos de la manera más eficiente posible.

EVALUACIÓN DE HERRAMIENTAS DE AUTOMATIZACIÓN

El primer paso para elegir una herramienta de automatización es hacer una lista de todo lo que la herramienta debe hacerlo por usted. Consideremos cómo puedes decidir tu prueba. Requisitos de herramientas.

Identificación de requisitos para su herramienta de automatización

Después de decidir cuál será el próximo desafío de automatización a abordar, piense en sus necesidades de herramientas. ¿Qué herramientas tienes ya? Si necesita más, usted probablemente quiera algo que se integre bien con sus pruebas existentes y su infraestructura de desarrollo. ¿Necesita una herramienta para integrarse fácilmente en el flujo de trabajo? ¿Su hardware actual admitirá la automatización que necesita? Configurar un segundo proceso de compilación para ejecutar pruebas funcionales puede requerir maquinaria adicional.

¿Quién utilizará la herramienta de prueba que espera implementar? ¿Los programadores escribirán los casos de prueba? ¿Quieren sus programadores una herramienta que les ayude a escribirlos? ¿Te sientes cómodo también? ¿Tiene miembros del equipo distribuidos que necesitan colaborar?

¿Quién automatizará y mantendrá las pruebas? Las habilidades que ya tienes en tu equipo son importantes. ¿Cuánto tiempo tienes para instalar una herramienta y aprender a usarla? Si su aplicación está escrita en Java, una herramienta que utilice Java para scripting puede ser el más apropiado. ¿Los miembros del equipo tienen experiencia con herramientas particulares? ¿Existe un equipo de prueba independiente con experiencia en una determinada herramienta? Si está iniciando la transición al desarrollo ágil y ya cuenta con un equipo de automatizadores de pruebas, puede tener sentido aprovechar su experiencia y seguir usando las herramientas que conocen.

Los requisitos de sus herramientas dependen de su entorno de desarrollo. Si estás probando una aplicación web y la herramienta que eliges no es compatible con SSL o AJAX, es posible que tengas un problema. No todas las herramientas de prueba pueden probar servicios web y aplicaciones. Las pruebas de sistemas integrados pueden volver a necesitar herramientas diferentes. El estudio de caso del Capítulo 12, "Resumen de los cuadrantes de prueba", muestra una forma de utilizar Ruby para probar una aplicación integrada.

Por supuesto, el tipo de prueba que estás automatizando es clave. Las pruebas de seguridad probablemente necesiten herramientas altamente especializadas. Existen muchos códigos abiertos y

herramientas de proveedores para el rendimiento, por lo que la tarea de seleccionar una no es abrumadora. A medida que domines un desafío, estarás mejor preparado para el siguiente. Tomó El equipo de Lisa tardó un par de años en desarrollar conjuntos de pruebas de regresión sólidas en el unidad, integración y niveles funcionales. Las pruebas de rendimiento fueron su siguiente zona de dolor. Las lecciones aprendidas de los esfuerzos de automatización anteriores ayudaron hacen un mejor trabajo al identificar los requisitos para una herramienta de prueba, como la facilidad de informes de resultados, compatibilidad con marcos existentes y secuencias de comandos idioma.

Escriba una lista de verificación que capture todos los requisitos de sus herramientas. Algunos podrían entrar en conflicto o contradecirse entre sí: "La herramienta debe ser fácil suficiente para que los clientes puedan especificar pruebas" o "Las pruebas deben ser fáciles de realizar". automatizar." Anótalos para que puedas encontrar el equilibrio adecuado. Entonces empezar haciendo tu investigación.

Una herramienta a la vez

Necesitará diferentes herramientas para diferentes propósitos. Implementar nuevas herramientas y aprender la mejor manera de usarlas puede resultar abrumador bastante rápido. Pruebe una herramienta a la vez, abordando su mayor área de dolor. Déle suficiente tiempo para un juicio justo y evalúe los resultados. Si esta funcionando para usted, domine esa herramienta antes de pasar a la siguiente área de dolor y la siguiente herramienta. La multitarea puede funcionar en algunas situaciones, pero las nuevas tecnologías exigen toda la atención.

Cuando haya elegido una herramienta para abordar una necesidad particular, dé un paso atrás y mira qué más necesitas. ¿Cuál es el próximo desafío de la automatización al que se enfrenta? ¿Tu equipo? ¿La herramienta que acabas de seleccionar para otro propósito funcionará para ese? ¿También necesitas o necesitas iniciar un nuevo proceso de selección?

 La bibliografía contiene sitios web que ayudan con la búsqueda y evaluación de herramientas.

Si ha decidido buscar herramientas fuera de su propia organización, la primera El primer paso es encontrar tiempo para probar algunos. Comience con una investigación básica: Internet búsquedas, artículos y otras publicaciones sobre herramientas y listas de correo son Buenos lugares para tomar ideas. Compile una lista de herramientas a considerar. Si tu equipo usa una herramienta wiki o foro en línea, publique información sobre herramientas e inicie una discusión sobre los pros y los contras.

Presupuestar tiempo para evaluar herramientas. Algunos equipos tienen un "sprint de ingeniería" o "iteración de refactorización" cada pocos meses donde, en lugar de entregar historias priorizados por la empresa, se ponen a trabajar para reducir la deuda técnica, actualizar las versiones de las herramientas y probar nuevas herramientas. Si tu equipo no tiene estos todavía, presente un caso a su gerencia para obtenerlos. Reduciendo su tecnología

deuda financiera y establecer una buena infraestructura de pruebas mejorará su velocidad en el futuro y tiempo libre para pruebas exploratorias. si nunca lo has hecho Es hora de hacer que el código sea más fácil de mantener o actualizar las herramientas, la deuda técnica aumentará. arrastre hacia abajo su velocidad hasta que se detenga.

Cuando tenga una lista de herramientas que puedan satisfacer sus necesidades, limite la posibilidades hasta una o dos, aprenda a usar cada una lo suficientemente bien como para Pruébalo y haz un pico: prueba un escenario simple pero representativo que puedas tirar a la basura. Evaluar los resultados frente a los requisitos. Utilice retrospectivas para considerar los pros y los contras.

¿Qué recursos necesita para implementar y utilizar la herramienta? ¿Qué impacto ¿Qué tendrá la herramienta en la productividad y velocidad del equipo? ¿Qué riesgos tiene? ¿Pose? ¿Qué le permitirá hacer a largo plazo que no pueda hacer ahora?

Elija su mejor candidato y comprométase a probarlo durante un período de tiempo. el tiempo suficiente para adquirir cierta competencia con él. Asegúrese de probar todas las funciones de misión crítica. Por ejemplo, si su aplicación usa mucho Ajax, asegúrese de poder automatizar las pruebas utilizando la herramienta. En retrospectivas, mire qué funcionó y qué no. Esté abierto a la idea de que podría no estar bien. y que hay que tirarlo y empezar de nuevo. No sientas que tienes que seguir Continúe con la herramienta porque ya ha invertido mucho en ella.

Todos sabemos que no existe una “solución milagrosa” que pueda resolver toda su automatización. problemas. Baja tus expectativas y abre tu mente. Soluciones creativas confiar tanto en el arte como en la ciencia.

La historia de Lisa

El Capítulo 11, “Crítica del producto mediante pruebas orientadas a la tecnología”, muestra un ejemplo de los resultados producidos por la herramienta de prueba de rendimiento elegida.

Al realizar la búsqueda de herramientas de prueba de rendimiento, recurrimos a una lista de correo de pruebas ágiles en busca de sugerencias. Muchas personas ofrecieron sus experiencias y algunas incluso se ofrecieron a ayudar a aprender e implementar una herramienta. Buscamos una herramienta que utilizara Java para secuencias de comandos, tuviera una curva de aprendizaje mínima y presentara los resultados en un formato gráfico útil. Enumeramos las herramientas y sus ventajas y desventajas en la wiki del equipo. Presupuestamos tiempo para las pruebas. El compañero de trabajo de Lisa, Mike Busse, probó los dos mejores candidatos y mostró los aspectos más destacados al resto del equipo. Se eligió una herramienta por consenso del equipo y ha demostrado ser una buena opción.

—Lisa

Elegir herramientas

Tenemos suerte de contar con una gama ya amplia y un conjunto de herramientas en constante crecimiento para elija entre: herramientas propias, de código abierto, de proveedores o una combinación de

cualquiera, son todas alternativas viables. Con tantas opciones, el truco está en saber dónde buscar y encontrar tiempo para probar herramientas y ver si se ajustan a sus necesidades. Como no podemos predecir el futuro, puede resultar difícil juzgar el retorno de la inversión (ROI) de cada solución potencial, pero un enfoque iterativo para evaluarlas ayuda llegar al correcto.

¿Deberías cultivar el tuyo propio?

¿Su aplicación presenta desafíos de prueba únicos, como integración ¿Software o integración con sistemas externos? ¿Los miembros del equipo tienen la Habilidades, tiempo e inclinación para escribir su propio marco de prueba o construir uno. ¿Además de una herramienta de código abierto existente? Si es así, es posible que se utilicen herramientas de prueba locales. el mejor ajuste.

Un resultado feliz (o quizás un importante factor de éxito) del desarrollo ágil es que muchos programadores están "infectados por pruebas". Las herramientas de desarrollo actuales y Los lenguajes facilitan la construcción de marcos de automatización. Rubí, Groovy, Rieles, y muchos lenguajes y marcos se prestan a la automatización. Se pueden aprovechar las herramientas de código abierto existentes, como Fit y HtmlUnit, con marcos personalizados creados sobre ellas.

Las herramientas caseras tienen muchas ventajas. Definitivamente son amigables para los programadores. Si su equipo está escribiendo sus propios marcos de automatización, estarán personalizado con precisión para las necesidades de sus equipos de desarrollo y de clientes, e integrado con su proceso de construcción existente y otra infraestructura—y puede hacerlos tan fáciles de ejecutar e interpretar los resultados como necesite.

Por supuesto, que sea de cosecha propia no significa que sea gratuito. Un equipo pequeño puede no tener la ancho de banda para escribir y soportar herramientas, así como desarrollar código de producción. A Una organización grande con requisitos únicos puede ser capaz de crear una equipo de especialistas en automatización que pueden colaborar con evaluadores, clientes, programadores y otros. Si sus necesidades son tan únicas que no existe ninguna herramienta los apoya, la cosecha local puede ser su única opción.

Herramientas de código abierto

Muchos equipos que escribieron sus propias herramientas las han puesto generosamente a disposición a la comunidad de código abierto. Debido a que estas herramientas fueron escritas por programadores infectados cuyas necesidades no fueron satisfechas por las herramientas del proveedor, son generalmente liviano y apropiado para el desarrollo ágil. Muchos de estos Las herramientas se desarrollan primero como prueba y puede descargar el conjunto de pruebas junto con el código fuente, haciendo la personalización más fácil y segura. Estas herramientas tienen un Amplio atractivo, con características útiles tanto para programadores como para evaluadores. El

El precio es correcto, aunque es importante recordar que el precio de compra es sólo una fracción del costo de cualquier herramienta.

No todas las herramientas de código abierto están bien documentadas y la capacitación puede ser un problema. Sin embargo, vemos seminarios y tutoriales sobre el uso de estas herramientas en muchas conferencias y reuniones de grupos de usuarios. Algunas herramientas de código abierto tienen excelentes usuarios manuales e incluso tener tutoriales en línea y clases programadas disponibles.



Consulte el Capítulo 9,
"Kit de herramientas para
Orientado a los negocios
Pruebas que respaldan al
equipo", para más
información sobre open
prueba de fuente automática
herramientas de formación.

Si está considerando una solución de código abierto, busque un desarrollador activo y comunidad de usuarios. ¿Existe una lista de correo con mucho ancho de banda? Son nuevos ¿Funciones publicadas con frecuencia? ¿Hay alguna manera de informar errores? ¿Alguien los soluciona? ¿A ellos? Algunas de estas herramientas tienen mejor soporte y respuesta más rápida a los errores. que las herramientas del proveedor. ¿Por qué? Las personas que los escriben también los utilizan, y necesitan esas características para probar sus propios productos.

Herramientas del proveedor

Las herramientas comerciales se perciben como una apuesta segura. Es difícil criticar a alguien por seleccionar una herramienta conocida que existe desde hace años. Es probable que vienen con manuales, soporte y capacitación. Para probadores u otros usuarios que Sin experiencia técnica, el avance inicial podría ser más rápido. Algunos son bastante robusto y rico en funciones. Es posible que su empresa ya tenga uno y tenga un equipo de especialistas que saben cómo utilizarlo.

Aunque cambian con los tiempos, las herramientas de los proveedores históricamente han sido poco amigables para los programadores. Tienden a utilizar lenguajes de scripting propietarios que Los programadores no quieren perder tiempo aprendiendo. También tienden a ser pesados. Los guiones de prueba pueden ser frágiles y romperse fácilmente con cambios menores en el aplicación y costoso de mantener. La mayoría de estas herramientas están grabando guiones para su posterior reproducción. Los guiones de grabación/reproducción son notoriamente costoso desde el punto de vista del mantenimiento.



Consulte la bibliografía
para una discusión
completa de Elisabeth.

hendrickson
en esta asignatura.

Elisabeth Hendrickson [2008] señala que herramientas especializadas como estas puede crear la necesidad de especialistas en automatización de pruebas. Silos como estos pueden funcionar contra equipos ágiles. Necesitamos herramientas que faciliten la prueba primero, en lugar de la última prueba desarrollo. Las herramientas de prueba no deberían obstaculizar el cambio.

Si ya cuenta con personas expertas en una herramienta del proveedor y el uso de una herramienta que Puede ser utilizado sólo por un subconjunto del equipo de desarrollo o un equipo separado.

A partir del desarrollo, una herramienta de proveedor podría tener mucho sentido. Los dos primeros XP de Lisa Los equipos utilizaron una herramienta de proveedor con cierto grado de éxito.

Al momento de escribir este artículo, están surgiendo mejores IDE y herramientas de prueba funcionales.

Estos facilitan las tareas de mantenimiento de pruebas con funciones como búsqueda/reemplazo global. Girar es un ejemplo de una herramienta implementada como una colección de complementos para Eclipse IDE, para que pueda aprovechar potentes funciones de edición y refactorización.

Herramientas ágiles



Parte III, "Uso de cuadrantes de pruebas ágiles"

y, en particular, el Capítulo 9, "Kit de herramientas para pruebas empresariales que respaldan al equipo", contiene ejemplos de herramientas de automatización de pruebas que funcionan bien en proyectos ágiles

Elisabeth Hendrickson [2008] enumera algunas características de una prueba ágil eficaz herramientas de automatización. Estas herramientas deberían:

- Apoyar el inicio inmediato del esfuerzo de automatización de pruebas, utilizando un enfoque de prueba primero.
- Separar la esencia de la prueba de los detalles de implementación.
- Apoyar y fomentar buenas prácticas de programación para la parte del código de la automatización de pruebas.
- Admitir la escritura de código de automatización de pruebas utilizando lenguajes reales, con IDE reales
- fomentar la colaboración

IMPLEMENTANDO LA AUTOMATIZACIÓN

Mientras evalúa las herramientas, piense en la rapidez con la que debe abordarse su necesidad de automatización de máxima prioridad. ¿Dónde obtendrás el apoyo para ayudar?

¿Implementarlo? ¿Qué formación necesita el equipo y cuánto tiempo será necesario? disponible para dedicarle? ¿Qué tan rápido tienes que mejorar esta herramienta?

Tenga en cuenta todas estas limitaciones cuando busque herramientas. Tu podrías Tendrá que conformarse con una herramienta menos robusta de la que realmente desea para obtener vitalidad. automatización en el corto plazo. Recuerda que nada es permanente.

Puede desarrollar su esfuerzo de automatización paso a paso. Muchos equipos experimentan intentos fallidos antes de encontrar la combinación adecuada de herramientas, habilidades, e infraestructura.

Selenio en acción

Joe Yakich, un ingeniero de software con experiencia en automatización de pruebas, describe cómo un equipo con el que trabajó implementó un esfuerzo de automatización de pruebas con Selenium, una herramienta de automatización de pruebas de código abierto.

La empresa de software para la que trabajaba (llamémosla XYZ Corp) tenía un problema. El producto, una aplicación web de nivel empresarial, era una oferta potente y madura. Los proyectos de desarrollo se gestionaron utilizando

Agile y Scrum, y un talentoso grupo de ingenieros produjeron nuevas funciones rápidamente. La empresa estaba creciendo de manera constante.

¿Entonces, Cual fue el problema? XYZ se enfrentaba a un futuro en el que los esfuerzos de prueba de software tal vez no pudieran seguir el ritmo de los esfuerzos de desarrollo.

Los problemas de calidad del software podrían ralentizar la adopción del producto o, peor aún, todavía—hacer que los clientes existentes busquen en otra parte.

La automatización de pruebas parecía una forma obvia de mitigar estos riesgos y XYZ era plenamente consciente de ello. De hecho, habían intentado crear un conjunto de automatización de pruebas dos veces antes y fracasaron.

La tercera vez, XYZ optó por utilizar Selenium RC, impulsado por el lenguaje de programación Ruby. Selenium RC (RC significa "Control remoto") es una herramienta para la automatización de pruebas. Selenium RC consta de un componente de servidor y bibliotecas de cliente. El componente del servidor Java actúa como un proxy HTTP, lo que hace que Selenium Core JavaScript parezca originarse en el servidor web de la aplicación bajo prueba (AUT). El servidor puede iniciar y detener sesiones del navegador (los navegadores compatibles incluyen casi todos los navegadores modernos, incluidos Internet Explorer, Firefox y Safari) e interpretar comandos para interactuar con elementos como botones, enlaces y campos de entrada.

Las bibliotecas cliente permiten escribir scripts de prueba en Java, .NET, Perl, Python y Ruby.

Nuestro equipo eligió Ruby porque es un lenguaje interpretado, dinámico y puramente orientado a objetos con una sintaxis elegante, expresa y concisamente poderosa. Lo más importante es que Ruby es una herramienta ideal para la creación de un lenguaje específico de dominio (DSL). Ruby es lo suficientemente maleable como para que el programador elija primero la estructura y la sintaxis del DSL y luego diseñe una implementación, a diferencia de un lenguaje más rígido que podría imponer restricciones a esa libertad. Uno de nuestros objetivos era crear un marco de automatización (un DSL) que ocultara detalles complejos. Queríamos poder decir cosas como

editor.guardar

en nuestras pruebas en lugar de

```
s.click("//table[@class='edit']/tbody/tr[0]/img[@src='save.gif']")
```

El primero no sólo es más legible, sino que también es mucho más fácil de mantener. La expresión XPath en este último se puede colocar en un método de biblioteca para llamarla según sea necesario. El uso de un DSL que emplea los sustantivos y verbos de la aplicación permite que un ingeniero que escribe una prueba se concentre en la prueba, no en la complejidad subyacente de interactuar con los controles en pantalla.

XYZ creó un equipo de automatización para crear el marco y las pruebas. La creación del marco en sí fue una tarea técnicamente desafiante y que requirió mucho tiempo.

tarea. Algunas de las clases marco eran lo suficientemente complicadas como para justificar sus propias pruebas unitarias. Después de construir una cantidad suficiente de marco de prueba, comenzamos a trabajar en pruebas de aplicaciones reales, utilizando la biblioteca Ruby RSpec. RSpec es en sí mismo un DSL para especificaciones de prueba. Uno de sus puntos fuertes es el uso de declaraciones declarativas simples para describir el comportamiento y las expectativas. Se podría, por ejemplo, escribir una prueba usando la declaración

"Un usuario debería poder acceder a `passwordsamuel` editor haciendo clic

Ahorra"

completando el cuerpo de la prueba con llamadas al marco de prueba basado en Selenium que habíamos creado.

Casi un año después, habíamos automatizado casi dos mil casos de prueba.

Aunque la mayor parte de la aplicación estaba cubierta por la automatización, otras partes de la aplicación requirieron pruebas manuales; nos vimos obligados a tomar decisiones y priorizar nuestros esfuerzos. Cada semana, el conjunto de pruebas tardaba más en ejecutarse que la semana anterior; Ahora nos llevó casi seis horas completarlo y habíamos empezado a pensar en realizar pruebas en paralelo. Aún no habíamos logrado expandir nuestras pruebas a todos los navegadores compatibles con la aplicación. El entusiasmo que generaba la automatización había disminuido un poco y nos pareció necesario gestionar cuidadosamente las expectativas, tanto con la alta dirección como con otros ingenieros. A pesar de estos problemas, Selenium fue una clara victoria, ya que si no hubiéramos invertido mucho en la automatización de pruebas, las pruebas en XYZ habrían requerido contratar un ejército de ingenieros de pruebas (lo que habría sido prohibitivamente costoso incluso si hubiéramos podido encontrar suficientes solicitantes calificados).

No todo se puede automatizar, por razones presupuestarias o técnicas. Además, las pruebas exploratorias son invaluables y no deben descuidarse. Sin embargo, cabe señalar que estos inconvenientes son compartidos por todas las demás herramientas de automatización de pruebas disponibles actualmente, y la mayoría de las otras herramientas de automatización que pueden rivalizar con la destreza de automatización de Selenium son productos comerciales que no pueden igualar su precio: gratis.

Las buenas prácticas de desarrollo son clave para cualquier esfuerzo de automatización. Utilice un enfoque orientado a objetos. A medida que crea su biblioteca de objetos de prueba, agregar nuevas pruebas se vuelve más fácil. Un lenguaje específico de dominio ayuda a que las pruebas empresariales sean comprensibles para los clientes, al tiempo que reduce los costos de redacción y mantenimiento de scripts de prueba automatizados.

Un buen diseño orientado a objetos no es la única clave para crear un conjunto de pruebas automatizadas mantenibles y que den frutos. También es necesario realizar las pruebas con frecuencia suficiente para obtener la retroalimentación que su equipo necesita. Cualquier herramienta que elijamos debe integrarse con nuestro proceso de construcción. Deben llegar resultados fáciles de interpretar a nosotros automáticamente.

Las herramientas que elegimos tienen que funcionar en nuestras plataformas y deben compartirse y funcionar bien con nuestras otras herramientas. Tenemos que modificarlos continuamente para ayudar con nuestros problemas actuales. ¿La construcción se rompe todos los días? Tal vez necesitemos conectar nuestros resultados a un semáforo real para que el equipo sea consciente de su estado. ¿Falló una prueba empresarial? Debería quedar claro exactamente qué falló y dónde. No tenemos tiempo extra para dedicarlo a aislar problemas.

Estas preocupaciones son una parte esencial del panorama, pero todavía son sólo una parte del panorama. Necesitamos herramientas que nos ayuden a diseñar entornos de prueba que imiten la producción. Necesitamos formas de mantener estos entornos de prueba independientes, sin verse afectados por los cambios que puedan estar realizando los programadores.

Crear una infraestructura de pruebas puede ser una gran inversión, pero es una que nuestro equipo ágil debe hacer para dar un salto en la automatización de pruebas. Es necesario identificar e implementar hardware, software y herramientas. Dependiendo de los recursos de su empresa, este podría ser un proyecto a largo plazo. Piense en formas de afrontar la situación a corto plazo, mientras planifica cómo montar la infraestructura que realmente necesita para minimizar el riesgo, maximizar la velocidad y ofrecer el mejor producto posible.

GESTIÓN DE PRUEBAS AUTOMATIZADAS

Digamos que necesitamos una manera de encontrar la prueba que verifica un escenario particular, para comprender qué hace cada prueba y saber qué parte de la aplicación verifica. Quizás necesitemos satisfacer un requisito de auditoría para la trazabilidad de cada requisito hasta su código y pruebas. Las pruebas automatizadas deben mantenerse y controlarse de la misma manera que el código fuente de producción. Cuando etiqueta su código de producción para su lanzamiento, las pruebas que verificaron esa funcionalidad deben ser parte de la etiqueta.

A continuación se muestra un ejemplo en el que esto resulta útil. Acabamos de encontrar un problema en el código en desarrollo. ¿Es un problema nuevo o ha estado acechando en el código durante un tiempo y de alguna manera no fue detectado en la prueba? Podemos implementar la etiqueta que está en producción, intentar reproducir el problema e investigar por qué las pruebas no lo detectaron. El equipo de Lisa recientemente tuvo una situación en la que la suite de regresión omitió un error porque faltaba una restricción de la base de datos en el esquema de prueba. Ese tipo de problema es difícil de identificar si no vincula las versiones del código de prueba con las versiones del código de producción.

Organización de pruebas

Muchas herramientas vienen con sus propios medios de organización. Por ejemplo, Fit-Nesse viene con su propia wiki, con una organización jerárquica y funciones integradas.

control de versiones. Al momento de escribir este artículo, FitNesse está comenzando a brindar soporte para Herramientas de control de código fuente como Subversion. Guiones escritos en otras herramientas de prueba, como Watir y Canoo WebTest, pueden y deben mantenerse dentro del

El mismo sistema de control de código fuente que el código de producción, al igual que las pruebas unitarias.

Organización de pruebas con el proyecto bajo prueba

Preguntamos a algunos expertos en pruebas ágiles cómo gestionan las pruebas. Dierk Konig, fundador y director de proyectos de Canoo WebTest, explicó cómo sus equipos han gestionado sus pruebas automatizadas para satisfacer las necesidades tanto de los equipos de desarrollo como de los clientes.

Siempre organizamos nuestras pruebas junto con el proyecto bajo prueba. Es decir, las fuentes de prueba se almacenan junto con las fuentes del proyecto exactamente en el mismo repositorio, utilizando los mismos mecanismos para el control de revisión, etiquetado y uso compartido de la base de prueba.

WebTest viene con un diseño estándar sobre cómo organizar pruebas y datos de pruebas en directorios. Puede adaptar esto a cualquier estructura que desee, pero la "convención sobre configuración" muestra su fuerza aquí. En proyectos grandes, cada subproyecto mantiene su propia base de pruebas en un "webtest" subdirectorio que sigue la convención.

Cada vez que un cliente no seguía este enfoque, la experiencia era muy dolorosa para todas las personas involucradas. Hemos visto enormes bases de datos de descripciones de pruebas que ni siquiera incluían un control de revisión adecuado (es decir, donde se podían, por ejemplo, ver diferencias con versiones anteriores o quién cambió qué prueba y por qué motivo).

Tenga en cuenta que las pruebas se componen de módulos para que pueda eliminar la duplicación del código de prueba; de lo contrario, el mantenimiento te matará.

Y antes de cambiar cualquier módulo, es necesario saber dónde se utiliza.

En resumen: asegúrese de que el maestro de sus pruebas y sus datos de prueba estén en un formato de texto versionado junto con su código bajo prueba.

El personal no técnico (por ejemplo, administración, control de calidad) puede requerir más información de alto nivel sobre la cobertura de la prueba, los últimos resultados de la prueba o incluso los medios para desencadenar una ejecución de prueba. No permita que estos requisitos válidos socaven el enfoque de ingeniería para la automatización de pruebas. En su lugar, escriba pequeñas herramientas, por ejemplo, aplicaciones de informes basadas en web, que aborden estas necesidades.

La capacidad de los clientes para acceder a información sobre las pruebas es tan importante como la capacidad de mantener coordinados el código de prueba y de producción. Como señaló Dierk, es posible que no puedas hacer todo esto con la misma herramienta.

La gestión de pruebas ayuda a su equipo a responder preguntas como las siguientes:

¿Qué casos de prueba se han automatizado?

¿Cuáles todavía necesitan automatización?

¿Qué pruebas se ejecutan actualmente como parte de un conjunto de regresión?

¿Qué pruebas cubren qué áreas funcionales?

¿Cómo está diseñada para funcionar la función XYZ?

¿Quién escribió este caso de prueba? ¿Cuando? ¿Quién lo cambió por última vez?

¿Cuánto tiempo hace que esta prueba forma parte del conjunto de regresión?

Debido a que una de las razones principales por las que escribimos pruebas es para guiar el desarrollo, necesitamos organizar las pruebas para que todos los miembros del equipo puedan encontrar las pruebas apropiadas para cada historia e identificar fácilmente qué funcionalidad cubren las pruebas.

Debido a que utilizamos pruebas como documentación, es fundamental que cualquier persona en el equipo de desarrollo o del cliente puede encontrar una prueba particular rápidamente cuando hay una pregunta sobre cómo debería comportarse el sistema. Podríamos necesitar múltiples herramientas para satisfacer diferentes objetivos de gestión de pruebas.

Es fácil perder el control de los scripts de prueba. Cuando una prueba falla, es necesario identificar el problema rápidamente. Es posible que necesite saber qué cambios se han realizado recientemente al script de prueba, lo cual es fácil con el historial disponible en una fuente sistema de control de código. Su equipo de clientes también necesita una forma de realizar un seguimiento progreso del proyecto, para comprender qué parte del código está cubierto con pruebas, y posiblemente para ejecutar pruebas ellos mismos. Sistemas de gestión de pruebas, como las pruebas, ellos mismos, deben promover la comunicación y la colaboración entre el equipo, miembros y entre diferentes equipos.

Transparencia de prueba

Declan Whelan, desarrollador de software y coach ágil, utiliza un enfoque de gestión de pruebas diseñado para mantener las pruebas visibles para los evaluadores, desarrolladores, administradores y otros equipos.

Tratamos todos los artefactos de prueba de la misma manera que el código fuente desde una perspectiva organizativa y de control de revisiones. Usamos Subversion y cualquiera que quiera ejecutar o editar las pruebas simplemente las verifica.

Las últimas pruebas de ajuste están disponibles en Confluence Wiki. Hicimos esto para respaldar la colaboración (el equipo está distribuido) y aprovechar las sólidas capacidades de Confluence. Tener las pruebas visibles en la wiki también fue útil para otros, como gerentes y otros equipos, que no querían consultarlas desde el repositorio.

Antes de esto, el equipo de control de calidad mantenía casos de prueba en una unidad a la que no podía acceder nadie fuera del control de calidad. Esto significaba que los desarrolladores no podían ver fácilmente lo que se estaba probando. Hacer que las pruebas fueran visibles, transparentes y respaldadas por un sistema de control de versiones (Subversion) realmente ayudó a romper las barreras entre los desarrolladores y los evaluadores del equipo.

Asegúrese de que sus pruebas se administren con un control de versiones sólido, pero auméntelo con una forma en que todos puedan usar las pruebas de manera que impulsen el proyecto hacia adelante y garanticen que se entregue el valor correcto.

Organización de los resultados de las pruebas

Todos los involucrados en la entrega de software necesitan un fácil acceso a las pruebas y a sus resultados. Otro aspecto de la gestión de pruebas es realizar un seguimiento de qué pruebas se de iteraciones anteriores y necesitan seguir pasando, frente a pruebas que conducen desarrollo en la iteración actual y es posible que aún no haya pasado. Un proceso continuo de integración y construcción ejecuta pruebas para obtener comentarios rápidos sobre el progreso. y detectar fallos de regresión. La Figura 14-4 muestra un ejemplo de un resultado de prueba. informe comprensible de un vistazo. Una prueba falló y la causa del El fracaso está claramente indicado.

Si está impulsando el desarrollo con pruebas y algunas de esas pruebas aún no se pasan, esto no debería fallar en una compilación. Algunos equipos, como el de Lisa, simplemente mantienen nuevas pruebas fuera del proceso de integración y construcción hasta que pasen por primera vez. tiempo. Después de eso, siempre tienen que pasar. Otros equipos usan reglas en la construcción. proceso mismo para ignorar las fallas de las pruebas escritas para cubrir el código actualmente siendo desarrollado.

Al igual que con cualquier herramienta de automatización de pruebas, puede resolver sus problemas de gestión de pruebas con sistemas comerciales, de código abierto o de cosecha propia. Los mismos criterios que describimos en la sección sobre evaluación de herramientas de prueba se pueden aplicar a seleccionar un enfoque de gestión de pruebas.

La gestión de pruebas es otra área más en la que se aplican los valores y principios ágiles, junto con el enfoque de todo el equipo. Empiece de forma sencilla. Experimentar en pequeños pasos hasta que encuentre la combinación correcta de control de código fuente, repositorios y administración de compilación que mantenga las pruebas y el código de producción en orden. sincronizar. Evalúe su enfoque de gestión de pruebas con frecuencia y asegúrese de que se adapte a todos los diferentes usuarios de las pruebas. Identifique qué está funcionando y lo que falta y planificar tareas o incluso historias para probar otra herramienta o proceso para llenar cualquier vacío. Recuerde mantener la gestión de pruebas liviana y fácil de mantener para que todos la utilicen.

Viewing all results for testcase: Ideal - Declined deposit				Bugs	Test Date/time	Details	Parent Test Set
Result	Environment	Build	Comments				
Pass			Member_Login (Member_Ideal_transactions.rb:56): Everything was good Member_NordeaDeposit (Member_Ideal_transactions.rb:57): correctly received a TRANSACTION_DECLINED exception		01-Aug-2008 10:36	view	view
Pass			Member_Login (Member_Ideal_transactions.rb:56): Everything was good Member_NordeaDeposit (Member_Ideal_transactions.rb:57): correctly received a TRANSACTION_DECLINED exception		31-Jul-2008 17:45	view	view
Pass			Member_Login (Member_Ideal_transactions.rb:56): Everything was good Member_NordeaDeposit (Member_Ideal_transactions.rb:57): correctly received a TRANSACTION_DECLINED exception		31-Jul-2008 13:17	view	view
Fail			Member_Login (Member_Ideal_transactions.rb:56): Everything was good Member_NordeaDeposit (Member_Ideal_transactions.rb:57): received RuntimeError which was completely unexpected		30-Jul-2008 14:11	view	view
Pass			Member_Login (Member_Ideal_transactions.rb:56): Everything was good Member_NordeaDeposit (Member_Ideal_transactions.rb:57): correctly received a TRANSACTION_DECLINED exception		28-Jul-2008 17:42	view	view
Pass			Member_Login (Member_Ideal_transactions.rb:56): Everything was good Member_NordeaDeposit (Member_Ideal_transactions.rb:57): correctly received a TRANSACTION_DECLINED exception		28-Jul-2008 16:10	view	view
Pass			Member_Login (Member_Ideal_transactions.rb:56): Everything was good Member_NordeaDeposit (Member_Ideal_transactions.rb:57): correctly received a TRANSACTION_DECLINED exception		26-Jul-2008 16:41	view	view
Pass			Member_Login (Member_Ideal_transactions.rb:56): Everything was good Member_NordeaDeposit (Member_Ideal_transactions.rb:57): correctly received a TRANSACTION_DECLINED exception		25-Jul-2008 16:42	view	view

Figura 14-4 Resultados de pruebas de una herramienta de gestión de pruebas propia

Gestión de pruebas para comentarios

Megan Sumrell, entrenadora y entrenadora ágil, describe cómo su equipo coordina su proceso de construcción y prueba para obtener una retroalimentación óptima.

Creamos un conjunto de pruebas de FitNesse para cada sprint. En esa suite, creamos un subwiki para cada historia de usuario que contiene sus pruebas. Según sea necesario, creamos una configuración y desmontaje por prueba o conjunto. Si por alguna razón no completamos una historia de usuario en el sprint, trasladamos las pruebas a la suite para el sprint en la que completamos la historia.

Escribimos la siguiente regla en nuestra compilación: si alguna de las suites del sprint anterior falla, entonces la compilación se interrumpe. Sin embargo, si las pruebas en el sprint actual fallan, no falle la compilación.

Cada conjunto de pruebas tiene un largo proceso de configuración, por lo que cuando nuestras pruebas de FitNesse comenzaron a tardar más de 10 minutos en ejecutarse, nuestra integración continua se volvió demasiado lenta. Usamos enlaces simbólicos para crear un conjunto de pruebas que sirven como pruebas de humo y se ejecutan como parte de nuestro proceso de construcción de integración continua. Realizamos el conjunto completo de pruebas FitNesse en una máquina independiente. Lo configuramos para verificar el servidor de compilación cada cinco minutos. Si existiera una nueva compilación, la cancelaría y ejecutaría todo el conjunto de pruebas de FitNesse. Cuando terminaba, volvía a comprobar el servidor de compilación cada cinco minutos y, una vez que existía una nueva compilación, repetía el proceso.

El equipo de Megan aprovechó las funciones integradas en sus herramientas, como enlaces simbólicos, para organizar conjuntos de pruebas de FitNesse para diferentes propósitos: uno para una prueba de humo y otros para una prueba de regresión completa. Los miembros del equipo obtienen comentarios inmediatos de las pruebas de humo y sabrán en una hora si hay un error que las pruebas de humo no detectaron.

EMPEZAR

No tenga miedo de poner algo (cualquier cosa) en su lugar, incluso si es algo deficiente. El factor más importante para el éxito es simplemente empezar. muchos, si No la mayoría, los equipos exitosos comenzaron con un proceso deficiente pero lograron convertir un proceso inadecuado en algo verdaderamente esencial para el éxito del equipo, una pieza a la vez. Como ocurre con tantos aspectos de las pruebas ágiles, mejorar en pequeños incrementos es la clave del éxito.

Si no comienza por algún lado, nunca logrará impulsar la automatización. Conseguir Todo el equipo se reúne y comienza un experimento. Sin el nivel adecuado de automatización de pruebas, su equipo no puede hacer su mejor trabajo. Necesita la automatización de pruebas adecuada para ofrecer valor empresarial con frecuencia. Dentro de uno o dos años, podrás Me pregunto por qué pensaba que la automatización de pruebas era tan difícil.

RESUMEN

En este capítulo, consideraremos cómo aplicar valores, principios y prácticas ágiles para desarrollar una estrategia de automatización. Hablamos de los siguientes temas. relacionado con la automatización:

Utilice los cuadrantes de pruebas ágiles para ayudar a identificar dónde necesita automatización de pruebas y cuándo la necesitará.

La pirámide de automatización de pruebas puede ayudar a su equipo a realizar las inversiones adecuadas en automatización de pruebas que generarán mayores beneficios.

Aplique valores, principios y prácticas ágiles para ayudar a su equipo a impulsar la automatización de pruebas.

Las tareas repetitivas, los procesos continuos de integración y construcción, las pruebas unitarias, las pruebas funcionales, las pruebas de carga y la creación de datos son buenos candidatos para la automatización.

Las pruebas del cuadrante 3, como las pruebas de usabilidad y las pruebas exploratorias, pueden beneficiarse de cierta automatización para configurar escenarios de prueba y analizar los resultados, pero los instintos humanos, el pensamiento crítico y la observación no se pueden automatizar.

Un enfoque simple de todo el equipo, el uso de retroalimentación iterativa y el tiempo suficiente pueden ayudarlo a comenzar a encontrar una buena solución.

Al desarrollar una estrategia de automatización, comience con el área más problemática, considere un enfoque de múltiples capas y esfuérzese por revisar y mejorar continuamente su estrategia en lugar de lograr la perfección desde el principio.

Considere el riesgo y el retorno de la inversión al decidir qué automatizar.

Tómese el tiempo para aprender haciendo; aplicar prácticas de codificación ágil a las pruebas.

Decida si puede simplemente crear entradas en memoria o si necesita datos de estilo de producción en una base de datos.

Suministre datos de prueba que permitan que las pruebas sean independientes, reejecutables y lo más rápidas posible.

Aborde una herramienta a la vez, identifique sus requisitos y decida qué tipo de herramienta elegir o construir que se ajuste a sus necesidades.

Utilice buenas prácticas de desarrollo para la automatización de pruebas y tómese el tiempo para un buen diseño de pruebas.

Las herramientas automatizadas deben encajar en la infraestructura de desarrollo del equipo.

Pruebas automatizadas de control de versiones junto con el código de producción que verifican.

Una buena gestión de pruebas garantiza que las pruebas puedan proporcionar una documentación eficaz del sistema y del progreso del desarrollo.

Comience hoy con la automatización de pruebas.

Esta página se dejó en blanco intencionalmente.

Parte V

UNA ITERACIÓN EN LA VIDA DE UN PROBADOR

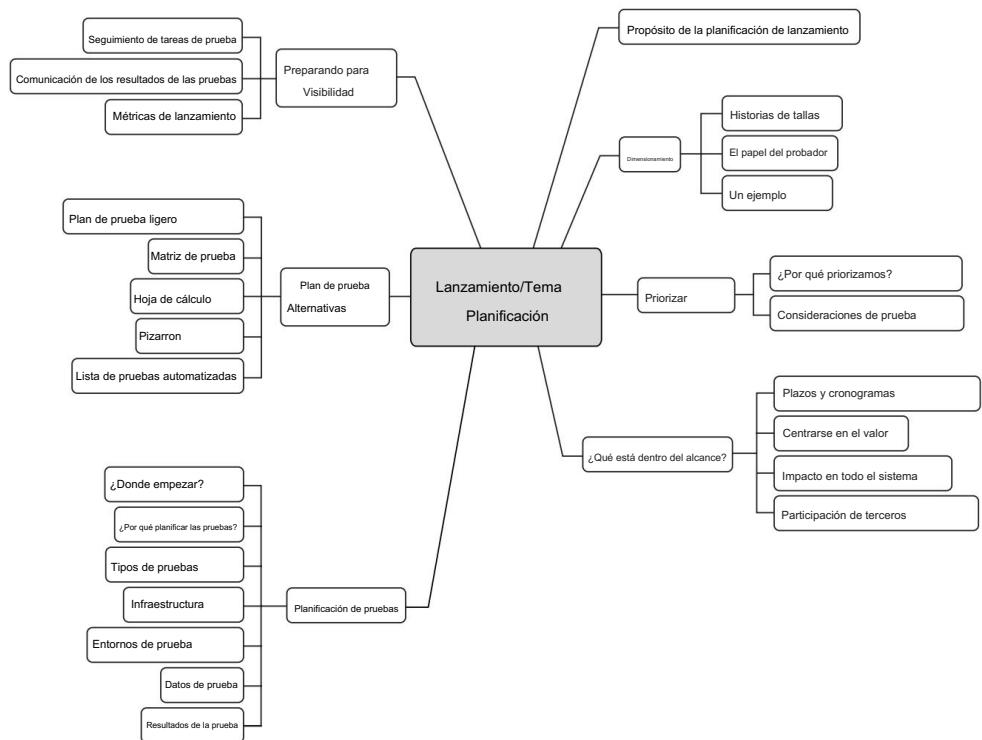
Siempre que realizamos tutoriales, seminarios web o sesiones de preguntas y respuestas con participantes que somos relativamente nuevos en el desarrollo ágil, siempre nos hacen preguntas como "¿Qué hacen los evaluadores durante la primera parte de una iteración antes de que algo suceda? ¿Listo para la prueba? o "¿Dónde encajan las pruebas de aceptación del usuario en una versión ágil?" "¿ciclo?" Es fácil exponer teorías sobre quién debería hacer qué y cuándo, en Es un proceso ágil, pero consideraremos que dar ejemplos concretos de nuestra propia experiencia es la mejor ayuda que podemos brindar a los novatos ágiles. A través de nuestra conversación con muchos diferentes equipos ágiles, hemos aprendido que hay muchos puntos en común en lo que Funciona bien para desarrollo y pruebas ágiles.

En esta parte del libro, seguiremos la vida de un evaluador ágil a lo largo de una iteración. En realidad, exploraremos más que una simple iteración. Empezaremos con lo que Los evaluadores hacen durante el lanzamiento o la planificación del tema, cuando el equipo analiza el trabajo. servirá para varias iteraciones futuras. Daremos ejemplos de lo que los probadores puede hacer para ayudar a los miembros del equipo a ponerse manos a la obra cuando inician el iteración. Mostraremos cómo la codificación y las pruebas son parte de un proceso integrado de entrega de software y describiremos cómo los evaluadores y programadores trabajar estrecha y progresivamente. Explicaremos diferentes formas en que los evaluadores pueden ayude a sus equipos a mantener el rumbo y medir el progreso, incluidos enfoques útiles para las métricas y el manejo de defectos. Analizaremos las actividades relacionadas con las pruebas involucradas en concluir una iteración y encontrar formas de mejorar para el el proximo. Finalmente, examinaremos el papel de un evaluador en una versión exitosa, incluido el juego final, UAT, empaquetado, documentación y capacitación.

Las actividades descritas en este análisis práctico de las pruebas ágiles pueden ser realizadas por cualquier miembro del equipo, no solo por los especialistas en pruebas. En algunos equipos, todos los miembros del equipo pueden realizar, y realizan, cualquier tarea, ya sea desarrollo, pruebas, base de datos, infraestructura u otras tareas. Para simplificar, en esta sección asumiremos que estamos siguiendo a alguien cuya función principal es realizar pruebas, ya que ayuda a entregar software de alta calidad.

Capítulo 15

ACTIVIDADES DEL PROBADOR EN LANZAMIENTO O PLANIFICACIÓN TEMÁTICA



Los equipos de desarrollo ágiles completan historias y entregan software listo para producción en cada iteración, pero planifican el panorama general o una mayor parte de la funcionalidad con anticipación. Un tema, epopeya o proyecto puede abarcar varias iteraciones. En este capítulo, analizamos lo que hacen los evaluadores cuando su equipo se toma el tiempo para planificar su lanzamiento. También consideraremos formas de rastrear si nuestro desarrollo avanza según lo previsto o si es necesario corregir el rumbo.

EL PROPÓSITO DE LA PLANIFICACIÓN DE LIBERACIÓN

Una de las razones por las que los equipos de software intentan el desarrollo ágil es porque saben que los planes a largo plazo no funcionan. La mayoría de los entornos empresariales son volátiles y las prioridades cambian cada semana o incluso cada día. Se supone que el desarrollo ágil Evite el "gran diseño desde el principio". La mayoría de nosotros hemos experimentado hacer planes que resultó ser una pérdida de esfuerzo. Pero tenemos que tener cierta comprensión de lo que nuestro cliente está buscando y cómo podemos entregárselo para conseguirlo. un buen comienzo. Afortunadamente, un enfoque ágil puede hacer que la planificación sea útil. manera de darnos una ventaja sobre cómo entregaremos el producto.

Planificación ágil aplicada

La hermana de Janet, Carol Vaage, enseña primer grado cuando no dirige conferencias. Ella relata su primera experiencia con el uso de prácticas ágiles para organizar una conferencia:

Mi mesa está repleta de carpetas y listas de tareas pendientes, y la sensación de estar abrumado me deja inactivo. Soy Director de la Conferencia y la tarea en este momento parece onerosa. Cuando mi hermana se ofrece a ayudarme, acepto, porque estoy desesperado por controlar esta planificación. Doy la bienvenida a Janet a mi desorden, le muestro mis páginas de listas escritas a mano de cosas que deben hacerse, le explico las enormes tareas que esperan mi atención y comparto cómo funciona mi comité.

Janet me mostró en un lenguaje sencillo cómo separar cada tarea en una nota adhesiva y cómo usar la coordinación de colores para diferentes responsabilidades y diferentes personas. Explicó sobre las columnas de "Tareas por hacer", "En Progreso", "Para revisar" y "Listo". Nunca antes había oído hablar de la palabra iteración, pero entendía perfectamente la línea de tiempo. Ella recomendó bloques de tiempo de dos semanas, pero yo elegí iteraciones de una semana. Instalamos una pared para mi tablero de planificación y Janet me dejó armarlo y agregar las tareas necesarias.

En los seis días transcurridos desde que Janet estuvo aquí, diez tareas se trasladaron de la columna Tareas pendientes a En progreso. Se realizan tres tareas y son específicas. Las tareas relacionadas con el tiempo han sido bloqueadas por el período de tiempo correcto. Lo más positivo es que a medida que agrego más tareas en la columna de tareas pendientes, no me siento abrumado. Entiendo que todo lo que necesito hacer es iniciar los pasos para iniciarla y luego el trabajo se vuelve más fácil. La sensación de caos ha desaparecido; Veo avances y entiendo que aún queda mucho trabajo por hacer. El cronograma es claro, las tareas son discretas y concretas. Y se ha abordado la tarea más difícil de todas: encontrar una manera de coordinar la videoconferencia para nuestro orador principal. ¡Este sistema funciona!

Las prácticas ágiles de planificación y seguimiento son útiles para algo más que el desarrollo de software. Un poco de tiempo cuidadosamente invertido y herramientas simples utilizadas para organizar y planificar las actividades de prueba y los recursos para una versión ayudarán al equipo a entregar software de alta calidad.

Los equipos de XP pueden tardar un día cada pocos meses en planificar el lanzamiento. Otros equipos ágiles realizan una planificación anticipada cuando se preparan para comenzar con un tema, una epopeya o una característica importante, que consideramos como un grupo relacionado de historias. Trabajan para comprender el tema o el lanzamiento a un alto nivel. ¿Cuál es la visión del cliente sobre lo que deberíamos ofrecer? ¿Cuál es el propósito del lanzamiento? ¿Cuál es el panorama general? ¿Qué valor aportará al negocio, a los clientes?

¿Qué otros equipos o proyectos están involucrados y requieren coordinación? ¿Cuándo se realizará la UAT? ¿Cuándo se lanzará el código para la puesta en escena y la producción?

¿Qué métricas necesitamos para saber si vamos por buen camino? Estas preguntas generales se abordan en la planificación del lanzamiento.

Algunos equipos no dedican mucho tiempo a realizar actividades de planificación de lanzamientos. Las prioridades cambian rápidamente, incluso dentro de un tema de características particular.

Nadie quiere hacer demasiado trabajo por adelantado que termine siendo desperdiciado. Algunos equipos simplemente miran las primeras historias para asegurarse de que pueden comenzar con buen pie. Como mínimo, los equipos quieren saber lo suficiente para orientar la arquitectura de su sistema en la dirección correcta y comenzar con las primeras historias.

Estas reuniones de planificación no tienen como objetivo planificar en detalle cada iteración del lanzamiento. Y sabemos que no podemos predecir exactamente cuántas historias podemos completar en cada iteración. Sin embargo, tenemos una idea de nuestra velocidad promedio, por lo que podemos tener una idea general del posible alcance del lanzamiento. El equipo habla sobre las características y las historias, tratando de obtener una vista de 20,000 pies de lo que puede incluirse en el lanzamiento y cuántas iteraciones podrían ser necesarias para completarse.

A ambos nos gusta el enfoque de Mike Cohn para la planificación de lanzamientos en su libro Agile Estimating and Planning [2005]. Las historias que la empresa quiere incluir se dimensionan entre sí y luego se priorizan las características según el valor que ofrecen. El equipo puede identificar "cortes finos" a través de las características para determinar qué historias es absolutamente necesario hacer, qué está dentro del alcance, qué "cosas interesantes" podrían posponerse para más adelante. Analizan las dependencias entre historias, el riesgo relativo y otros factores que determinan el orden en el que se deben codificar las características. El orden en que se codifican las historias es tan importante, o a veces más, que el tamaño de las historias. Los equipos quieren ofrecer valor en la primera versión del lanzamiento.

La planificación del lanzamiento es una oportunidad para que los desarrolladores y clientes consideren el impacto de las funciones planificadas en el sistema más grande, aclaren suposiciones y observen las dependencias que podrían afectar qué historias se hacen primero. Quizás piensen en realizar pruebas a un alto nivel y si se necesitarán nuevos recursos, como entornos de prueba y software.

Sigamos a nuestra evaluadora ágil a través de las actividades de planificación de lanzamientos y veamos cómo aporta valor a través de su perspectiva y enfoque únicos.

DIMENSIONAMIENTO

Los equipos ágiles estiman el tamaño relativo de cada historia. Algunos equipos dimensionan a medida que avanzan, lo que retrasa la estimación hasta la iteración en la que realmente completarán la historia. Otros se reúnen para estimar historias incluso antes de planificar su lanzamiento. Algunos equipos de desarrolladores y clientes se reúnen para escribir y estimar el tamaño de las historias al mismo tiempo. El objetivo del dimensionamiento es que los programadores le den a la empresa una idea del costo de cada historia y les ayuden a priorizar y planificar las primeras iteraciones. Los equipos de alto funcionamiento que han trabajado juntos durante años pueden adoptar un enfoque menos formal. Para los nuevos equipos ágiles, aprender a dimensionar las historias requiere mucha práctica y experiencia. No es importante que cada historia tenga el tamaño correcto, sino que esté lo suficientemente cerca como para darles a los clientes una idea de qué tan grandes son las historias para que puedan priorizar con mejor información. Con el tiempo, las variaciones en el tamaño de las historias individuales se promediarán y descubriremos que un tema o grupo relacionado de historias requiere aproximadamente la cantidad de tiempo esperada.

Cómo dimensionar las historias

En cuanto a cómo calcular el tamaño de la historia, diferentes equipos utilizan diferentes técnicas, pero nuevamente, nos gusta el enfoque de Mike Cohn para determinar el tamaño de la historia. Dimensionamos puntos de la historia, días ideales o simplemente "pequeño, mediano, grande". El tamaño relativo de cada historia con respecto a las demás es el factor importante. Por ejemplo, agregar un campo de entrada a una interfaz de usuario existente es obviamente mucho más pequeño que desarrollar una pantalla nueva desde cero.

Si la empresa conoce la velocidad promedio (la cantidad de puntos de la historia que el equipo completa en cada iteración) y tiene las estimaciones del tamaño inicial de cada historia que quiere completar, tiene una idea de cuánto tiempo podría llevar implementar un tema determinado. Como ocurre con cualquier otra metodología de desarrollo, no hay garantías, porque las estimaciones son sólo eso. Aún así, la empresa puede planificar lo suficientemente bien como para llevar a cabo sus actividades habituales.

Nuestros equipos utilizan el póquer de planificación (explicado en el libro Agile Estimating and Planning de Mike Cohn) para estimar el tamaño de la historia. Al planificar el póquer, cada miembro del equipo tiene una baraja de cartas. Cada carta tiene una cantidad de puntos. El proceso comienza cuando el cliente o propietario del producto lee una historia y explica su propósito y el valor que aportará. Podría enumerar algunas condiciones de satisfacción o casos de prueba de alto nivel. Después de una breve discusión, cada miembro del equipo levanta una tarjeta de puntos que representa cuán "grande" creen que es la historia desde su perspectiva. Discuten cualquier diferencia importante en el valor de los puntos y vuelven a estimar hasta llegar a un consenso. La figura 15-1 muestra a los miembros del equipo hablando sobre



Figura 15-1 Póquer de planificación

los valores de puntos que cada uno acaba de mostrar. Este debe ser un proceso rápido.

Las largas discusiones sobre los detalles no dan como resultado estimaciones de tamaño más precisas.

Algunos equipos calculan los tamaños relativos de las historias según la cantidad de personas que se necesitan completar una historia determinada en un tiempo determinado. Otros estiman cuántos

Días ideales necesitaría una persona para terminarlo. Utilice una medida que haga

sentido para todos los miembros del equipo y que proporcione coherencia entre las estimaciones.

El papel del evaluador en las historias de dimensionamiento

Uno de nuestros dichos favoritos es: "Ninguna historia está terminada hasta que se prueba". Sin embargo,

Nos hemos topado con equipos donde las pruebas no se incluyeron en las estimaciones del tamaño de la historia.

En algunos casos, probar una funcionalidad puede llevar más tiempo que codificarla.

Según nuestra experiencia, los evaluadores suelen tener un punto de vista diferente al de otros equipos.

miembros. A menudo tienen un amplio conocimiento del dominio y pueden

Identificar rápidamente los "efectos domino" que una historia podría tener en el resto de la historia.

sistema. También tienden a pensar en actividades no directamente relacionadas con el desarrollo que podrían

necesitar realizarse, como capacitar a los usuarios sobre un tema nuevo o modificado.

interfaz.

La historia de Lisa

¿Qué hace un evaluador durante el proceso de dimensionamiento de la historia? Pienso rápidamente en la historia desde varios puntos de vista. ¿Qué problema empresarial resuelve la historia o qué valor empresarial ofrece? Si esto no está claro, le hago preguntas al propietario del producto.

¿Cómo utilizará realmente el usuario final la función? Si aún no está claro, le pido al propietario del producto un ejemplo rápido. Podría preguntar: "¿Qué es lo peor que podría salir mal?" Este enfoque negativo ayuda a medir el riesgo de la historia. ¿Qué consideraciones de prueba podrían afectar el tamaño de la historia? Si será difícil obtener datos de prueba o la historia involucra a un tercero, las pruebas pueden llevar más tiempo que la codificación. Intento eliminar rápidamente cualquier suposición oculta. ¿Existen dependencias o riesgos especiales de seguridad? ¿Esta parte de la aplicación necesitará manejar una gran carga?

Muchas historias no son lo suficientemente grandes como para merecer tanta atención. Por lo general, no lo hacemos. Se necesitan muchos detalles para tener una idea del tamaño relativo. Sin embargo, su equipo puede sufrir mucho si una historia se subestima en un factor de cinco o diez. Una vez dimos una estimación relativamente pequeña de una historia que acabó teniendo al menos diez veces su tamaño.

Estos son los desastres que queremos evitar haciendo buenas preguntas.

—Lisa

Los evaluadores deben ser parte del proceso de dimensionamiento. Algunos equipos piensan que sólo los programadores deberían participar, pero cuando los evaluadores son participantes activos, puede ayudar a obtener un tamaño de historia mucho más preciso, lo cual redunda en el mejor interés de todo el equipo.

Un ejemplo de historias de tallas

Imaginemos que tenemos la historia de la Figura 15-2 para evaluarla.

Historia PA-3	
Como comprador en nuestro sitio, quiero eliminar artículos	
fuera de mi carrito de compras para no comprar	
artículos adicionales que decido que no quiero.	

Figura 15-2 Historia para eliminar elementos

Después de que el propietario del producto lee la historia, se produce la siguiente discusión:

Propietario del producto: "Solo queremos una manera fácil para que los usuarios eliminen elementos, pero no tenemos una implementación específica en mente".

Probador: "¿Deberían poder eliminar varios elementos a la vez?"

Propietario del producto: "Oh, sí, hazlo lo más fácil posible".

Probador: "¿Qué pasa si eliminan accidentalmente un artículo que querían comprar?"

Propietario del producto: "¿Hay alguna forma de guardar los elementos eliminados para recuperarlos más adelante?"

Programador: "Claro, pero deberías escribir una nueva historia para eso. Por ahora, deberíamos comenzar con la funcionalidad básica de eliminación".

Probador: "En la última versión implementamos una función de lista de deseos. ¿Quiere que los usuarios puedan mover artículos de su carrito de compras a su lista de deseos? Esa también sería una historia nueva".

Propietario del producto: "Sí, esas son dos historias más que queremos hacer, sin duda. Los anotaré, también podemos dimensionarlos. Pero definitivamente podríamos posponerlos hasta el próximo lanzamiento, si es necesario".

Probador: "¿Qué es lo peor que podría pasar con esta función?"

Propietario del producto: "Si no saben cómo eliminarlo, es posible que simplemente abandonen toda su cesta de la compra. Tiene que ser realmente fácil y obvio".

El ScrumMaster pide un presupuesto. El equipo entiende que están dimensionando solo la historia básica para eliminar elementos, no para hacer otra cosa con el Objeto eliminado. Rápidamente se ponen de acuerdo sobre un valor en puntos.

Veamos otra historia. (Ver Figura 15-3.)

Historia PA-4	
Como cliente, quiero saber cuánto mi	
El envío del pedido tendrá un coste según el envío.	
velocidad que elijo para poder elegir una diferente	
Velocidad de envío si quiero.	

Figura 15-3 Historia sobre la velocidad de envío

Probador: "¿Cuáles son las velocidades de envío que el usuario puede elegir?"

Propietario del producto: "Estándar de 5 días, 2 días y el día siguiente".

Programador: "Probablemente deberíamos comenzar ofreciendo solo una velocidad y calcular ese costo. Entonces podremos implementar fácilmente las otras dos velocidades".

Propietario del producto: "Está bien dividirlo así".

Probador: "¿Utilizaremos la API de BigExpressShipping para calcular el costo según el peso y el destino?"

Programador: "Eso sería lo más fácil".

El equipo levanta sus tarjetas de puntos. El tester y uno de los programadores levante un 8; los otros desarrolladores levantan un 5.

ScrumMaster: "¿Por qué eligieron ustedes dos 8?"

Probador: "Nunca antes habíamos utilizado la API de costos de BigExpressShipping y no estoy seguro de cómo afectará eso a nuestras pruebas. Tenemos que descubrir cómo acceder a su sistema para realizar pruebas".

Otro programador con 8: "Estoy de acuerdo, creo que el esfuerzo de prueba es más intenso que el esfuerzo de codificación para esta historia".

El equipo acuerda dimensionar la historia en ocho puntos.

Este proceso de dimensionamiento puede ocurrir antes de la reunión de planificación, y si las historias se dimensionaron o estimaron hace mucho tiempo, es posible que el equipo quiera hacer Asegúrese de que se sientan cómodos con el tamaño de las historias. Los equipos pueden haber cambiado o Puede que tenga más experiencia. Cualquiera de esos factores puede hacer que un equipo cambie las estimaciones.

Hay muchas ocasiones en las que una historia tendrá un gran componente de prueba y el esfuerzo de codificación es pequeño. En otras ocasiones sucederá lo contrario. Es importante considerar todas las perspectivas.

La historia de Lisa

Nuestro equipo empezó a temer las reuniones de dimensionamiento de historias, porque teníamos discusiones demasiado largas sobre los detalles, y las reuniones siempre duraban mucho más allá del tiempo programado. Desde entonces, nuestro ScrumMaster ha encontrado formas de mantenernos encaminados. Utiliza un cronómetro para cronometrar las discusiones y las detiene cada vez que se acaba el tiempo para ver si creemos que realmente necesitamos más tiempo para hacer preguntas. Nuestro propietario de producto también ha aprendido qué información necesitamos para realizar estimaciones y, por lo general, tiene la que necesitamos. También aprendimos a trabajar solo en historias que probablemente surgirían en las próximas iteraciones.

En todas nuestras reuniones, han crecido pequeñas tradiciones para hacerlas más divertidas. Siempre hay alguien que trae regalos a las reuniones de planificación de iteraciones. En las reuniones de pie, pasamos una combinación de linterna y puntero láser, de modo que cada uno de nosotros lo sostiene mientras informamos sobre en qué estamos trabajando. Siempre terminamos las reuniones de análisis de historias con una competencia para ver quién puede arrojar su mazo de cartas de póquer de planificación en el pequeño recipiente de plástico donde viven. La figura 15-4 muestra esta actividad ridícula pero divertida para finalizar una reunión. Recuerda siempre el valor ágil de disfrutar y divertirte con tus reuniones.

—Lisa



Figura 15-4 Una tradición de finalizar una reunión

PRIORIZAR

El propósito de la reunión de planificación del lanzamiento también es tener una idea de lo que Historias que el equipo intentará terminar antes de la fecha de lanzamiento. Los clientes priorizan las historias, pero puede haber dependencias, por lo que tiene sentido hacer ciertas las historias primero, incluso si no son la máxima prioridad. Es importante que el equipo comprende la posibilidad de que no todas las historias se completen para la fecha de lanzamiento. Una de las premisas básicas de Agile es entregar trabajo software, por lo que es importante completar primero las historias de mayor valor para que el software que entregamos satisfaga las necesidades del cliente.

Por qué priorizamos las historias

El objetivo de todos es ofrecer valor real en cada iteración. Los probadores pueden ayudar al equipo selecciona la funcionalidad principal que tiene que funcionar. En el Capítulo 8, explicamos el concepto de "corte fino" o "hilo de acero", identificando un camino a través la funcionalidad para codificar y probar primero, y agregar más funciones después de la primera El camino crítico funciona. Este concepto también se aplica a nivel de lanzamiento. El orden de Las historias son críticas. A veces, el equipo de Lisa divide una historia y la retira. una parte central de una característica que se debe realizar en la primera iteración.

Algunos equipos que no planifican completamente el lanzamiento se toman tiempo para analizarlo. las historias y decide cuáles dos o tres deberían ser las primeras. De esa manera, brindan valor comercial en la primera versión del lanzamiento.

Veamos un ejemplo.

Si nuestro tema es brindar la posibilidad de que un comprador en línea elija opciones de envío y luego calcular el costo de envío según el peso, velocidad de envío y destino, puede ser una buena idea completar historias simples o incluso subconjuntos de historias para que el proceso de pago pueda continuar de principio a fin. Comience permitiendo únicamente envíos estándar de 5 días, artículos menos de 10 libras y destinos en los Estados Unidos continentales. Cuando el usuario pueda obtener el costo de envío para ese escenario y realizar el pago, el equipo puede decidir las próximas prioridades. Pueden incluir peso pesado artículos, velocidades de envío más rápidas, envíos a Hawái y Alaska, y envíos a Canadá y México.

Al proporcionar esta fina porción primero, los evaluadores tienen algo con lo que comenzar a probar. inmediatamente. Los programadores también han probado su diseño y los pasos de integración del código y, por lo tanto, tienen una idea sólida de cómo funcionarán las cosas cuando todo La función está completa.

Consideraciones de prueba al priorizar

Es importante que el equipo comprenda el panorama o el tema general. En nuestro ejemplo, los miembros del equipo conocen las historias de envíos fuera del continente.

Estados Unidos vendrá después. Este conocimiento puede afectar la forma en que implementan la primera historia. Esto no significa que tengan que planificar para cada eventualidad, pero si saben que necesitan más opciones de envío, pueden implementar una lista desplegable en lugar de un campo de texto básico. No es necesario hacer más trabajo o reelaborar de lo necesario.

Durante la planificación del lanzamiento, también consideramos el riesgo relativo de las historias. Si ciertas historias tienen muchas incógnitas, sería mejor incluirlas en una iteración temprana, por lo que hay tiempo para recuperarse si una historia "explota" y requiere mucho más tiempo del estimado. Lo mismo puede aplicarse a una historia que, si no se completa o se implementa incorrectamente, tendría un impacto negativo costoso.

Programarlo con anticipación dejará más tiempo para realizar pruebas.

Si se necesita nueva tecnología o software, podría ser bueno aprenderlo desarrollando una historia sencilla y planificando otras más difíciles para iteraciones posteriores. Esta nueva tecnología puede afectar o no la automatización de sus pruebas. Tú

Es posible que desee más tiempo para comprobar el impacto. Si las características son todas de marca nuevo y el equipo necesita más tiempo para comprender cómo deben funcionar, planee hacer menos que su velocidad promedio para la primera iteración. De esa manera, tendrá más tiempo para escribir pruebas que guiarán correctamente el desarrollo. Identificar riesgos y decidir qué enfoque tiene más sentido a partir de una prueba perspectiva y también una perspectiva de desarrollo. Ésta es una de las razones por las que Es importante incluir a todo el equipo en las sesiones de planificación.

Es esencial mirar las historias desde un punto de vista de prueba. Aquí es donde los evaluadores agregan el mayor valor. El equipo necesita desarrollarse en unidades pequeñas y comprobables. fragmentos para ayudar a decidir qué historias se planean tentativamente para qué iteración. La clave aquí es comprobable. Muchos nuevos equipos ágiles piensan en pequeñas porciones significa hacer todo el trabajo de la base de datos primero, o todo el material de configuración. Comprobable tampoco significa necesariamente que necesite una GUI. Por ejemplo, el algoritmo que calcula el costo de envío es un fragmento de código independiente que Se puede probar independientemente de cualquier interfaz de usuario, pero requiere pruebas exhaustivas. Esa podría ser una buena historia para la primera iteración. Se puede probar como código independiente y luego probarlo en combinación con la interfaz de usuario y otros partes del sistema.

Los evaluadores pueden ejercer presión para conseguir que una bala trazadora de extremo a extremo atraviese el codificar rápidamente, para que puedan construir un marco de automatización y luego desarrollarlo

a medida que avanza el desarrollo de la historia. Si hay historias que presentan una gran desafío de prueba, podría ser bueno hacerlo desde el principio. Por ejemplo, si el lanzamiento incluye la implementación de una nueva herramienta de terceros para crear documentos. A partir de plantillas y datos dinámicos, hay muchas permutaciones para probar. Si el equipo no está familiarizado con la herramienta, los evaluadores pueden pedirle que considere hacer esas historias en la primera versión del lanzamiento.

¿QUÉ HAY ALCANCE ?

Los equipos ágiles gestionan continuamente el alcance para cumplir con los plazos comerciales preservando la calidad. Las historias de alto valor son la primera prioridad. Historias que son "buenos para tener" podrían ser eliminados del lanzamiento.

La historia de Lisa

Los clientes de nuestro equipo enumeran sus historias en orden de prioridad y luego trazan una línea entre las historias que deben realizarse antes de que se produzca el lanzamiento y las que podrían posponerse de forma segura. Llaman a las historias menos importantes "debajo de la línea". y es posible que esas historias nunca se hagan.

Por ejemplo, cuando abordamos el tema de permitir que los participantes del plan de jubilación tomaran dinero prestado de sus cuentas de jubilación, hubo un "debajo de la línea". story para enviar correos electrónicos a cualquier participante cuyos préstamos estén cambiando de estado a "incumplimiento pendiente" o "incumplimiento". Cuando el préstamo se encuentra en estado de "incumplimiento", el prestatario debe pagar impuestos y multas sobre el saldo. El correo electrónico sería de gran ayuda para los prestatarios, pero no era tan importante para nuestro negocio como el software para solicitar, aprobar y distribuir préstamos o procesar pagos de préstamos.

La historia del correo electrónico no apareció en el comunicado. No se hizo hasta más de dos años después, después de suficientes quejas de personas que no sabían que sus préstamos estaban en mora hasta que fue demasiado tarde.

—Lisa

Janet trabajó con un equipo cuyos clientes tenían la suposición equivocada de que todas las funciones se incluirían en su lanzamiento y que cuando estaban priorizando, simplemente eligiendo qué historias se hacían primero. Cuando el resto del equipo se dio cuenta del malentendido, también implementaron la idea de historias por encima y por debajo de la línea. Ayudó a seguir el progreso. además de hacer que las historias que se colocaron debajo de la línea sean muy visibles.

Plazos y cronogramas

Muchos dominios giran en torno a fechas fijas en el calendario. Negocios minoristas obtener la mayor parte de sus ganancias durante la temporada navideña. Un sitio de venta minorista en Internet es inteligente tener todas las funciones nuevas implementadas antes del 1 de octubre. Implementar un

La nueva característica cerca del período pico de compras es arriesgada. Los clientes de la empresa de Lisa deben completar tareas requeridas por el gobierno durante ciertos períodos de tiempo. el año. Cuando es demasiado tarde para lanzar una función este año, a menudo se vuelve posponer para el próximo año, porque es necesario abordar prioridades más urgentes. Los cambios regulatorios tienen cronogramas específicos y las organizaciones no tienen otra opción sobre la línea de tiempo.

La historia de Janet

Mientras trabajaba en este libro, estaba planeando su lanzamiento con mi equipo de WestJet. Teníamos varias historias posibles y trabajamos con los clientes para decidir cómo sería el lanzamiento. Tuvimos un cambio regulatorio que fue un trabajo liviano para los programadores, pero pesado para los evaluadores. Tenía que estar en producción en una fecha determinada, por lo que las otras historias que estábamos considerando para el lanzamiento lo tuvieron en cuenta.

Decidimos crear una pequeña versión de mantenimiento con solo esa característica principal, junto con algunos errores del trabajo pendiente para que no se pusiera en peligro la publicación del cambio regulatorio. Mientras los evaluadores completaban sus pruebas, el resto del equipo comenzó algunas historias de elaboración para la próxima versión.

Un plan alternativo podría haber sido que los programadores colaboraran y ayudaran a probar e incorporar más funciones. Sin embargo, todo el equipo decidió que este plan funcionaría mejor con el menor riesgo.

—Janet

Centrarse en el valor

Es bastante fácil para un equipo empezar a discutir una historia compleja y perder de vista qué valor ofrecen realmente las funciones. La planificación del lanzamiento es el momento de empezar pidiendo ejemplos y casos de uso de cómo se utilizarán las funciones y qué valor que proporcionarán. Dibujar diagramas de flujo o cálculos de muestra en la pizarra puede ayudar a identificar la funcionalidad principal.

La historia de Lisa

El propietario de nuestro producto escribió una historia para brindar una advertencia si un empleador anula la fecha en que un participante se vuelve elegible para contribuir a una cuenta de jubilación después de que el participante ya haya realizado contribuciones.

La advertencia debía incorporarse al código de interfaz de usuario heredado, que no acomodarlo fácilmente. El equipo discutió cómo podría implementarse, pero cada opción era bastante costosa. La codificación no sólo sería complicada, sino que se necesitaba mucho tiempo para probarla adecuadamente y actualizar las pruebas automatizadas existentes. Esta característica no proporcionaría mucho valor al negocio, sólo un poco de ayuda a los usuarios finales. El lanzamiento ya estaba bastante cerca del límite de funciones.

Uno de los programadores sugirió proporcionar un informe de los participantes que cumplían con los criterios para que los administradores del plan pudieran simplemente llamar a los empleadores que pudieran

necesidad de hacer correcciones. La historia del informe era mucho más pequeña que la historia de advertencia, podía encajar fácilmente en el comunicado y era aceptable para el cliente.

—Lisa

No hay garantía de que estas “estimaciones” iniciales sobre lo que será en un la liberación dada se mantendrá con el tiempo. Es por eso que los clientes deben comprender sus prioridades, realizar puntos de control al final de cada iteración y reevaluar las prioridades de las historias restantes.

Impacto en todo el sistema

 Hablamos de los “efectos domino” en el Capítulo 8, “Pruebas de cara al negocio que respaldan al equipo”.

Uno de nuestros trabajos como evaluadores es tener presente el panorama general. El probador ágil Piensa en cómo cada historia podría afectar al sistema en su conjunto, o a otros sistemas con los que el nuestro tiene que trabajar. Por ejemplo, si el almacén de juguetes hace un cambie su software de inventario y el nuevo código tiene un error que exagera la cantidad de artículos en stock, el sitio web podría vender más de la nueva muñeca de los disponibles, decepcionando a miles de niños y a sus padres en Navidad. Cuando el riesgo es alto, enumerar las áreas del sistema que podrían estar afectados por un tema o grupo de historias puede ser un ejercicio que valga la pena incluso durante la planificación del lanzamiento.

Puntos de contacto entre nuestro sistema y el de socios o proveedores siempre merecer consideración. Incluso un cambio menor en el formato de un archivo csv o xml podría tendrá un gran impacto si no lo comunicamos correctamente a los socios que utilizan ftp archivos a nosotros. Las historias que significan que los cambios para terceros deben realizarse con anticipación suficiente en el ciclo de lanzamiento para permitir que terceros realicen los cambios necesarios.

La Figura 15-5 muestra un diagrama simplificado de un nuevo sistema que afecta a muchos piezas del sistema existente. Es posible que se necesiten diferentes herramientas para probar la integraciones.

Probadores que hayan trabajado con algunos de los otros sistemas o entiendan qué Las pruebas necesarias en esos sistemas pueden ofrecer información valiosa sobre el Impacto de una nueva historia. A menudo, será necesario retrasar las historias hasta un lanzamiento futuro si no se ha explorado el impacto. Este es un buen momento para recordar lanzamientos anteriores que no terminaron tan bien.

Participación de terceros

Trabajar con herramientas de proveedores, socios u otros equipos de contratistas en un gran El proyecto complica la planificación del lanzamiento. Si alguien fuera de su equipo es responsable de alguna parte del proyecto, esa es una parte que está fuera de su control. Si

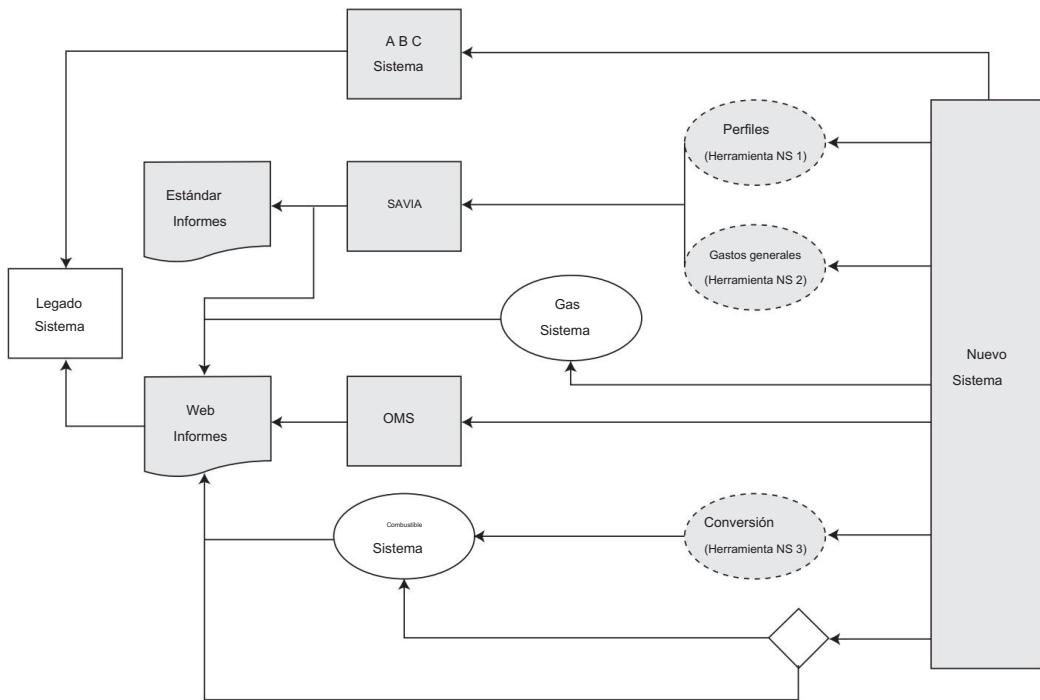


Figura 15-5 Impactos del sistema

Si necesita coordinarse con otros, incluidos posibles nuevos usuarios del sistema, es mejor comenzar temprano.

El equipo de Lisa ha escrito varias interfaces para permitir a los usuarios cargar datos en sus sistemas. En cada caso, tenían que hacer llegar el formato de archivo propuesto a los usuarios con antelación para asegurarse de que funcionaría para ellos. Otros proyectos implicaron el envío de datos a socios o proveedores. Estos requirieron una planificación adicional para organizar las pruebas con sus sistemas de prueba y obtener comentarios sobre si los datos eran válidos y estaban formateados correctamente.

Si está utilizando un producto de terceros como parte de su solución, puede suponer que ha sido probado, pero esa podría ser una suposición errónea. Necesitará dedicar tiempo adicional para probar su aplicación junto con el software del proveedor. Si hay un problema en el software de la otra empresa, su resolución puede tardar mucho tiempo. El equipo de Lisa utiliza software de terceros para tareas críticas como la creación de documentos. Si un tema incluye modificar o crear nuevos documentos, planean tiempo adicional para actualizar el software si es necesario, y más

tiempo para realizar pruebas en caso de que se necesiten correcciones. Si es posible, traiga al tercero el software en el proyecto con antelación y comenzar las pruebas de un extremo a otro. Cuanto más pueda trabajar con la interfaz, mejor estará.

Otro software de terceros que a menudo olvidamos hasta que es demasiado tarde es nuestro propios entornos de prueba. A veces un equipo incorporará código nuevo que aprovecha las nuevas funciones en el idioma elegido. Por ejemplo, si Los miembros del equipo están usando AJAX o JavaScript, es posible que necesiten actualizar el kit de desarrollo de software que están usando. Esto significa que un equipo tendrá que actualice también su entorno de ejecución de producción, así que téngalo en cuenta y pruébelo con antelación.

Los clientes o socios pueden tener inquietudes sobre una versión que no está dentro de su alcance del propio equipo. El equipo de Lisa se vio impedido en el último momento. lanzar una nueva función porque un socio no tuvo tiempo de aprobar la cambiar con sus asesores legales. Los programadores tuvieron que idear rápidamente una manera de desactivar la funcionalidad sin requerir pruebas adicionales extensas. Curiosamente, los socios que no utilizan el desarrollo ágil a veces tienen problemas para cumplir con sus propios plazos. Es posible que no estén preparados cuando su equipo cumple con el plazo.

La historia de Janet

Trabajé en un proyecto para implementar una función que requería una nueva pieza de hardware para escanear un nuevo código de barras 2D. El equipo decidió implementarlo por etapas porque no se sabía cuándo estarían disponibles los escáneres para realizar pruebas completas, pero el cliente quería que el código estuviera listo para cuando llegaran los escáneres.

La fase inicial requirió mucha programación porque había mucha investigación por hacer. Después de determinar cómo implementarían la función, se creó la historia para agregarla al código. Sin embargo, sabíamos que no podríamos probarlo a fondo hasta que los escáneres estuvieran disponibles. El código estaba listo para probarse, pero en lugar de revertirlo todo, solo teníamos que preocuparnos de probar que la función podía desactivarse para el lanzamiento. La próxima versión requeriría más pruebas, pero sólo si los escáneres estuvieran disponibles. Las pruebas de la historia se mantuvieron en el trabajo pendiente del producto para que no nos olvidáramos de hacerlo.

—Janet

Si va a trabajar con otros equipos desarrollando diferentes componentes del mismo sistema, o sistemas relacionados, presupuestar tiempo para coordinar con ellos. Es una buena idea designar a un miembro de cada equipo para coordinar juntos.

La planificación del lanzamiento es el momento de identificar roles adicionales que necesita en su equipo, recursos adicionales y tiempo necesario para circunstancias fuera de lo común.

PLANIFICACIÓN DE PRUEBAS



Capítulo 8,
"Presentación empresarial
Pruebas que apoyan al
equipo", explica
cómo identificar
hilos de acero o
cortes finos en una
historia o tema.

No podemos esperar planificar las iteraciones en una versión a un nivel detallado. Podemos tener una idea de los hilos de acero del tema, priorizar historias y adivinar qué historias estarán en qué iteración. La planificación detallada de las pruebas debe esperar para la planificación de iteraciones. Aún así, debemos pensar en realizar pruebas a un alto nivel, y trate de presupuestar suficiente tiempo para ello. Incluso podríamos tomarnos un tiempo por separado de la reunión de planificación del lanzamiento para diseñar estrategias para nuestras pruebas para el lanzamiento. En Capítulo 8, Pruebas de cara al negocio que respaldan al equipo, mencionamos una de los peligros de las pruebas ágiles: "olvidar el panorama general". La planificación de pruebas ayudarte con ese problema.

Donde empezar

Durante la planificación del lanzamiento, es útil conocer las condiciones comerciales de satisfacción para cada historia o caso de prueba de aceptación del usuario de alto nivel. cuando las historias Necesita aclaración, los evaluadores ágiles piden ejemplos. En esta etapa, se presentarán ejemplos. Ser de alto nivel y cubrir solo lo básico, pero lo suficiente como para poder dimensionar y priorizar la historia. Dibujar diagramas de flujo o escribir cálculos en la pizarra y discutirlos nos ayuda a identificar problemas de prueba específicos del proyecto.

Como mínimo, el equipo necesita comprender las historias de máxima prioridad que están programado para realizarse primero. La planificación ligera podría implicar sólo mirando esas historias centrales con el entendimiento de que se necesitará más tiempo necesario para definir pruebas adicionales.

A medida que tengamos una idea de qué historias probablemente se incluirán en el lanzamiento, Podemos empezar a pensar en el alcance de las pruebas. ¿Qué suposiciones tienen? ¿Se ha hecho algo que pueda afectar las pruebas? El uso de software de terceros, como el ejemplo de uso de la API de cálculo de envío de una empresa de envío, afecta la prueba planificación. ¿Existe algún riesgo inusual en esta versión que afecte las pruebas? Si tenemos historias para implementar trabajos por lotes y nunca hemos tenido ningún trabajo por lotes procesamiento en el sistema antes, probablemente haya nuevos marcos que afecten las pruebas. Necesitamos dedicar tiempo a aprenderlos.

¿Por qué escribir un plan de prueba?

En la planificación del lanzamiento, hablamos sobre el propósito del lanzamiento, su alcance, y qué suposiciones estamos haciendo. Hacemos un análisis rápido de riesgos y planificar nuestro enfoque de prueba para abordar esos riesgos. Consideramos la automatización y lo que necesitamos para entornos de prueba y datos de prueba. Ciertamente queremos identificar hitos y resultados. Hmmm, esto está empezando a sonar como... . a

¡Plan de prueba!

Consulte el Capítulo 5,
“Transición
Procesos típicos”,
para más sobre
Planes de prueba y
estrategias de prueba.

Si, como nosotros, pasó tiempo trabajando en un entorno de cascada tradicional, es posible que haya perdido el tiempo escribiendo voluminosos planes de prueba que nadie leyó. y nadie se molestó en mantener. En desarrollo ágil, queremos que nuestra prueba planes para satisfacer nuestras necesidades. Es posible que su cliente necesite un plan de prueba para cada versión por motivos de cumplimiento. Incluso si no es un entregable requerido, puede serlo. útil. Mantenlo conciso y ligero. Sólo tiene que servir a tus propósitos. durante este lanzamiento. Abordar los problemas de prueba que son específicos de esta versión. o proyecto. Incluya su análisis de riesgos e identifique suposiciones. Delinear el factores críticos de éxito que su cliente ha identificado. Pensar en qué la gente necesita saber lo relacionado con las pruebas y eliminar todo lo superfluo.

Incluso si no crea un plan de prueba formal, asegúrese de haber tomado nota de todos estos diferentes factores de prueba involucrados en la liberación. Querrás conservarlos en mente durante cada sesión de planificación de iteraciones. El mayor beneficio en la prueba. La planificación es la planificación misma. Le permite considerar y abordar problemas como los requisitos de datos de prueba, la infraestructura o incluso qué resultados de prueba se requieren. La planificación de pruebas es una estrategia de mitigación de riesgos. Consideraremos algunos de estos asuntos.

Tipos de pruebas

En la Parte III, cubrimos los cuatro cuadrantes de las pruebas y hablamos de todos los diferentes tipos de pruebas que puedes realizar durante tu proyecto. La planificación del lanzamiento es un buen momento para considerar estas diferentes necesidades. ¿Necesita planificar ¿Traerá una herramienta de prueba de carga o será necesario construir algún tipo de arnés de prueba especial?

Podría ser que su próximo lanzamiento sea sólo una extensión del último, y usted Simplemente continuará creando sus ejemplos, automatizando las pruebas de su historia y haciendo el resto de las pruebas como lo ha estado haciendo. Eres uno de los afortunados. unos. Para aquellos que estás empezando un nuevo proyecto sin previo procesos implementados, ahora es el momento de considerar qué pruebas necesitará. Nosotros No quiero decir que tengas que decidir cómo probar cada historia, sino mirar el panorama general y pensar en los cuadrantes. ¿Necesitará planificar una UAT especial? ¿O serán suficientes las demostraciones de iteración? Es importante plantear estas cuestiones temprano para que el equipo pueda planificarlos.

Infraestructura

Mientras planifica sus pruebas, debe considerar su infraestructura. La infraestructura puede significar su configuración y prueba de integración continua entornos y base de datos de prueba. Puede significar cómo promocionas tus construcciones.

a sus entornos de prueba. Podría significar su laboratorio de pruebas, si tiene uno, o tener un servidor separado para ejecutar todas sus pruebas de automatización. Estos son generalmente piezas de infraestructura que necesitan algo de tiempo para instalarse. Este es el Es hora de hacer un plan.

La historia de Lisa

Algunos tipos de pruebas pueden requerir un esfuerzo adicional. Mi equipo tenía una herramienta para realizar pruebas de rendimiento y algunos scripts, pero carecíamos de un entorno de estilo de producción donde pudieramos controlar todas las variables que podrían afectar el rendimiento. Por ejemplo, la base de datos de prueba fue compartida por evaluadores, programadores y dos procesos de compilación. Un rendimiento más lento podría simplemente significar que alguien estaba ejecutando pruebas intensivas en la base de datos. Usamos nuestro entorno de preparación para obtener una línea de base, pero le faltaban algunos de los componentes de producción. Nos fijamos un objetivo de seis meses para adquirir hardware y software para un entorno de prueba adecuado y configurarlo. Escribimos una o dos tarjetas de tareas en cada iteración para establecer el entorno paso a paso.

—Lisa

Cualesquiera que sean sus necesidades, asegúrese de comprenderlas y poder planificarlas. Que necesitas. Si no tienes la infraestructura adecuada, desperdiciarás tiempo tratando de ensamblarlo y causar un cuello de botella en mitad de la iteración.

Entornos de prueba

Al observar los tipos de funciones de la próxima versión, es posible que veamos la necesidad para un entorno de prueba completamente nuevo. Piense en entornos de prueba especializados es posible que también lo necesites. ¿Necesitará más herramientas? ¿Necesitas ampliar tu laboratorio de pruebas para que puedas realizar pruebas con diferentes navegadores y sistemas operativos? Este es el momento de pensar en todas las consideraciones de prueba.

Si está planificando su primera versión, los entornos de prueba son una consideración clave. Es posible que necesite una historia o iteración solo para configurar la infraestructura que necesita. necesidad. Hemos iniciado más de un proyecto donde el único lugar donde podíamos probar era el entorno de desarrollo. Descubrimos que eso no funciona muy bien, porque el entorno nunca es lo suficientemente estable para realizar pruebas efectivas.

Así como los programadores tienen sus propios entornos sandbox para trabajar y probar, funciona bueno si cada tester tiene esa misma disponibilidad y control. Reconocemos que No todas las aplicaciones se prestan a esto, pero al menos es necesario Sepa qué compilación está probando. También necesita datos de prueba que otros no obtendrán. caminar con sus pruebas. Si no tiene una zona de pruebas de prueba que esté debajo su propio control, tómese el tiempo para planificar lo que necesita establecer para su prueba ambientes. Piensa con tu equipo sobre cómo puedes obtener el

hardware y software necesarios. Puede llevar tiempo, así que desarrolle un plan B para hacer algo mientras espera la infraestructura que necesita.

Si está trabajando en un sistema grande, es posible que tenga que hacer cola junto con otros equipos para obtener tiempo en un entorno de prueba o preparación que incluya todas las distintas piezas de software con las que el suyo debe funcionar. Este entorno de preparación debe imitar su sistema de producción tanto como sea posible. Si

Su organización no tiene a alguien responsable de crear entornos, es posible que su equipo necesite funciones adicionales dedicadas a obtener los entornos de prueba que necesita. Estos roles pueden implicar trabajar con otros equipos como

Bueno. La planificación del lanzamiento es el momento de considerar todos estos requisitos de infraestructura de prueba.

Datos de prueba

La planificación del lanzamiento o del tema también es un buen momento para pensar en qué datos de prueba que puedas necesitar durante el proyecto.

Generalmente es una buena práctica utilizar datos de prueba que se asemejen mucho a los datos reales.

Planifique los datos que necesita. Hemos tenido la oportunidad en varias organizaciones de utilizar una copia de los datos de producción. Los datos reales proporcionan una buena base para

Diferentes escenarios para pruebas exploratorias. Es posible que los datos de producción deban ser "se limpia" antes de utilizarlo para realizar pruebas a fin de eliminar cualquier información confidencial, como números de identificación o de cuentas bancarias. Los datos necesitan debe modificarse para ocultar los valores originales pero sigue siendo válido para que no violar las restricciones de la base de datos. Porque a los expertos en bases de datos les lleva tiempo transferir datos de producción a un entorno de prueba, asegúrese de que estén incluidos en su planificación.

La historia de Janet

En una de las organizaciones con las que estaba trabajando, utilizamos dos esquemas de datos de prueba de referencia diferentes. Para nuestros entornos de prueba individuales, utilizamos dispositivos Fit para cargar datos predefinidos. Intentamos acercar estos datos lo más posible a la producción, pero también los sembramos con algunos datos de prueba muy específicos. Cada vez que revisábamos una nueva versión del código, podíamos recargar un conjunto base de datos. De esta manera, también probamos el esquema de la base de datos para ver si algo había cambiado.

Para nuestro entorno de prueba más estable donde queríamos que los datos persistieran, utilizamos los scripts de migración de datos que los programadores desarrollaron a medida que realizaban cambios en la base de datos. Estos scripts de migración finalmente se utilizaron para el corte inicial de producción y para entonces estábamos bastante seguros de que eran correctos.

—Janet

Consiga el apoyo de sus clientes para obtener datos de prueba significativos. Si eres Al trabajar en una historia que implica enviar un archivo a un proveedor externo, su El experto en negocios puede averiguar qué datos espera el proveedor en el archivo. la de lisa El equipo desarrolló funciones que permiten a los corredores de planes de jubilación ofrecer a sus clientes carteras de fondos mutuos. Le pidieron al propietario del producto que proporcionara muestras de carteras, incluido el nombre, la descripción y el conjunto de fondos para cada. Esto les ayudó a realizar pruebas con datos realistas.

Los datos de las pruebas tienden a quedar obsoletos y desactualizados con el tiempo. Datos más antiguos, incluso si proviene de la producción, es posible que ya no refleje con precisión la producción actual datos. Una prueba de "aprobación" que utiliza datos que ya no son válidos da una sensación engañosa de confianza. Revise continuamente sus necesidades de datos de prueba. Actualizar datos o crear utilizar un nuevo enfoque, según sea necesario.

Capítulo 14, "Un Agile Test Automation Strategy", explora diferentes enfoques para obtener datos de prueba.

Los requisitos de datos de prueba varían según el tipo de prueba. Pruebas de regresión Por lo general, pueden crear sus propios datos o ejecutarlos contra un pequeño conjunto representacional. de datos que se pueden actualizar rápidamente a un estado conocido. Prueba exploratoria Es posible que necesite una réplica completa de los datos del tipo de producción.

Resultados de la prueba

Diferentes equipos tienen diferentes requisitos para informar los resultados de las pruebas. Pensar sobre cómo vas a informar los resultados de las pruebas en esta etapa del juego para que puede hacerlo de manera efectiva cuando llegue el momento de realizar los informes reales. Su organización puede tener requisitos de cumplimiento de auditoría, o tal vez su El cliente solo quiere saber cómo lo probó. Entienda sus necesidades para que puede elegir el enfoque adecuado para su equipo.

Hay muchas maneras de informar los resultados de las pruebas. Hay herramientas de proveedores que registrar resultados tanto automáticos como manuales. Su equipo puede encontrar una manera de conservar los resultados de herramientas como Fit, o puede simplemente optar por mantener una gran participación. gráfico manual visible.

El enfoque que han adoptado algunos equipos es crear resultados de pruebas locales. aplicaciones. Por ejemplo, una aplicación Ruby simple escrita con Ruby en Los rieles para la base de datos o una base de datos MySQL con una interfaz PHP pueden hacer una Sistema de gestión de pruebas muy simple pero fácil de usar.

Una herramienta como esta puede ser muy simple o puede incluir complejidad adicional como como la capacidad de categorizar sus pruebas. Lo importante son los resultados de las pruebas. Si sus pruebas automatizadas registran su resultado de aprobación o falla junto con el error, tiene algún historial para ayudar a determinar la fragilidad de la prueba.

Su equipo puede configurar su proceso de compilación automatizado para proporcionar resultados de pruebas. de cada compilación, por correo electrónico, o una utilidad de comentarios o interfaz web que el equipo los miembros pueden verlo en línea. Los resultados a lo largo del tiempo se pueden resumir en una variedad de formatos que hagan visible el progreso. Uno de los equipos de Lisa produjo un gráfico diario. de pruebas escritas, ejecutadas y aprobadas que se publicó en el área de trabajo del equipo. Otro produjo un calendario diario con el número de pruebas unitarias que pasaban cada día. Incluso los resultados visuales simples son efectivos.

Hablamos de algunas de las métricas que puede utilizar más adelante en este capítulo.

ALTERNATIVAS DEL PLAN DE PRUEBAS

Hemos hablado sobre por qué probar el plan y qué debe considerar. Ahora nosotros Hable sobre algunas de las alternativas a los planes de prueba pesados a los que puede estar acostumbrado. Cualquiera que sea el tipo de plan de prueba que utilice su organización, hágalo suyo. Úselo en un manera que beneficie a su equipo y asegúrese de satisfacer las necesidades de sus clientes. Como ocurre con cualquier documento que produzca su equipo, debe cumplir un propósito.

Planes de prueba ligeros

Si su organización o cliente insiste en un plan de prueba para el cumplimiento de SOX o Otras necesidades regulatorias, considere un plan de prueba liviano que cubra las necesidades pero no los extras. No repetir elementos que ya han sido incluidos en el Plan del Proyecto o en la Carta del Proyecto. Un plan de prueba de muestra podría parecerse al que se muestra en la Figura 15-6.

Un plan de prueba no debe cubrir todas las eventualidades ni todas las historias, y no es destinado a abordar la trazabilidad. Debería ser una herramienta que le ayude a pensar en los riesgos de prueba para su proyecto. No debería sustituir la conversación cara a cara con su cliente o el resto de su equipo.

Usando una matriz de prueba

Janet utiliza la planificación de lanzamientos para trabajar con los evaluadores y los clientes para desarrollar una matriz de prueba de alto nivel. Una matriz de prueba es una forma sencilla de comunicar los grandes ítems sobre qué funcionalidad desea probar. Le da a tu equipo una descripción general rápida de las pruebas requeridas.

Una matriz de prueba es solo una lista de funcionalidades al costado y condiciones de prueba. a través de la cima. Al pensar en las condiciones y la funcionalidad de la prueba, considere toda la aplicación y cualquier impacto en la funcionalidad nueva o modificada.

Plan de pruebas del proyecto ABC Preparado por: Janet Gregory y Lisa Crispin							
Introducción El Plan de Pruebas pretende ser una base para identificar lo que se considera dentro y fuera del alcance de las pruebas, y cuáles son los riesgos y suposiciones.							
Recursos <table><thead><tr><th colspan="2">Probador % comprometido</th></tr></thead><tbody><tr><td>janet</td><td>100%</td></tr><tr><td>Lisa</td><td>50%</td></tr></tbody></table>		Probador % comprometido		janet	100%	Lisa	50%
Probador % comprometido							
janet	100%						
Lisa	50%						
En alcance Las pruebas incluyen todas las funciones nuevas, funcionalidad del conjunto de regresión de alto riesgo identificada, UAT y pruebas de carga. La localización es parte de este proyecto. Si el tiempo lo permite, se ejecutarán pruebas de regresión manual consideradas de baja prioridad.							
Fuera del alcance Las pruebas de traducción reales se subcontratan, por lo que no forman parte de este plan de pruebas.							
Nueva funcionalidad La siguiente funcionalidad se cambiará en esta versión.							
<table><thead><tr><th>Descripción de la característica</th><th>Profundidad de las pruebas</th></tr></thead><tbody><tr><td>Agregar una nueva palanca para la selección de idioma en la página de inicio</td><td>Prueba de los 5 idiomas (inglés, español, francés, italiano y alemán). Probando que somos capaces de cambiar de idioma dinámicamente.</td></tr></tbody></table>		Descripción de la característica	Profundidad de las pruebas	Agregar una nueva palanca para la selección de idioma en la página de inicio	Prueba de los 5 idiomas (inglés, español, francés, italiano y alemán). Probando que somos capaces de cambiar de idioma dinámicamente.		
Descripción de la característica	Profundidad de las pruebas						
Agregar una nueva palanca para la selección de idioma en la página de inicio	Prueba de los 5 idiomas (inglés, español, francés, italiano y alemán). Probando que somos capaces de cambiar de idioma dinámicamente.						
Pruebas de rendimiento y carga Las pruebas de carga se concentrarán en las siguientes áreas. Los detalles de las pruebas de carga se encontrarán en el documento Plan de pruebas de carga [enlace al Plan de pruebas de carga].							
UAT (Prueba de aceptación del usuario) La UAT se realizará y coordinará con la oficina de París y con la oficina de Calgary. Los usuarios serán elegidos por su experiencia en áreas y transacciones seleccionadas, así como por su dominio de uno de los siguientes idiomas: alemán, italiano, español o francés.							
Consideraciones de infraestructura El laboratorio de pruebas necesitará los 5 idiomas instalados y disponibles para realizar pruebas.							
Suposiciones La traducción ha sido probada antes de ser entregada al equipo del proyecto.							
Riesgos Se han identificado los siguientes riesgos y se han identificado las acciones apropiadas para mitigar su impacto en el proyecto. El impacto (o gravedad) del riesgo se basa en cómo se vería afectado el proyecto si se desencadenara el riesgo.							
<table><thead><tr><th># Riesgo</th><th>Plan de mitigación de impactos</th></tr></thead><tbody><tr><td>1</td><td>Los usuarios no están preparados para UAT High</td></tr></tbody></table>		# Riesgo	Plan de mitigación de impactos	1	Los usuarios no están preparados para UAT High		
# Riesgo	Plan de mitigación de impactos						
1	Los usuarios no están preparados para UAT High						

Figura 15-6 Ejemplo de plan de prueba

podría tener en el resto de la aplicación. Probadores sentados con clientes y Lo importante es pensar en las condiciones de prueba.

También puede ser un mecanismo para realizar un seguimiento de la cobertura y puede ser tan detallado como desee. como. El equipo puede utilizar una matriz de prueba de alto nivel para mostrarle al cliente equipo o dirección lo que ya se ha probado y lo que queda. un mas

El equipo puede utilizar una matriz de prueba detallada para mostrar lo que se planea para las pruebas y realizar un seguimiento del progreso de las mismas. Una vez creada la matriz, Resulta fácil completar los cuadrados cuando se realizan las pruebas. Mantenlo simple. Como nos gustan los gráficos grandes, visibles y fáciles de leer, recomendamos colores Eso significa algo para tu equipo. Por ejemplo, verde (G) significa que la prueba está hecho y el equipo está contento con ello, mientras que el amarillo (Y) podría significar que se han realizado algunas pruebas, pero se necesitan más pruebas exploratorias si hay tiempo. Rojo (R) significa que algo está roto. Un cuadrado blanco significa que no ha sido probado. todavía, y un cuadrado gris (no aplicable) significa que no es necesario probarlo.

Veamos un ejemplo. Tenemos un pequeño comunicado que queremos publicar que calcula los costos de envío. En la Figura 15-7, se muestran diferentes piezas de funcionalidad representado en un eje, y las propiedades del envío se representan en el otro. Las celdas individuales están codificadas por colores para mostrar qué casos se analizan y que necesitan más atención. Todas las celdas para " ≤ 2 lbs" están terminadas, la parte superior Se realizan tres celdas para > 4 libras, pero se necesitan más pruebas exploratorias, y el La celda "Envío a Alaska"/">4 libras" indica un posible problema.

Funcionalidad	Destino	Detalles	Dirección	Condiciones de la prueba				Estimaciones
				libras	Notas	Mismo	Difícil	
Enviar dentro de EE. UU.		GG			Y			
Enviar a Canadá			—		Y			
Enviar a Hawái			—		Y			
Enviar a Alaska			—		R			
Estimaciones de envío			—					n / A

Figura 15-7 Una matriz de prueba de muestra

La historia de Janet

Tuve un efecto secundario inesperado al usar una matriz de prueba en un proyecto en el que estaba. Los clientes y evaluadores armaron la matriz de prueba y pensaron en todas las funciones afectadas para el proyecto y las condiciones de prueba de alto nivel que necesitarían. Como era de esperar, el acto de planificación sacó a la luz muchos problemas que se habrían pasado por alto hasta más tarde.

Cuando colgaron la matriz en la pared del área de su equipo, Dave, el líder del equipo de desarrolladores, expresó interés. Uno de los evaluadores le explicó la matriz y me sorprendió cuando dijo que también era muy útil para ellos. Dave dijo: "No sabía que esta funcionalidad afectaría esta área. Necesitamos asegurarnos de que nuestras pruebas unitarias abarquen esto también".

Mirando hacia atrás, no debería haberme sorprendido, pero nunca antes había tenido esa experiencia con los programadores.

—Janet

Una matriz de prueba es una herramienta muy poderosa y puede usarse para ayudar a abordar problemas de trazabilidad si su equipo tiene esos problemas. Piensa en lo que tiene sentido para su equipo y adáptelo para su equipo y lo que tenga sentido para usted.

Hoja de cálculo de prueba

Janet también ha visto cómo se utiliza con cierto éxito un formato de hoja de cálculo. Por ejemplo, en WestJet, la primera pestaña de un libro de trabajo era una lista de alto nivel de funcionalidades que existían en la aplicación. Para cada fila, el equipo determinó si el El proyecto afectó esa pieza de funcionalidad. De ser así, dieron una calificación del impacto esperado. Una vez determinado el impacto de los cambios, se podrían tomar decisiones sobre los entornos de prueba, los datos de prueba o la UAT.

Las pestañas se usaban para riesgos y suposiciones, pero podían usarse para cualquier cosa. tu equipo pueda necesitar. Un formato flexible como una hoja de cálculo significa que puedes adáptalo para que funcione para usted.

Esta información se puede utilizar de varias maneras diferentes. Se puede utilizar para determinar dónde concentrar sus esfuerzos de pruebas exploratorias, o tal vez ayudar a crear una matriz de prueba de alto nivel para asegurarse de tocar todas las áreas durante su prueba.

una pizarra

Si tu equipo es informal y tiene lanzamientos pequeños, cualquier tipo de documentación tal vez demasiado. A veces es suficiente enumerar los riesgos y supuestos sobre una pizarra o en fichas. Janet ha utilizado una pizarra para gestionar riesgos,

y funcionó bastante bien. Si un riesgo realmente se convertía en un problema, el resultado se documentaba y se tachaba. Fue fácil agregar nuevos riesgos y estrategias de mitigación, y la lista era visible para todo el equipo. Esto también se podría hacer en una página wiki.

No podemos enfatizar lo suficiente que necesita conocer a su equipo y sus necesidades.

Lista de pruebas automatizadas

A veces es posible que se le solicite que presente más información a sus clientes, como una lista de casos de prueba. Si su equipo tiene una herramienta de la cual pueda extraer una lista de nombres de casos de prueba, podría proporcionar esta lista fácilmente a cualquiera que la necesite. Esto presentaría un plan de prueba más detallado de tipo tradicional, pero no estaría disponible hasta después de que se escribieran las pruebas. No recomendamos dedicar tiempo a esto porque no vemos valor agregado, pero a veces esta lista puede ser necesaria para la evaluación de riesgos o la auditabilidad.

PREPARÁNDOSE PARA LA VISIBILIDAD

Si su equipo recién está comenzando con el desarrollo ágil, asegúrese de contar con la infraestructura necesaria para sus primeras iteraciones. Puede cambiar la forma en que sigue el progreso a medida que avanza, y sus retrospectivas le ayudarán a sacar a la luz estos problemas. Si tiene problemas para completar el trabajo planificado para cada iteración, tal vez necesite gráficos más visibles o ayudas visuales que le ayuden a medir el progreso y realizar ajustes a mitad de la iteración. ¿Tienen sus clientes alguna forma de saber cómo avanza la iteración y qué historias se completan? Tómese un tiempo antes de cada iteración para evaluar si estás recibiendo el tipo de retroalimentación adecuado para realizar un seguimiento de las pruebas.

Seguimiento del estado y las tareas de prueba

Los equipos ágiles eficaces que todos conocemos siguen esta sencilla regla: "Ninguna historia está terminada hasta que se prueba". Esta regla se puede ampliar para decir que no solo se debe probar la historia, sino que también se debe registrar el código, debe tener pruebas automatizadas que se ejecuten mediante un proceso de construcción continuo, debe estar documentado o cualquiera que sea el criterio de "listo" de su equipo. Son. En cualquier momento durante una iteración, es necesario poder evaluar rápidamente cuánto trabajo de prueba queda en cada historia y qué historias están "terminadas". Los tableros de historias o de tareas son ideales para este propósito, especialmente si utilizan códigos de colores para indicar las tareas de prueba frente a las de desarrollo y otros tipos de tareas. Los tableros de corcho, las láminas de acero con imanes, las notas adhesivas tamaño póster o las pizarras blancas funcionan bien. Asigne a cada historia su propia fila y ordénelas por prioridad. Tenga columnas para "por hacer", "trabajo en progreso", "verificar" y "hecho".

La historia de Janet

Comencé con miembros del equipo que habían estado trabajando de manera ágil durante algunos meses con solo un par de programadores y un evaluador. Habían estado usando XPlanner para realizar un seguimiento de sus tareas e historias, y les estaba funcionando bien. Al mismo tiempo que me uní, se agregaron un par de nuevos programadores y los monólogos se volvieron menos efectivos; el equipo no estaba completando las historias que había planeado. Sugerí un guión gráfico y, aunque se mostraron escépticos acerca de mantener dos conjuntos de "tareas", dijeron que lo intentarían.

Tomamos una pared abierta y usamos adhesivos para crear nuestro guión gráfico. Empezamos a hacer monólogos frente al guión gráfico y nuestra discusión se volvió más específica. Proporcionó una forma agradable y visible de saber cuándo se realizaron las tareas y qué quedaba por hacer. Después de un par de meses, el equipo volvió a crecer y tuvimos que trasladar el storyboard a una oficina. También trasladamos allí nuestros stand-ups y nuestras tablas de resultados de pruebas. Sin embargo, la visibilidad constante se perdió y los programadores y evaluadores dejaron de realizar sus tareas.

Tuvimos que reevaluar lo que queríamos hacer. Una talla única no sirve para todos los equipos.

Asegúrese de planificar lo que es adecuado para su equipo.

—Janet

Algunos equipos utilizan fichas de diferentes colores para los diferentes tipos de tareas: verde para pruebas, blanco para codificación, amarillo y rojo para errores. Otros equipos usan una tarjeta por tarea de desarrollo y agregue pegatinas de diferentes colores para mostrar que las pruebas están en progreso o muestran que hay errores que resolver. Usa cualquier método que le permite ver de un vistazo rápido cuántas historias están "terminadas", con todas las tareas de codificación, bases de datos, pruebas y otras tareas completadas. A medida que avanza la iteración, es fácil ver si el equipo va por buen camino, o si necesitas sacar una historia o tener los programadores colaboran en las tareas de prueba.

La historia de Janet

Nuestro guión gráfico (que se muestra en la Figura 15-8) no era muy grande y no teníamos mucho espacio en la pared para expandirlo y tener el tablero de tareas tipo columna normal. En su lugar, decidimos utilizar pegatinas para designar el estado.

Las tarjetas blancas, como las que se muestran en la primera fila de la Figura 15-8, eran tareas regulares, las tarjetas azules designaban historias técnicas como refactorización o picos, y las tarjetas rosas, que se muestran hacia el lado derecho del tablero como el color más oscuro., eran errores que debían solucionarse. Es fácil ver que esta fotografía fue tomada al comienzo de una iteración porque no hay círculos de colores en cada tarjeta. En la esquina superior derecha puedes ver la leyenda. Las pegatinas azules significaban que se había codificado, las verdes indicaban que se había completado (probado) y las rojas significaban que la tarea se consideraba no completada o que un error se había rechazado por no solucionarse. A medida que se completaba una tarea o historia (es decir, una etiqueta verde), se movía a la derecha del tablero.

—Janet



Figura 15-8 Ejemplo de guión gráfico

La historia de Lisa

Durante más de cuatro años, nuestro guión gráfico consistió en un par de láminas de metal, pintadas con los colores de la empresa, utilizando fichas codificadas por colores adheridas al tablero con imanes. La Figura 15-9 muestra una imagen al principio de una iteración. Nuestras tarjetas de tareas también estaban codificadas por colores: blanco para las tareas de desarrollo, verde para las tareas de codificación, amarillo y rojo para los errores y con rayas para las tarjetas no planificadas originalmente en la iteración. El tablero fue tan eficaz a la hora de indicar nuestro progreso que finalmente dejamos de preocuparnos por un gráfico de avance de tareas. Nos permite centrarnos en completar una historia a la vez. También lo usamos para publicar otros gráficos visibles de gran tamaño, como un gran cartel rojo que muestra que la compilación había fallado. Nos encantó nuestra tabla.

Luego, uno de los miembros de nuestro equipo se mudó al extranjero. Intentamos usar una hoja de cálculo junto con nuestro guión gráfico físico, pero nuestro compañero de equipo remoto encontró que la hoja de cálculo era demasiado difícil de usar. Probamos varios paquetes de software diseñados para equipos Scrum, pero eran tan diferentes de nuestro guión gráfico real que no pudimos adaptarnos a usarlos. Finalmente encontramos un producto (Mingle) que se parecía y funcionaba lo suficiente como nuestro tablero físico para que todos, incluida nuestra persona remota, pudieran usarlo. Pintamos nuestro antiguo guión gráfico de blanco y ahora podemos proyectarlo en la pared durante las reuniones de pie.

—Lisa



Figura 15-9 Otro guión gráfico de muestra

Los equipos distribuidos necesitan algún tipo de guión gráfico en línea. Podría ser una hoja de cálculo o un software especializado que imite un guión gráfico físico como lo hace Mingle.

Comunicación de los resultados de las

pruebas Anteriormente, hablamos sobre la planificación de cómo realizar un seguimiento de los resultados de las pruebas. Ahora queremos hablar de comunicarlos eficazmente. Los resultados de las pruebas son una de las formas más importantes de medir el progreso, ver si se escriben y ejecutan nuevas pruebas para cada historia y si todas se aprueban. Algunos equipos publican grandes gráficos visibles de la cantidad de pruebas escritas, ejecutadas y aprobadas. Otros hacen que su proceso de construcción envíe por correo electrónico los resultados de las pruebas automatizadas a los miembros del equipo y a las partes interesadas. Algunas herramientas de integración continua proporcionan herramientas GUI para monitorear las compilaciones y generar resultados.

Hemos oido hablar de equipos que tienen un proyector conectado a la máquina que ejecuta pruebas de FitNesse de forma continua y muestra los resultados de las pruebas en todo momento.

Los resultados de las pruebas son una descripción concreta del progreso del equipo. Si el número de

Las pruebas no aumentan todos los días o cada iteración, eso podría indicar un problema. O el equipo no está escribiendo pruebas (suponiendo que estén desarrollando pruebas primero) o no están completando mucho código. Por supuesto, es posible que estén eliminando el código antiguo y las pruebas que lo acompañaban. Es importante analizar por qué las tendencias van por el camino equivocado. La siguiente sección le brinda algunas ideas sobre los tipos de métricas que quizás desee recopilar y mostrar.

Independientemente de cómo su equipo decida que quiere comunicar su progreso, asegúrese de pensarlo desde el principio y que todos obtengan valor de ello.

Métricas de lanzamiento

Incluimos esta sección aquí porque es importante comprender qué métricas desea recopilar desde el comienzo de un lanzamiento. Estas métricas deberían brindarle retroalimentación continua sobre cómo avanza el desarrollo, para que pueda responder a eventos inesperados y cambiar su proceso según sea necesario. Recuerde, debe comprender qué problema está tratando de resolver con sus métricas para poder realizar un seguimiento de las correctas. Las métricas de las que hablamos aquí son sólo algunos ejemplos de los que puede optar por realizar un seguimiento.

Número de pruebas aprobadas

Muchos equipos ágiles realizan un seguimiento del número de pruebas en cada nivel: unitarias, funcionales, pruebas de historia, GUI, carga, etc. La tendencia es más importante que el número. Tenemos una sensación cálida y confusa al ver aumentar el número de pruebas. Sin embargo, un número sin contexto es sólo un número. Por ejemplo, si un equipo dice que tiene 1000 pruebas, ¿qué significa eso? ¿1000 pruebas dan una cobertura del 10% o del 90%? ¿Qué sucede cuando se elimina el código que tiene pruebas?

Hacer un seguimiento del número de pruebas escritas, ejecutadas y aprobadas a nivel de historia es una forma de mostrar el estado de una historia. La cantidad de pruebas escritas muestra el progreso de las pruebas para impulsar el desarrollo. Saber cuántas pruebas no se han superado todavía le da una idea de cuánto código aún queda por escribir.

Una vez que se supera una prueba, debe permanecer "verde" siempre que la funcionalidad esté presente en el código. Los gráficos del número de pruebas aprobadas y fallidas a lo largo del tiempo muestran si hay un problema con las fallas de regresión y también muestran el crecimiento de la base del código. Una vez más, lo importante es la tendencia. Esté atento a las anomalías.

Este tipo de mediciones se pueden informar de forma sencilla y seguir siendo eficaces.

Full Build						
December 2007						
Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
					1	2
2879 JUnits Passed	2879 JUnits Passed	2878 JUnits Passed	2880 JUnits Passed	2888 JUnits Passed	8	9
10	11				15	16
Tests not passing	Tests not passing	2888 JUnits Passed	2906 JUnits Passed	2906 JUnits Passed		
Cruise Control Issues	Cruise Control Issues	2956 JUnits Passed	2960 JUnits Passed	2956 JUnits Passed	22	23
2958 JUnits Passed	Merry Christmas!	2958 JUnits Passed	2958 JUnits Passed	2958 JUnits Passed	29	30
2958 JUnits Passed - See ya next year!						

Figura 15-10 Correo electrónico con el resultado completo de la compilación del equipo de Lisa

La historia de Lisa

Mi equipo envía por correo electrónico un calendario codificado por colores todos los días que muestra si el "completo build" con el conjunto completo de pruebas de regresión aprobadas cada día (consulte la Figura 15-10).

Dos días "rojos" seguidos (el color más oscuro) son motivo de preocupación y lo notan tanto la dirección como el equipo de desarrollo. Ver los resultados de las pruebas visuales ayuda a la organización a trabajar juntos para corregir las pruebas fallidas o cualquier otro problema que provoque que la compilación no se ejecute, como problemas de hardware o de base de datos.

—Lisa

Hay diferentes formas de medir el número de pruebas. Elige uno y prueba mantener la coherencia en todos los ámbitos con todo tipo de pruebas; de lo contrario, sus métricas pueden resultar confusas. Medir el número de guiones de prueba o clases es una manera, pero cada uno puede contener múltiples casos de prueba individuales o "afirmaciones", por lo que Puede ser más exacto contarlos.

Si va a contar las pruebas, asegúrese de informar la información para que pueda ser usado. Los correos electrónicos de compilación o las IU de estado de compilación pueden comunicar la cantidad de pruebas ejecutar, aprobar y fallar en varios niveles. El equipo de atención al cliente puede estar contento

para ver esta información solo al final de cada sprint, en la revisión del sprint o un correo electrónico.

Independientemente de las métricas que elija recopilar, asegúrese de que el equipo las acepte.

La historia de Janet

Comencé un nuevo contrato con un equipo que había estado trabajando en metodología ágil durante un par de años y habían desarrollado una gran cantidad de pruebas funcionales automatizadas. Comencé a realizar un seguimiento de la cantidad de pruebas que pasaban cada día. El equipo no vio ningún problema cuando la tendencia mostró que cada vez se pasaban menos pruebas. Las pruebas unitarias se mantuvieron y estaban haciendo lo que se suponía que debían hacer, por lo que el equipo se sintió confiado en el lanzamiento. Parecía que esto sucedía con cada lanzamiento, y el equipo pasaba la última semana antes del lanzamiento para pasar todas las pruebas. Era costoso mantener las pruebas, pero el equipo no quería demorarse para arreglarlas. Todos estaban de acuerdo con esto excepto yo.

No vi cómo arreglar las pruebas en esa fecha tardía podría garantizar que se capturaran los resultados esperados correctos. Sentí que corríamos el riesgo de obtener falsos positivos.

Al comienzo del siguiente ciclo de lanzamiento, logré que el equipo aceptara intentar arreglar las pruebas cuando fallaran. No pasó mucho tiempo para que el equipo se diera cuenta de que no era tan difícil arreglar las pruebas tan pronto como supimos que estaban rotas, y encontramos muchos problemas temprano que normalmente no se detectaban hasta mucho más tarde. El equipo pronto se fijó el objetivo de que el 95% de las pruebas pasaran en todo momento.

También nos dimos cuenta de lo frágiles que eran las pruebas. El equipo hizo un esfuerzo concertado para refactorice algunas de las pruebas más complejas y elimine las redundantes. Con el tiempo, se redujo el número de pruebas de alto nivel, pero se incrementó la calidad y cobertura.

Empezamos midiendo las tasas de aprobación, pero terminamos con muchas más.

—Janet

No se deje atrapar tanto por las medidas reales que no reconozca otros efectos secundarios de la tendencia. Esté abierto a ajustar lo que está midiendo si es necesario.

Cobertura de código

La cobertura del código es otra métrica tradicional. ¿Cuánto de nuestro código se ejercita en nuestras pruebas? Hay excelentes herramientas comerciales y de código abierto disponibles, que pueden integrarse en su proceso de construcción para que que sepa de inmediato si la cobertura ha aumentado o disminuido. Como ocurre con la mayoría de las métricas, la tendencia es lo que hay que observar. La Figura 15-11 muestra una cobertura de código de muestra. informe.

GHIDRAH

Resumen de cobertura general

Nombre	Clase, %	Método, %	Bloque, %	Línea, %
todas las clases	95% (1727/1809)	77% (13605/17678)	72% (201131/279707)	75% (43454,5/58224)

Resumen de estadísticas generales

paquetes totales:	240
archivos ejecutables totales:	1329
clases totales:	1809
métodos totales:	17678
líneas ejecutables totales:	58224

WHITNEY

Resumen de cobertura general

Nombre	Clase, %	Método, % 8%	Bloque, %	Línea, %
Todas las clases	15% (109/737)	(669/8760)	4% (16292/363257)	5% (3713,7/80358)

Resumen de estadísticas generales

paquetes totales:	46
archivos ejecutables totales:	655
clases totales:	737
métodos totales:	8760
líneas ejecutables totales:	80358

Figura 15-11 Informe de cobertura de código de muestra del equipo de Lisa. “Ghidrah” es la nueva arquitectura; “Whitney” es el sistema heredado.

Las figuras 15-12 y 15-13 son dos ejemplos de tendencias que funcionan juntas.

La Figura 15-12 muestra una tendencia del número total de métodos en cada iteración. La figura 15-12 es la cobertura del código coincidente. Estos ejemplos muestran por qué las gráficas deben ser vistos en contexto. Si sólo miras el primer gráfico que muestra

Con la cantidad de métodos, solo obtendrás la mitad de la historia. El número de métodos está aumentando, lo que parece bueno, pero en realidad la cobertura está disminuyendo.

No sabemos el motivo de la disminución de la cobertura, pero debería ser una gatillo para preguntar al equipo: “¿Por qué?”

Recuerde que estas herramientas sólo pueden medir la cobertura del código que ha escrito. Si se perdió alguna funcionalidad, su informe de cobertura de código no

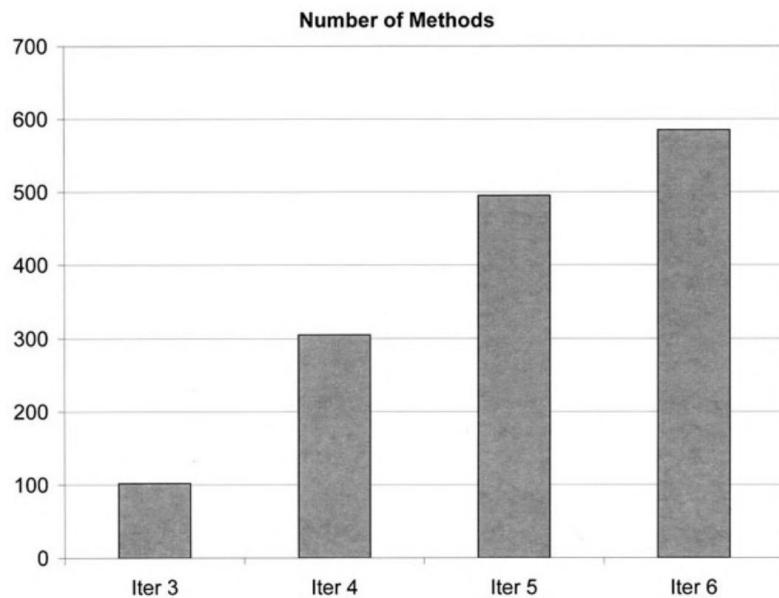


Figura 15-12 Tendencia del número de métodos

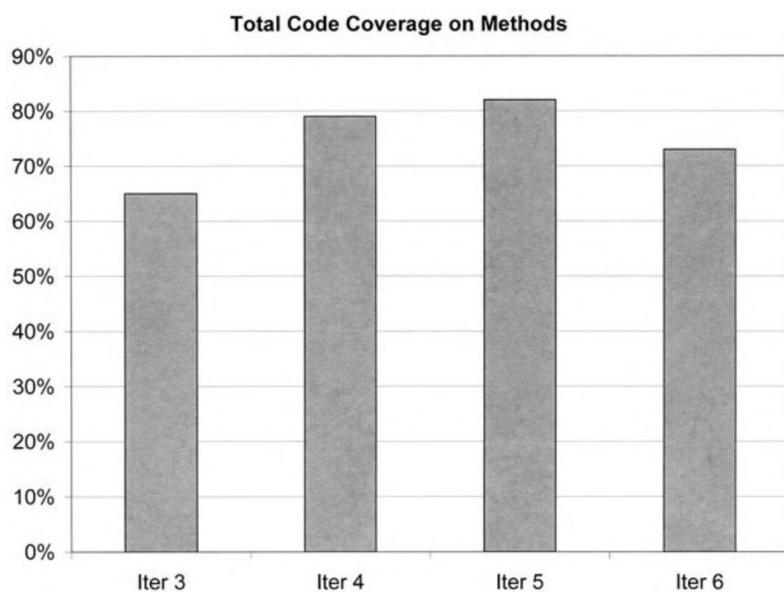


Figura 15-13 Cobertura de prueba

sacar eso a la luz. Es posible que tenga una cobertura de código del 80% con sus pruebas, pero te falta el 10% del código que deberías tener. Impulsar el desarrollo con Las pruebas ayudan a evitar este problema, pero no valoran más las estadísticas de cobertura del código. de lo que se merecen.

Sepa lo que está midiendo

Alessandro Collino, ingeniero en informática e información de Onion SpA que trabaja en proyectos ágiles, nos contó una experiencia en la que la cobertura del código cayó repentina y desastrosamente. Su ágil equipo desarrolló middleware para un sistema operativo en tiempo real en un sistema integrado. Él explicó:

Se siguió un enfoque TDD para desarrollar una gran cantidad de buenas pruebas unitarias orientadas a lograr una buena cobertura de código. Escribimos muchas pruebas de aceptación efectivas para verificar todas las funcionalidades complejas. Después de eso, instrumentamos el código con una herramienta de cobertura de código y alcanzamos una cobertura de declaración del 95%.

El código que no se pudo probar se verificó mediante inspección, lo que los llevó a declarar el 100 % de la cobertura del estado de cuenta después de diez sprints de cuatro semanas.

Después de eso, el cliente nos pidió que agregáramos una pequeña característica antes de entregar el producto de software. Implementamos esta solicitud y aplicamos la optimización del código del compilador.

Esta vez, cuando realizamos las pruebas de aceptación, el resultado fue desastroso; ¡El 47% de las pruebas de aceptación fallaron y la cobertura de las declaraciones se redujo al 62%!

¿Qué pasó? El problema resultó ser que se habilitaba la optimización del compilador pero con una configuración incorrecta. Debido a esto, un valor clave se leía una vez cuando se iniciaba la aplicación y se almacenaba en un registro de la CPU.

Incluso cuando la variable se modificó en la memoria, el valor en el registro de la CPU nunca fue reemplazado. La rutina seguía leyendo este mismo valor obsoleto en lugar del valor actualizado correcto, lo que provocaba que las pruebas fallaran.

Alessandro concluye: "La lección aprendida de este ejemplo es que la habilitación de las opciones de optimización del compilador debe planificarse al comienzo del proyecto. Es un error activarlos en las etapas finales del proyecto".

Las buenas métricas requieren una buena planificación. El esfuerzo extra puede darte más datos significativos. Los miembros del equipo de Pierre Veragen utilizan una línea de base de prueba de rotura técnica para saber si su métrica de cobertura de código es significativa. Introducen manualmente un defecto en cada método y luego ejecutan sus pruebas para asegurarse las pruebas detectan el problema. Algunas pruebas simplemente aseguran que el código devuelva algo valor, cualquier valor. El equipo de Pierre se asegura de que las pruebas devuelvan el valor correcto . En De esta manera, pueden determinar si la cobertura de su prueba es lo suficientemente buena.

La cobertura del código es sólo una pequeña parte del rompecabezas. Úselo como tal, no lo dice usted sabe qué tan buenas son sus pruebas, pero solo si se ejecutó una cierta porción de código durante la prueba. No le dice si se ejecutaron diferentes rutas a través de la aplicación, cualquiera. Comprenda su aplicación e intente identificar sus áreas de mayor riesgo. y establecer una meta de cobertura que sea mayor para esas áreas que para las áreas de bajo riesgo. No olvide incluir también sus pruebas funcionales en el informe de cobertura.

Métricas de defectos

A medida que su equipo establece objetivos relacionados con los defectos, utilice métricas adecuadas para medir avances hacia esos objetivos. Hay tendencias que querrás monitorear, para toda la versión, y hay algunos que son específicos de la iteración. Por ejemplo, si intenta lograr cero defectos, es posible que desee realizar un seguimiento de los defectos abiertos. errores al final de cada iteración, o cuántos errores se encontraron después del desarrollo pero antes del lanzamiento. La mayoría de nosotros estamos interesados en saber cuántos Se han informado defectos después de que el código está en producción, lo cual es algo completamente diferente. Estos problemas te lo dirán después del hecho. qué tan bien le fue a su equipo en la última versión, pero no qué tan bien le está yendo a usted en la versión actual. Es posible que le den alguna indicación de qué procesos necesita cambiar para reducir el número de defectos. El equipo de Lisa está más preocupado por los defectos de producción encontrados en el "nuevo" código que se reescribió. en la nueva arquitectura. Están trabajando duro para producir este nuevo código con cero defectos, por lo que necesitan saber qué tan bien lo están haciendo. ellos esperan que Los errores se encontrarán con bastante frecuencia en el sistema heredado, donde sólo la funcionalidad más crítica está cubierta por pruebas de humo de GUI automatizadas, y hay pocos pruebas unitarias automatizadas y detrás de la GUI.

Conocer la tasa de defectos del código heredado podría ser una buena justificación para refactorizarlo o reescribirlo, pero la principal prioridad del equipo es hacer un buen trabajo con el nuevo código, por lo que agrupan los errores por código "nuevo" y "antiguo", y se centran en el Errores "nuevos".

 Puede encontrar más información sobre los sistemas de seguimiento de defectos en el Capítulo 5, "Transición de procesos tradicionales".

Asegúrese de que su base de datos de errores pueda rastrear lo que desea medir. Puedes tendrá que hacer algunos cambios tanto en la base de datos como en su proceso para obtener el datos que necesitas. Por ejemplo, si desea medir cuántos defectos hubo encontrado en producción después de un lanzamiento, debe asegurarse de tener entorno y versión como campos obligatorios, o asegurarse de que las personas que ingresan los errores siempre los completan.

Debido a que los sistemas de seguimiento de defectos a menudo se utilizan para fines además del seguimiento errores, asegúrese de no confundir los números. Una solicitud de actualización manual de la base de datos no refleja necesariamente un problema con el código existente. Usar su herramienta de seguimiento de defectos correctamente para garantizar que sus métricas sean significativas.

La historia de Lisa

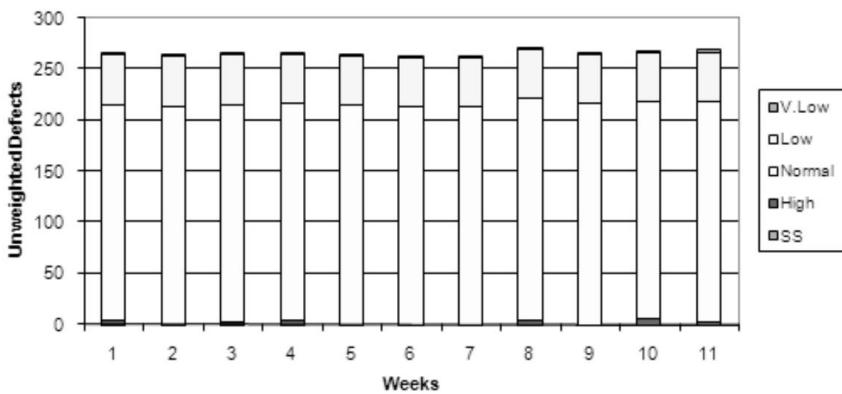
Evalué periódicamente las métricas que está informando y vea si siguen siendo relevantes.

La Figura 15-14 muestra dos informes de defectos que el equipo de Lisa utilizó durante años.

Cuando hicimos la transición a la metodología ágil por primera vez, los gerentes y otras personas examinaron estos informes para ver el progreso resultante del nuevo proceso. Cuatro años más tarde, nuestro ScrumMaster descubrió que ya nadie leía estos informes, por lo que dejamos de producirlos. En ese momento, las tasas de nuevos defectos se habían reducido drásticamente y a nadie realmente le importaban los viejos defectos que aún persistían en el código heredado.

—Lisa

Unweighted Defects by Priority



Unweighted Inflow and Outflow - Last 12 Weeks

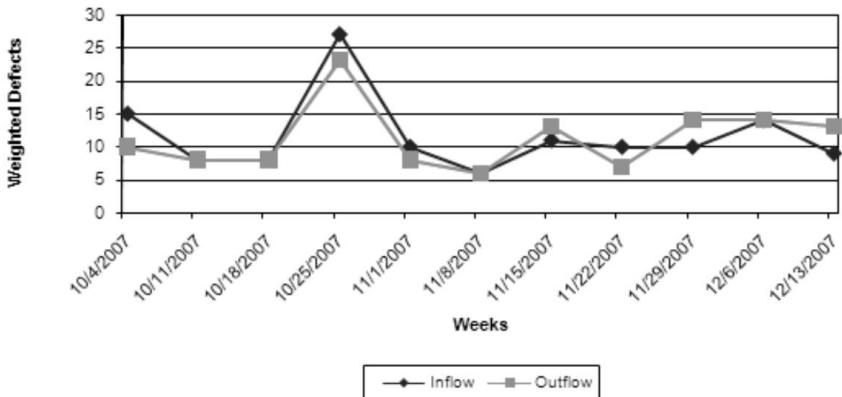


Figura 15-14 Ejemplos de informes de defectos utilizados (y que ya no se utilizan) por el equipo de Lisa

La planificación del lanzamiento es un buen momento para evaluar el ROI de las métricas que ha estado rastreando. ¿Cuánto esfuerzo estás dedicando a recopilar e informar la información? ¿Métrica? ¿Te dicen lo que necesitas saber? ¿El código que publicas? ¿Cumple con los estándares de calidad interna de su equipo? ¿Está aumentando el porcentaje de cobertura del código? ¿El equipo está cumpliendo sus objetivos de reducir la cantidad de defectos que llegan a producción? Si no, ¿hubo una buena razón?

Las métricas son sólo una pieza del rompecabezas. Utilice su lanzamiento, tema o proyecto planificar reuniones para volver a centrarse en ofrecer valor empresarial cuando el negocio lo necesita. Tómate un tiempo para conocer las funciones que estás a punto de desarrollar. No se deje atrapar por comprometerse con sus planes: la situación está destinada a cambiar. En lugar de eso, prepárese para realizar las actividades correctas y obtener los recursos a tiempo para satisfacer las prioridades de los clientes.

RESUMEN

Mientras su equipo elabora su plan para un nuevo tema o lanzamiento, mantenga el tema principal puntos de este capítulo en mente.

Al dimensionar una historia, considere diferentes puntos de vista, incluido el valor comercial, el riesgo, la implementación técnica y cómo se utilizará la función. Haga preguntas aclaratorias, pero no se estanque en detalles.

Los evaluadores pueden ayudar a identificar la "parte delgada" o la "ruta crítica" a través de un conjunto de funciones para ayudar a priorizar las historias. Programe historias de alto riesgo con anticipación si es posible que requieran pruebas adicionales con anticipación.

El tamaño del esfuerzo de prueba para una historia ayuda a determinar si esa historia está dentro del alcance del lanzamiento.

Los evaluadores pueden ayudar al equipo a pensar en cómo las nuevas historias afectarán al sistema en general.

Planifique tiempo y recursos de prueba adicionales cuando las características puedan afectar sistemas o subsistemas desarrollados por equipos externos.

A medida que el equipo identifique el alcance del lanzamiento, evalúe el alcance de las pruebas y presuponga suficiente tiempo y recursos para ello.

Dedique algo de tiempo durante la planificación del lanzamiento para abordar las inquietudes sobre la infraestructura, el entorno de prueba y los datos de prueba.

Un plan de prueba ligero y ágil puede ayudar a garantizar que todas las consideraciones de prueba se aborden durante la vida del lanzamiento o proyecto.

Considere alternativas a los planes de prueba que podrían ser más apropiadas para su equipo; matrices de prueba, hojas de cálculo o incluso una pizarra pueden ser suficientes.

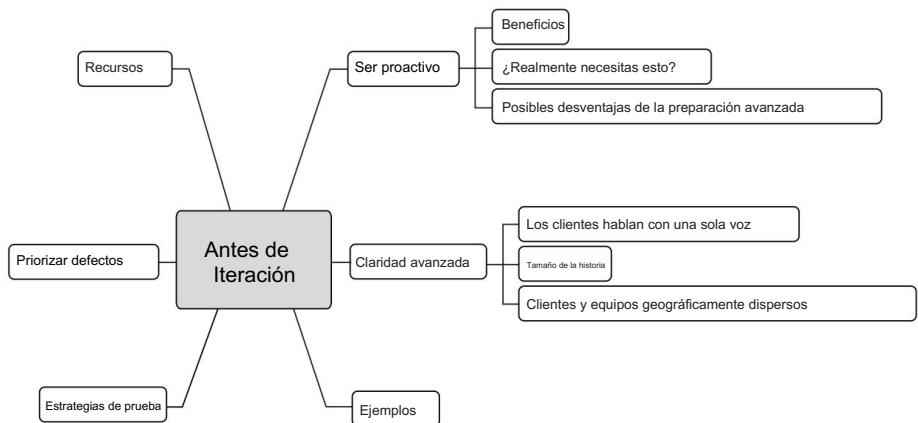
Es posible que la planificación formal de la liberación no sea apropiada para su situación. En ausencia de una planificación de lanzamiento, considere identificar y discutir al menos las primeras historias que deberían realizarse primero.

Planifique qué métricas desea capturar durante la vida del lanzamiento; Piense en el problema que está intentando resolver y capture solo aquellas métricas que sean significativas para su equipo.

Esta página se dejó en blanco intencionalmente.

Capítulo 16

ARRANCAR A TODA VELOCIDAD



En el desarrollo ágil, generalmente nos gusta realizar las tareas "justo a tiempo". No podemos ver las curvas del camino que tenemos por delante, por lo que nos concentramos en las actividades que tenemos entre manos. Por otra parte, queremos empezar a trabajar cuando comenzemos cada nueva iteración. Eso puede requerir un poco de preparación. Hornear es una buena analogía aquí. Decides que quieres hornear galletas porque viene alguien. Antes de comenzar, asegúrate de tener los ingredientes adecuados. Si no lo haces, vas a comprar lo que necesitas o eliges otro tipo para hacer.

No se excede: si una actividad previa a la iteración no le ahorra tiempo durante la iteración o no le ayuda a hacer un mejor trabajo, no la haga antes de la iteración. Haga lo que sea apropiado para su equipo y siga experimentando. Tal vez realices algunas de estas actividades después de que comience la iteración. A continuación se ofrecen algunas ideas en las que pensar que podrían ayudarle a "integrar la calidad" a su producto.

SER PROACTIVO

En el Capítulo 2, "Diez principios de un evaluador ágil", explicamos cuán ágil Los evaluadores tienen que cambiar su forma de pensar. En lugar de esperar a que llegue el trabajo nosotros, desarrollamos una actitud proactiva en la que nos levantamos y buscamos maneras de

contribuir. Con el ritmo rápido y constante del desarrollo ágil, es fácil Sumérgete en las historias de la versión actual. Estamos muy ocupados asegurándonos Hemos cubierto las funciones con pruebas iniciales, realizando pruebas exploratorias para asegurarnos de que hemos entendido los requisitos comerciales y automatizando pruebas de regresión adecuadas, es difícil pensar en otra cosa. Sin embargo, es A veces es apropiado tomarnos un poco de tiempo para ayudar a nuestros clientes y a nuestros El equipo se prepara para la siguiente iteración. Cuando nuestro equipo está a punto de hacer algo nuevo. terreno, o trabajar en historias complejas y arriesgadas, algo de trabajo antes de la iteración puede ayudar a maximizar la velocidad de nuestro equipo y minimizar la frustración.

Seguramente no queremos pasar todo nuestro tiempo en reuniones o planificando historias. que podrían volver a priorizarse. Sin embargo, si podemos hacer nuestra planificación de iteración ir más rápido y reducir el riesgo de las historias que estamos a punto de emprender, es Vale la pena investigar un poco y hacer una lluvia de ideas antes de comenzar la iteración.

Beneficios

Trabajar en historias antes de la iteración puede ser especialmente útil para equipos que están divididos en diferentes ubicaciones geográficas. Al trabajar por delante, hay tiempo para hacer llegar información a todos y darles la oportunidad de dar sus aportes.

El problema es que estamos tan ocupados durante cada iteración breve y ágil que es Es difícil encontrar tiempo para reunirse sobre las historias de la próxima iteración, y mucho menos para comenzar. escribir casos de prueba. Si sus iteraciones siempre van bien, con historias entregadas de forma incremental y con mucho tiempo para realizar pruebas, y el software entregado coincide expectativas del cliente, es posible que no necesite tomarse el tiempo para prepararse con anticipación. Si su equipo tiene problemas para terminar historias o termina con grandes discrepancias entre el comportamiento real y deseado de las funciones, puede ser posible realizar un poco de planificación anticipada. ahorrarle tiempo durante la iteración.

La historia de Lisa

Nuestro equipo solía sentir que no teníamos tiempo para planificar con anticipación la siguiente iteración. Después de muchas experiencias de malinterpretar historias y de que superaran con creces las estimaciones, y de descubrir que la mayoría de los "errores" eran requisitos no cumplidos, decidimos presupuestar tiempo en la iteración para comenzar a hablar de la siguiente. Ahora todo el equipo, incluido el propietario del producto y otros clientes según sea necesario, se reúnen durante una hora o menos el día antes de la reunión de planificación para nuestro próximo sprint.

A esto, entre risas, lo llamamos la reunión de "planificación previa". Repasamos las historias para la próxima iteración. El propietario del producto explica el propósito de cada historia. Repasa las condiciones de satisfacción empresarial y otros elementos de sus listas de verificación de historias, y da ejemplos de comportamiento deseado. Realizamos una lluvia de ideas sobre posibles riesgos y dependencias, e identificamos hilos de acero cuando corresponde.

A veces basta con dedicar unos minutos a escuchar las opiniones del propietario del producto. explicación de los cuentos. En otras ocasiones, nos tomamos el tiempo para diagramar pequeños fragmentos de una historia en la pizarra. La Figura 16-1 muestra un diagrama de ejemplo en el que analizamos tanto los detalles del flujo de la interfaz de usuario como las tablas de la base de datos. Tenga en cuenta los números de los "hilos". El hilo número 1 es nuestro camino crítico. El hilo número 2 es la segunda capa, y así sucesivamente. Subimos fotos de estos diagramas a la wiki para que nuestro desarrollador remoto también pueda verlas.

Podemos empezar a pensar en qué tarjetas de tareas podríamos escribir al día siguiente y qué enfoque podríamos adoptar para cada historia. Por alguna razón, poder reflexionar sobre las historias durante la noche nos hace más productivos en la reunión de planificación de iteraciones del día siguiente. Despues de hacer esto durante algunas iteraciones, en general dedicamos menos tiempo a planificar la iteración, a pesar de que tuvimos dos reuniones para hacerlo.

A veces invitamos a otros clientes a hablar sobre historias que les afectan directamente. Si no están disponibles en ese momento, todavía tenemos tiempo antes de nuestra reunión de planificación de iteración para hablar con ellos y aclarar la historia.

En una reunión de planificación previa, el propietario de nuestro producto presentó una historia sobre la obtención de datos de rendimiento para fondos mutuos. Envíaríamos un archivo al proveedor que contiene una lista de fondos mutuos, y el proveedor proporcionaría un archivo XML en un sitio web con la información más reciente sobre el rendimiento de esos fondos. Luego cargaríamos esos datos en nuestra base de datos.

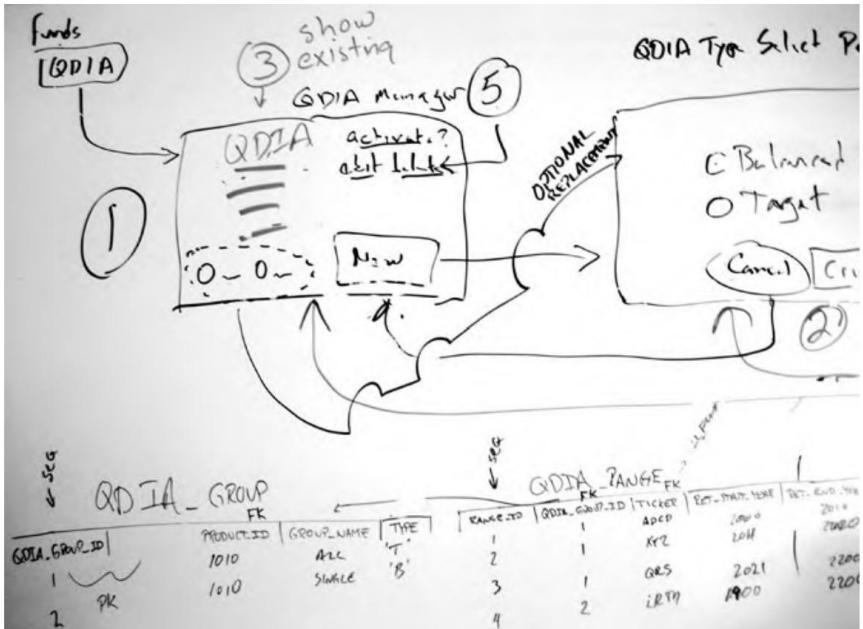


Figura 16-1 Ejemplo de diagrama de pizarra de planificación

En la reunión de planificación previa, hicimos preguntas como: "¿Cuál es el formato del archivo que enviamos al proveedor?" "¿La fecha 'a partir' de cada fondo es siempre el último día del mes?" "¿Existe alguna seguridad en el sitio web que contiene el XML?" "Voluntad

¿Alguna vez obtenemos un registro para el mismo fondo y la fecha "al momento" que tiene datos nuevos, o podemos ignorar los registros con una fecha que ya tenemos en nuestra base de datos?

En la reunión de planificación de iteraciones del día siguiente, el propietario del producto había obtenido respuestas a todas nuestras preguntas. Escribir tarjetas de tareas fue rápido y la codificación pudo proceder con suposiciones correctas.

A menudo, encontramos una solución mucho más simple para una historia cuando la discutimos en la discusión de planificación previa a la iteración. Descubrimos que para ir rápido, primero teníamos que reducir la velocidad.

—Lisa

¿Realmente necesitas esto?

Es posible que su equipo no necesite mucha o ninguna preparación previa. Pierre Veragen y Erika Boyer le describieron a Lisa cómo funcionan sus equipos en iLevel por Weyerhaeuser escriban pruebas de aceptación de usuarios juntos en su reunión de inicio de iteración.

Estas pruebas, que fueron escritas en una página wiki o alguna herramienta similar junto con la narrativa de la historia, se utilizan más adelante cuando los miembros del equipo escriben tarjetas de tareas para cada historia y comenzar a escribir más pruebas y código. Los ejemplos se convierten en pruebas ejecutables. Porque las pruebas cambian a medida que el equipo aprende más sobre la historia, el equipo puede optar por no mantener las originales que fueron escritas al principio. Manténgalo simple al principio y profundice en los detalles más adelante.

Posteriormente, Lisa observó una de sus sesiones de planificación y vio de primera mano cuán efectiva fue esta técnica. Incluso cuando el gerente de producto proporciona ejemplos concretos, convertirlos en pruebas puede eliminar los requisitos faltantes. Su equipo no necesitaba hacer esto antes de la sesión de planificación de iteraciones, pero no es el caso con todos los equipos.

La historia de Lisa

A mi equipo le gustó la práctica de escribir pruebas juntos, pero como estábamos escribiendo tarjetas de tareas durante la planificación de la iteración, decidimos escribir pruebas de aceptación del usuario juntos durante la reunión de planificación previa. Descubrimos que esto mantuvo nuestras discusiones enfocadas y entendimos cada historia más rápidamente. También hicimos un mejor trabajo al entregar exactamente lo que el cliente tenía en mente. Nuestros clientes notaron una diferencia en la calidad y el propietario de nuestro producto nos animó a continuar con esta práctica.

—Lisa

Experimente con breves debates previos a la iteración y sesiones de redacción de pruebas.

Le llevará varias iteraciones encontrar el ritmo de su equipo y descubrir si las discusiones avanzadas sobre la historia lo hacen más productivo durante la iteración.

Potenciales desventajas de la preparación anticipada

Existe el riesgo de "trabajar por delante". Podría dedicar tiempo a aprender más detalles sobre una función solo para que la gente de negocios vuelva a priorizar en el último momento. minuto y posponer esa función indefinidamente. Invierta tiempo de preparación cuando sea adecuado. Cuando sabes que tienes un tema o una historia compleja por delante, y tiene una fecha límite estricta como la que tuvo el equipo de Lisa con la historia de la declaración, Considere pasar algún tiempo desde el principio comprobando diferentes puntos de vista. La única razón para discutir historias con anticipación es ahorrar tiempo durante la iteración. planificación y durante el desarrollo. Una comprensión más profunda del comportamiento de las funciones puede acelerar las pruebas y la codificación, y puede ayudar a garantizar que se entregue la funcionalidad correcta.

Si su situación es tan dinámica que las historias podrían volver a priorizarse el día en que comienza la iteración, no vale la pena intentar hacer esta planificación. En lugar de eso, haz asegúrese de reservar tiempo para estas discusiones durante su reunión de planificación.

CLARIDAD AVANZADA

Al propietario del producto de Lisa, Steve Perkins, se le ocurrió el término "claridad avanzada".

Diferentes partes de cada organización tienen diferentes prioridades y agendas.

Por ejemplo, Business Development busca nuevas funciones para atraer nuevos negocios, mientras que Operaciones está priorizando características que reducirían el Número de llamadas telefónicas de los usuarios. El equipo de desarrollo intenta comprender la gama de necesidades empresariales y tener una idea del trabajo de cada individuo.

Con muchas agendas diferentes, alguien necesita decidir qué historias deberían implementarse en la próxima iteración. Debido a que hay muchas maneras de implementar cualquier historia determinada, alguien tiene que decidir los requisitos específicos y capturarlos en forma de ejemplos, condiciones de satisfacción y pruebas. casos. Steve reúne a todos para ponerse de acuerdo sobre el valor que quieren de cada uno. historia y proporcionar "claridad avanzada".

Los clientes hablan con una sola voz

Scrum proporciona el papel útil del propietario del producto para ayudar a todos los clientes a "hablar con una sola voz". Ya sea que estés o no en un equipo Scrum, encuentra alguna forma de ayudar a sus clientes a ponerse de acuerdo sobre la prioridad de las historias y cómo

Se deben implementar los componentes de cada historia. El apoyo administrativo es crucial, porque cualquier persona en este rol necesita tiempo y autoridad para hacer que todos estén en la misma página.

Otros equipos utilizan analistas de negocios para ayudar a desarrollar las historias antes de la siguiente iteración. En una organización con la que trabajó Janet, los clientes no estaban disponible a tiempo completo para responder preguntas, pero cada equipo tenía un analista de negocios que trabajó con los clientes para desarrollar los requisitos antes de la reunión de planificación de iteraciones. Si había alguna pregunta que no podía responder en la reunión, el equipo llamaba al cliente directamente o al

El analista hizo un seguimiento inmediatamente después de la reunión.

Como evaluador, querrás participar en las reuniones de redacción de historias y priorización. Haga preguntas que ayuden a los clientes a centrarse en la funcionalidad principal, la valor empresarial crítico que necesitan. Ayude a los participantes a mantenerse enfocados en lo concreto. ejemplos que cristalizan el significado de las historias. En reuniones que involucran múltiples clientes, es fundamental tener un facilitador fuerte y un método para determinar el consenso.

Al igual que con el código, las historias son mejores si tienen lo mínimo. Por ejemplo, un El carrito de compras de Internet necesita alguna forma de eliminar elementos no deseados, pero el La capacidad de mover artículos del carrito a una lista de "guardar para más tarde" probablemente pueda esperar. Puede ser útil hablar de esto antes de la iteración, para que el equipo esté tener claro qué tareas deben planificarse. Concéntrate primero en lo más simple y Utilice un ejemplo para que quede claro.

Obtener todos los puntos de vista

Obtener requisitos de diferentes clientes para una historia, cada uno de los cuales tiene una agenda diferente, podría crear caos. Por eso es esencial para alguien en el equipo del cliente para lograr consenso y coordinar todos los puntos de vista. Esto no significa que no debamos recibir comentarios de diferentes clientes. Como un Probador, estás considerando cada historia desde múltiples puntos de vista. ayuda a Sepa lo que significa la historia para las personas que desempeñan diferentes roles.

La historia de Lisa

Cuando mi empresa decidió rediseñar los estados de cuenta trimestrales de los participantes del plan de jubilación, diferentes personas del sector empresarial querían cambios por diferentes motivos. Los administradores del plan querían un diseño claramente comprensible que minimizara la cantidad de llamadas de participantes confundidos al servicio de atención al cliente.

Por ejemplo, querían que el estado de cuenta mostrara la fecha y el monto de la contribución más reciente del participante. Esto ayuda a la participante a saber si su

El empleador se retrasa en contabilizar las contribuciones a las cuentas. El desarrollo empresarial quería nuevas funciones llamativas que pudieran vender a clientes potenciales, como gráficos de rendimiento por categoría de fondos. Nuestra persona jurídica necesitaba algún texto y datos nuevos en la declaración para cumplir con las regulaciones federales.

Mientras el propietario del producto equilibraba todas las diferentes necesidades y presentaba el resultado final diseño de la declaración, todavía era importante para nuestro equipo comprender el propósito detrás de cada nueva información. Necesitábamos hablar directamente con expertos comerciales en las áreas de administración de planes, desarrollo comercial y legal, y con el propietario del producto. Un tester y un programador se reunieron con cada grupo para recoger los diferentes puntos de vista. Al hacer esto antes de comenzar con las historias para recopilar y mostrar datos, entendimos los requisitos mucho más claramente e incluso hicimos sugerencias para producir la información de manera más eficiente.

—Lisa

Asegúrese de ser lo más eficiente posible en la recopilación de estos datos. A veces eso
Es importante que todo el equipo comprenda la necesidad y, a veces, es
suficiente para que uno o dos miembros del equipo realicen la investigación.

Tamaño de la historia

Mientras discute historias para la próxima iteración con los miembros del equipo del cliente, haga preguntas para ayudarlos a asegurarse de que cada historia brinde el valor necesario. Este es un buen momento para identificar nuevas historias que quizás necesiten escribir. Aunque el equipo evaluó las historias previamente, es posible que descubras que una historia es más grande de lo que se pensaba anteriormente. Incluso podrías descubrir que una característica puede ser implementado de manera más simple de lo planeado, y la historia puede ser más pequeña.

A veces se hacen suposiciones cuando se dimensiona la historia y, al realizar más investigaciones, resultan ser falsas. Incluso las historias más sencillas merecen una mirada más cercana. Es difícil para cualquier persona recordar todos los detalles de una solicitud.

La historia de Lisa

A continuación se muestran algunos ejemplos de historias que resultaron ser significativamente más grandes o más pequeñas de lo que se pensaba originalmente.

1. La historia consistía en producir un archivo de estados de cuenta para todos los participantes en un plan de jubilación de la empresa, que debía enviarse a un proveedor que imprimiría y enviaría por correo los estados de cuenta. Originalmente se dimensionó asumiendo que todas las declaraciones tenían exactamente tres páginas. Tras una mayor investigación, descubrimos que algunos participantes tenían declaraciones de cuatro páginas, pero el proveedor exigía que todas las declaraciones tuvieran la misma longitud. Nuestros expertos en negocios tuvieron que decidir si tener una función para marcar cualquier plan cuyos participantes tuvieran declaraciones de cuatro páginas y tratarlas manualmente, o cambiar las declaraciones para que tuvieran cuatro páginas. Ese es un esfuerzo mucho mayor que el original.

historia. Después de que comenzamos a desarrollar la historia, los clientes revelaron otro requisito: si faltaba la dirección de algún participante o no era válida, la declaración debía enviarse por correo al empleador. Es razonable, pero no lo sabíamos cuando evaluamos la historia.

2. Nuestros clientes querían comenzar a mostrar el número de teléfono de ventas en varias ubicaciones de la interfaz de usuario. Hay un número de teléfono de ventas diferente para cada socio. sitio, y en ese momento había alrededor de 25 sitios asociados diferentes. Esto sonaba como una historia tan sencilla que ni siquiera se la explicó al equipo. El gerente de desarrollo simplemente le asignó un pequeño valor en puntos y simplemente lo "agregó" a la iteración. Había asumido que el número de teléfono estaba almacenado en la base de datos, cuando en realidad estaba codificado en el HTML de la página de "contacto" de cada socio. Almacenar el número correcto para cada socio en la base de datos y cambiar el código para recuperar el valor hizo que la historia fuera dos veces más grande y no había espacio para ella en esa iteración, por lo que no se completó.
3. Dimensionamos una historia para la interfaz de usuario para permitir a los administradores enviar una Solicitud de un trabajo por lotes para reequilibrar las cuentas de los participantes que cumplieron una determinada condición. Incluía una página de confirmación que mostraba el número de participantes afectados. Debido a que la solicitud estaba en cola para ejecutarse como un trabajo por lotes asincrónico, el código para determinar qué participantes se vieron afectados estaba en el código del trabajo por lotes. Refactorizar el código para obtener el número de participantes en el momento de la solicitud fue un gran trabajo. Después de que comenzamos a trabajar en la historia, le preguntamos al usuario principal de la función si realmente necesitaba ese número al enviar la solicitud y decidió que no era necesario. La historia se volvió mucho más pequeña de lo que se pensaba originalmente. Siempre hacemos preguntas para descubrir el verdadero valor comercial que desean los clientes y eliminamos los componentes que no tienen un buen retorno de la inversión.

—Lisa

Estas historias muestran que algunas preguntas por adelantado pueden ahorrar tiempo durante la iteración que podría dedicarse a descubrir qué hacer con los nuevos descubrimientos.

Sin embargo, reconocemos que no todos los descubrimientos se pueden encontrar temprano. Por ejemplo, en la primera historia, una simple pregunta sobre el tamaño de las declaraciones puede haber evitado una confusión de último momento sobre cómo manejar declaraciones de cuatro páginas. pero es posible que la cuestión de la dirección inexacta no se haya considerado hasta que se siendo codificados o probados.

Sabemos que siempre habrá descubrimientos en el camino, pero si podemos captar Primero, los grandes "errores", que ayudarán al equipo a trabajar de la manera más efectiva posible.

Equipos geográficamente dispersos

Un poco de preparación para la siguiente iteración puede ser útil para equipos que están divididos en diferentes ubicaciones. Equipos que están distribuidos en múltiples ubicaciones. pueden planificar su iteración mediante conferencia telefónica, reunión en línea o teleconferencia. Una práctica que utilizó un equipo de Lisa es asignar a cada equipo un

subconjunto de las próximas historias y pídale que escriban tarjetas de tareas con anticipación. Durante la reunión de planificación, todos pueden revisar todas las tarjetas de tareas y hacer cambios según sea necesario. El trabajo inicial mejora la comunicación, hace que las historias y tareas visibles para todos, y acelera el proceso de planificación.

Por supuesto, esto supone que el equipo está utilizando una historia o tarea electrónica. Junta. El equipo de Lisa utiliza Thoughtwork's Mingle, pero hay muchos otros productos que sirven para este propósito.

Hacer frente a la diversidad geográfica

Hablamos con un equipo que conocemos de una empresa de software que tiene clientes, desarrolladores y evaluadores repartidos por todo el mundo. Los clientes no sólo están lejos del equipo técnico sino que no tienen ancho de banda para estar disponibles para responder las preguntas del equipo de desarrollo. En cambio, el equipo depende de analistas funcionales que entienden tanto el lado comercial de la aplicación a un nivel detallado como la implementación técnica del software.

Estos analistas funcionales actúan como enlaces entre los equipos técnicos y comerciales.

Patrick Fleisch y Apurva Chandra son consultores que trabajaron con esta empresa y actuaron como analistas funcionales en un proyecto para desarrollar software de derechos basado en la web, porque son expertos en este ámbito. Viajaron entre ubicaciones para facilitar la comunicación entre las partes interesadas y los desarrolladores.

Los analistas funcionales trabajaron antes de la iteración, dimensionando y preparando las historias, ayudando al equipo técnico a comprender las historias.

Ingresaron historias en una herramienta en línea y las desarrollaron definiendo casos de prueba, condiciones extremas y otra información que ayudó al equipo técnico a comprender la historia. Documentaron la funcionalidad de alto nivel en una wiki dirigida a los usuarios empresariales.

Apurva y Patrick jugaron un papel clave en la toma de decisiones que el equipo técnico necesitaba para comenzar con las nuevas historias. Su profundo conocimiento técnico y comercial les permitió brindarle al equipo los requisitos que necesitaban para comenzar a codificar, porque los clientes reales no eran

disponible para ellos. David Reed, probador e ingeniero de automatización, nos contó cómo confió en Apurva y Patrick para obtener la información que necesitaba para realizar y automatizar pruebas. Si bien los principios ágiles dicen que se debe colaborar estrechamente con el cliente, en algunas situaciones hay que ser creativo y encontrar otra forma de obtener requisitos comerciales claros.

Si los clientes no están disponibles para responder preguntas y tomar decisiones, Se debe empoderar a otros expertos en el dominio que estén accesibles en todo momento. Guiar al equipo determinando prioridades y expresando el sistema deseado.

comportamiento con ejemplos. A menudo se recurre a evaluadores y analistas de negocios para realizar estas actividades.

EJEMPLOS

Puede notar que hablamos de ejemplos en casi todos los capítulos de este libro. Los ejemplos son una forma eficaz de conocer e ilustrar los objetivos deseados. funcionalidad (y no deseada); Vale la pena usarlos durante todo el ciclo de desarrollo. Nuestro lema fue acuñado por Brian Marick: "Un ejemplo sería útil ahora mismo". (Consulte la Figura 16-2.) Inicie sus discusiones sobre características e historias con un ejemplo realista. La idea ha despegado, de modo que en un En un taller reciente sobre pruebas funcionales estuvimos discutiendo ideas sobre cómo llamarlo "Desarrollo basado en ejemplos".

Cuando los miembros del equipo de Lisa se reúnen con el propietario de su producto para hablar sobre el En la siguiente iteración, le piden ejemplos de comportamiento deseado para cada historia. Esto mantiene la discusión en un nivel concreto y es una manera rápida de aprender cómo Las nuevas funciones deberían funcionar. Tenga una pizarra a mano mientras hace esto y empieza a dibujar. Si algunos miembros del equipo están en un lugar distante, considere usar herramientas que permiten a todos ver diagramas de pizarra y participar en el discusión. Revise ejemplos reales con sus clientes o sus representantes. Al igual que durante la planificación del lanzamiento, considere diferentes puntos de vista: el negocio, usuarios finales, desarrolladores y socios comerciales. A diferencia de la planificación de lanzamientos, usted Estamos viendo muchos más detalles porque estas son las historias que estás planeando. para la siguiente iteración.

Usando ejemplos, puedes escribir pruebas de alto nivel para desarrollar un poco cada historia. más. Es posible que no necesite hacer esto antes de que comience la iteración, pero para completar



Figura 16-2 Etiqueta de ejemplo de Brian Marick

historias complejas, puede ser una buena idea escribir al menos un camino feliz y un caso de prueba de ruta negativa por adelantado. Consideremos la historia de la Figura 16-3.

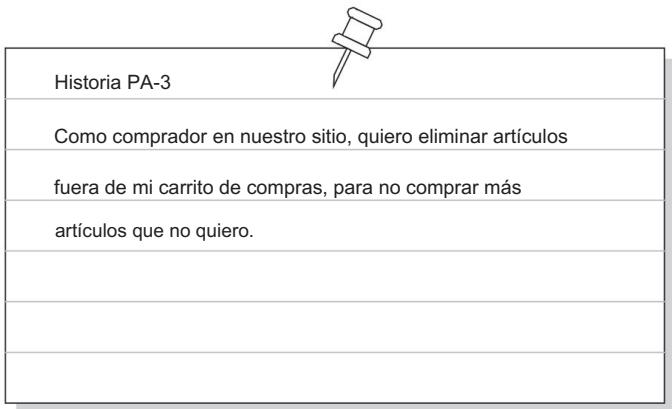


Figura 16-3 Historia para eliminar artículos del carrito de compras

El propietario del producto dibuja la interfaz de usuario deseada en la pizarra. Hay una casilla de verificación "eliminar" junto a cada artículo y un botón "actualizar carrito". El usuario Puede seleccionar uno o más elementos y hacer clic en el botón para eliminarlos. El

Las pruebas de alto nivel podrían ser:

Cuando el usuario hace clic en la casilla de verificación eliminar junto al artículo y hace clic en el botón "actualizar carrito", la página se actualiza mostrando que el artículo ya no está en el carrito.

Cuando el usuario hace clic en las casillas de verificación de eliminación junto a cada artículo del carrito y hace clic en el botón "actualizar carrito", la página se actualiza mostrando un carrito vacío. (Esto generará preguntas: ¿debería dirigirse al usuario a otra página? ¿Debería aparecer el botón "seguir comprando"?)

Cuando el usuario hace clic en el botón "actualizar carrito" sin marcar un artículo para eliminarlo, la página se actualiza y no se elimina nada del carrito.

Pida a sus clientes que escriban ejemplos y casos de prueba de alto nivel antes la iteración. Esto puede ayudarles a pensar más en las historias y ayudar a definir sus condiciones de satisfacción. También les ayuda a identificar qué características son críticas y cuáles podrían esperar. También ayuda a definir cuándo La historia está hecha y gestionar las expectativas entre el equipo.

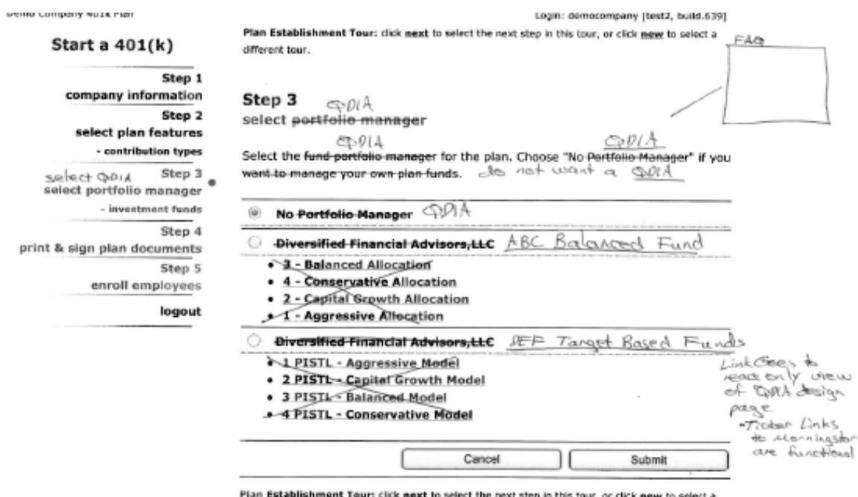


Figura 16-4 Ejemplo de maqueta de cliente

La Figura 16-4 muestra una maqueta de muestra, donde el propietario del producto marcó cambios en la página existente. Tenga cuidado al utilizar una captura de pantalla existente de un sistema antiguo, porque correrás el riesgo de tener un sistema nuevo que exactamente como el anterior, incluso si eso no es lo que querías.

Las maquetas son esenciales para historias que involucran la interfaz de usuario o un informe. Pida a sus clientes que expongan sus ideas sobre cómo debería verse la página. Comparte estos ideas con el equipo. Una idea es escanearlos y subirlos a la wiki, para que todos tengan acceso. Úselos como punto de partida y haga más prototipos en papel o dibújelos en la pizarra. Estos pueden ser fotografiados y subido para que lo vean los miembros remotos del equipo.

ESTRATEGIAS DE PRUEBA

A medida que conozca las historias de la próxima iteración, piense en cómo abordarlas para probarlas. ¿Presentan algún desafío especial de automatización? Son:

¿Se necesitan nuevas herramientas?

La historia de Lisa

Recientemente, nuestra empresa necesitaba reemplazar el hardware de la unidad de respuesta de voz y el software de interfaz de voz interactiva. Un contratista debía proporcionar el software para operar la aplicación de voz, pero necesitaba interactuar mediante procedimientos almacenados con la base de datos.

Esta fue una gran diferencia con respecto a cualquier software en el que habíamos trabajado antes, por lo que fue útil tener un día adicional para investigar cómo otros equipos han probado este tipo de aplicación antes de la planificación de la primera iteración que involucraba una historia relacionada con este proyecto. Durante la sesión de planificación de iteraciones, pudimos escribir tareas que eran pertinentes para las pruebas necesarias y dar mejores estimaciones.

—Lisa

Capítulo 9, "Kit de herramientas para empresas".
Enfrentando pruebas que Apoyen el Equipo", Capítulo 10, "De cara al negocio Pruebas que critican el producto".
y el Capítulo 11, "Criticar el Uso del producto Tecnología- Facing Tests", proporciona ejemplos de herramientas para diferentes tipos de pruebas.

Cuando su equipo se embarca en un nuevo tipo de software, puede decidir hacer una pico de desarrollo para ver qué puede aprender sobre cómo desarrollarlo. En el mismo tiempo, pruebe un pico de prueba para asegurarse de saber cómo impulsar el desarrollo con pruebas y cómo probar el software resultante. Si una novedad importante se acerca una característica o epopeya, escribe algunas tarjetas para investigarla y organiza reuniones de lluvia de ideas con una o dos iteraciones por adelantado. Eso te ayuda a saber qué Historias y tareas para planificar cuando empiezas a codificar. Una idea es tener un Equipo "explorador" que analiza qué soluciones técnicas podrían funcionar para próximas historias o temas.

PRIORIZAR DEFECTOS

En nuestro mundo ideal, queremos cero defectos al final de cada iteración y definitivamente al final del lanzamiento. Sin embargo, reconocemos que no vivimos en un mundo ideal. A veces tenemos que preocuparnos por defectos del sistema heredado y A veces, solucionar un defecto simplemente no tiene el valor suficiente para que la empresa lo solucione. ¿Qué pasa con estos defectos? Hablaremos de estrategias en el Capítulo 18, "Codificación y Pruebas", pero por ahora, consideremos simplemente que tenemos defectos que tratar con.

Antes de la próxima iteración es un momento ideal para revisar los problemas pendientes con al cliente y evaluar el valor de arreglarlo versus dejarlo en el sistema. Aquellos que se considere necesario arreglar deben programarse en el próxima iteración.

RECURSOS

Otra cosa que debes verificar antes de la iteración es si tu equipo tiene todos los recursos que necesita para completar cualquier historia de alto riesgo. Necesitas algunos expertos que se comparten con otros proyectos? Por ejemplo, es posible que necesite un experto en seguridad si una de las historias presenta un riesgo de seguridad o es por una característica de seguridad. Si se van a realizar pruebas de carga, es posible que necesite tener una herramienta especial o tener ayuda de un especialista en pruebas de carga de otro equipo, o incluso de un proveedor que Proporciona servicios de pruebas de carga. Esta es su última oportunidad de planificar el futuro.

RESUMEN

Es posible que su equipo necesite o no realizar alguna preparación antes de una iteración. Debido a que las prioridades cambian rápidamente en el desarrollo ágil, no querrás perder el tiempo planificando historias que pueden posponerse. Sin embargo, si estás a punto de implementar alguna tecnología nueva, embarcarse en un tema nuevo y complejo, esperar. Para ahorrar tiempo en la planificación de iteraciones, o si su equipo está dividido en diferentes ubicaciones, es posible que una planificación e investigación iniciales le resulten productivas. Como tester, puedes hacer lo siguiente:

Ayude a los clientes a lograr “claridad avanzada” (consenso sobre el comportamiento deseado de cada historia) haciendo preguntas y obteniendo ejemplos.

Sea proactivo, aprenda sobre historias complejas antes de la iteración y asegúrese de que tengan el tamaño correcto.

No siempre es necesaria una preparación previa para poder empezar a trabajar en la siguiente iteración. No haga ninguna preparación que no ahorre tiempo durante la iteración o garantice un mayor éxito en el cumplimiento de los requisitos del cliente.

Coordinar entre diferentes ubicaciones y facilitar la comunicación.

Hay muchas herramientas para ayudar con esto.

Obtenga ejemplos para ayudar a ilustrar cada historia.

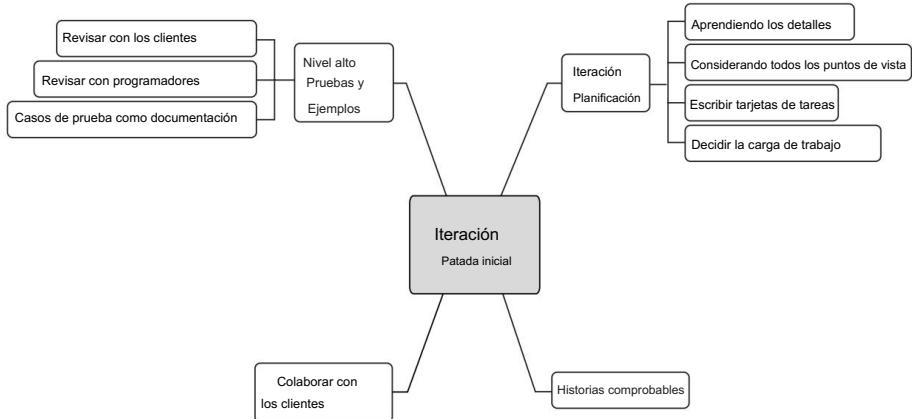
Desarrolle estrategias de prueba antes de la próxima iteración para características nuevas e inusuales.

Clasifique y priorice los defectos existentes para determinar si alguno debe programarse para la siguiente iteración.

Determine si es necesario alinear los recursos de prueba necesarios que no están disponibles actualmente para la próxima iteración.

Capítulo 17

INICIO DE LA ITERACIÓN



Los probadores ágiles desempeñan un papel esencial durante la planificación de iteraciones, ayudando a planificar las tareas de prueba y desarrollo. A medida que avanza la iteración, los evaluadores colaboran activamente con clientes y desarrolladores, escribiendo pruebas de alto nivel que ayudan a guiar el desarrollo, obteniendo e ilustrando ejemplos, asegurándose de que las historias sean comprobables. Echemos un vistazo más de cerca a las actividades del evaluador ágil al comienzo de cada iteración.

PLANIFICACIÓN DE ITERACIONES

La mayoría de los equipos inician su nueva iteración con una sesión de planificación. Esto podría ir precedido de una sesión retrospectiva o de “lecciones aprendidas”, para mirar hacia atrás y ver qué funcionó bien y qué no en la iteración anterior. Aunque los elementos de acción de la retrospectiva o las sugerencias de “iniciar, detener, continuar” afectarán la iteración que está a punto de comenzar, hablaremos de la retrospectiva como una actividad de final de iteración en el Capítulo 19, “Conclusión de la iteración”.

Mientras planifica el trabajo para la iteración, el equipo de desarrollo analiza una historia a la vez, escribiendo y estimando todas las tareas necesarias para implementar.



Figura 17-1 Reunión de planificación de iteraciones

Esa historia. Si ha trabajado un poco con antelación para prepararse para la iteración, Es probable que esta sesión de planificación transcurra con bastante rapidez.

Los equipos nuevos en el desarrollo ágil a menudo necesitan mucho tiempo para su iteración. sesiones de planificación. La planificación de iteraciones a menudo tomaba un día entero cuando Lisa El equipo empezó por primera vez. Ahora están hechos en dos o tres horas, lo que incluye el tiempo para la retrospectiva. El equipo de Lisa usa un proyector para mostrar al usuario. casos de prueba de aceptación y condiciones de satisfacción de su wiki para que todos los miembros del equipo pueden verlos. También proyectan su historia online herramienta de tablero, donde escriben las tarjetas de tareas. Otro componente tradicional de sus reuniones de planificación es un plato de delicias que se turnan para servir. La Figura 17-1 muestra una reunión de planificación de iteraciones en curso.

Aprendiendo los detalles

Lo ideal es que participen el propietario del producto y/u otros miembros del equipo del cliente. en la planificación de la iteración, respondiendo preguntas y proporcionando ejemplos que describen los requisitos de cada historia. Si nadie del lado comercial puede asistir, los miembros del equipo que trabajan en estrecha colaboración con los clientes, como los analistas. y evaluadores, pueden actuar como apoderados. Explican detalles y toman decisiones sobre en nombre de los clientes, o tome nota de las preguntas para obtener respuestas rápidamente. Si

su equipo repasó las historias con los clientes antes de la iteración, Puede pensar que no los necesita a mano durante la planificación de la iteración. sesión. Sin embargo, sugerimos que estén disponibles en caso de que tenga preguntas adicionales.

Como hemos enfatizado a lo largo del libro, use ejemplos para ayudar al equipo comprenda cada historia y convierta estos ejemplos en pruebas que impulsen la codificación. Aborde las historias en orden de prioridad. Si no has repasado historias anteriormente Junto con los clientes, el propietario del producto u otra persona que represente al equipo del cliente lee primero cada historia que se va a planificar. Explican el propósito de la historia, el valor que aportará y díjieron ejemplos de cómo se utilizará. Esto podría implicar pasar ejemplos o escribir en una pizarra. interfaz de usuario y las historias de los informes pueden tener ya marcos de alambre o maquetas que el equipo puede estudiar.

Una práctica que ayuda a algunos equipos es escribir pruebas de aceptación de usuario para cada uno. historia juntos, durante la planificación de la iteración. Junto con el propietario del producto y posiblemente otras partes interesadas, escriben pruebas de alto nivel que, una vez aprobadas, mostrarán que la historia está terminada. Esto también podría hacerse poco antes de la planificación de la iteración como parte del "trabajo de preparación" de la iteración.

Las historias deben tener un tamaño que no demore más de unos días en completarse. Cuando recibimos pequeñas historias para probar de forma regular, no las tenemos todas. terminado de una vez y apilado al final de la iteración esperando ser probado. Si una historia ha superado la planificación del lanzamiento y las discusiones previas a la iteración y aún es demasiado grande, esta es la última oportunidad para dividirla en partes más pequeñas. piezas. Incluso una historia pequeña puede resultar compleja. El equipo puede realizar un ejercicio para identificar las porciones finas o la ruta crítica a través de la funcionalidad. Usar Ejemplos para guiarlo y encontrar los escenarios de usuario más básicos.

Los evaluadores ágiles, junto con otros miembros del equipo, están alerta ante el "desplazamiento del alcance". No Tenga miedo de levantar una bandera roja cuando una historia parece estar creciendo en todas direcciones. El equipo de Lisa hace un esfuerzo consciente para señalar lo que es "bling" o "es bueno tenerlo". componentes, que no son centrales para la funcionalidad de la historia. Esos pueden ser posponer para el final, o posponerlo, en caso de que la historia tarde más de lo planeado en finalizar.

Considerando todos los puntos de vista

Como evaluador, intentarás poner cada historia en el contexto del sistema más amplio. y evaluar el potencial de impactos imprevistos en otras áreas. Como lo hiciste tú En la reunión de planificación del lanzamiento, póngase en las diferentes mentalidades de

usuario, parte interesada del negocio, programador, redactor técnico y todos los involucrados en la creación y el uso de la funcionalidad. Ahora estás trabajando a un nivel detallado.

En el capítulo de planificación de lanzamiento, utilizamos esta historia de ejemplo:

Como cliente, quiero saber cuánto costará el envío de mi pedido.

Según la velocidad de envío que elijo, puedo cambiarla si así lo deseo.

Decidimos tomar una “pequeña porción” y cambiar esta historia para asumir que solo hay Una velocidad de envío. Las otras velocidades de envío serán historias posteriores. Para esto En esta historia, necesitamos calcular el costo de envío según el peso del artículo y el destino, y decidimos usar la API de BigExpressShipping para el cálculo.

Nuestra historia ahora es como se muestra en la Figura 17-2.

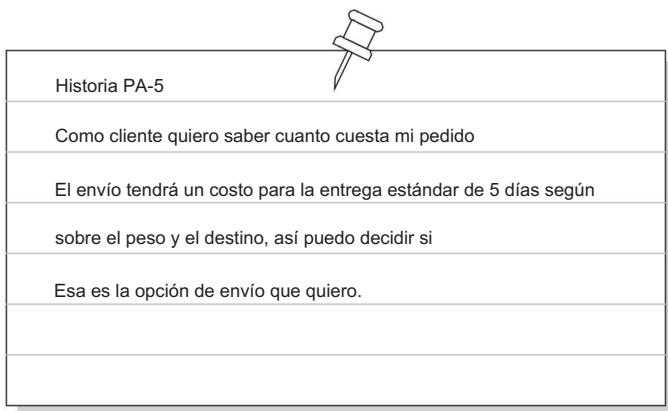


Figura 17-2 Velocidad de envío de Story para entrega en 5 días

El equipo comienza a discutir la historia.

Probador: “¿Esta historia se aplica a todos los artículos disponibles en el sitio? ¿Algunos artículos son demasiado pesados o están descalificados para la entrega en 5 días?

Propietario del producto: “Hay disponibilidad de terreno de 5 días para todos nuestros artículos. Son los de noche y de 2 días los que están restringidos a menos de 25 libras”.

Probador: “¿Cuál es el objetivo aquí, desde la perspectiva empresarial? ¿Facilitar el cálculo del coste para acelerar el pago? ¿Estás esperando

Anímelos a comprobar los otros métodos de envío: ¿son más rentables?

Propietario del producto: "La facilidad de uso es nuestro principal objetivo, queremos que el proceso de pago sea rápido y queremos que el usuario determine fácilmente el costo total del pedido para que no tenga miedo de completar la compra".

Programador: "Podríamos hacer que el costo de envío de 5 días se muestre de forma predeterminada tan pronto como el usuario ingrese la dirección de envío. Cuando hacemos las historias para las otras opciones de envío, podemos poner botones para que aparezcan esos costos rápidamente".

Propietario del producto: "Eso es lo que queremos: cubrir los costos por adelantado. Vamos a promocionar nuestro sitio como el más amigable para el cliente".

Probador: "¿Hay alguna manera de que el usuario pueda equivocarse? ¿Qué harán en esta página?

Propietario del producto: "Cuando agregamos las otras opciones de envío, pueden optar por cambiar su opción de envío. Pero por ahora, todo es muy sencillo. Ya tenemos validación para asegurarnos de que su código postal coincida con la ciudad que ingresan como dirección de envío".

Probador: "¿Qué pasa si se dan cuenta de que equivocaron su dirección de envío?

Quizás accidentalmente dieron la dirección de facturación. ¿Cómo pueden volver para cambiar la dirección de envío?

Programador: "Pondremos botones para editar direcciones de facturación y envío, por lo que será muy fácil para el usuario corregir errores. Mostraremos ambas direcciones en esta página donde se muestra el costo de envío. Podemos ampliar esto más adelante cuando agreguemos la opción de múltiples direcciones de envío".

Probador: "Eso haría que la interfaz de usuario fuera fácil de usar. Sé que cuando compro online me molesta no poder ver el coste de envío hasta la confirmación del pedido. Si el envío es carísimo y no quiero continuar, ya he perdido el tiempo. Queremos asegurarnos de que los usuarios no se queden atrapados en el proceso de pago, se frustren y simplemente se rindan. Entonces, la siguiente página que verán es la página de confirmación del pedido. ¿Existe alguna posibilidad de que el costo de envío sea diferente cuando el usuario accede a esa página?

Programador: "No, la API que nos da el costo estimado siempre debe coincidir con el costo real, siempre y cuando los mismos artículos todavía estén en el carrito de compras".

Propietario del producto: "Si cambian las cantidades o eliminan algún artículo, debemos asegurarnos de que el costo de envío cambie inmediatamente para reflejar eso".

Como puede ver en la conversación, salieron a la luz muchas aclaraciones. Ahora todos los miembros del equipo tienen una comprensión común de la historia. Es importante hablar de todos los aspectos de la historia.

Escribir pruebas de aceptación del usuario como

El grupo es una buena manera de asegurarse de que el equipo de desarrollo comprenda el requerimientos del cliente. Sigamos monitoreando esta conversación.

Probador: "Escribamos algunas pruebas rápidas para asegurarnos de que lo hacemos bien".

Cliente: "OK, ¿qué tal este ejemplo?

Puedo seleccionar dos artículos con una opción de envío de 5 días y ver mis costos inmediatamente.

Probador: "Buen comienzo, pero no sabremos a dónde enviarlo en ese momento.

¿Qué tal una prueba más genérica como:

Verifique que el costo de envío de 5 días se muestre como predeterminado tan pronto como el usuario ingresa una dirección de envío.

Cliente: "Eso funciona para mí".

Considerando todas las facetas

Paul Rogers relata una situación durante una reunión de planificación de iteración, donde surgió un problema de rendimiento en una historia que parecía sencilla y rápida.

Durante nuestra reunión de iteración, una de las historias que estábamos discutiendo fue cómo agregar algunas imágenes nuevas a parte de una aplicación web. Esta discusión siguió.

Propietario del producto: "También me gustaría aparecer en la historia para ver imágenes adicionales".

Desarrollador 1: "Está bien, ¿quién tiene idea de cuánto tiempo llevará?"

Desarrollador 2: "Es bastante rápido, tal vez medio día".

Desarrollador 3: "Pero qué pasa con los cambios en la base de datos?"

Desarrollador 2: "Los incluí en el presupuesto".

Desarrollador 1: "Está bien, vamos con medio día".

Yo: "Espera. Analizamos algunos problemas de rendimiento en la última iteración.

Si agregamos todas esas imágenes, estaremos sufriendo un impacto en el rendimiento.

Desarrollador 1: "Está bien, deberíamos pensar en eso un poco más. Quizás haya otras formas de implementarlo.

Desarrollador 2: "¿Por qué no hacemos un aumento rápido, agregamos las imágenes simuladas y ejecutamos otra prueba de rendimiento?"

Fue realmente bueno que esta discusión incluso antes de comenzar una historia nos diera algunas ideas de los problemas que podemos encontrar.

Cualquiera que no esté seguro del impacto de una historia en el resto del sistema o de la dificultad de desarrollar la funcionalidad puede y debe plantear un problema durante la planificación de la iteración. Es mejor abordar la incertidumbre desde el principio y luego investigar más o aumentar para obtener más información.

Hacer preguntas basadas en diferentes puntos de vista ayudará a aclarar la historia. y permitir que el equipo haga un mejor trabajo.

Escribir tarjetas de tareas

Cuando su equipo comprenda bien una historia, podrá empezar a escribir y tarjetas de tareas de estimación. Porque el desarrollo ágil impulsa la codificación con pruebas, escribimos tarjetas de tareas de prueba y desarrollo al mismo tiempo.

Si ha realizado alguna planificación previa, es posible que ya tenga algunas tarjetas de tareas. escrito y publicado. Si no, esríbalos durante la reunión de planificación de iteraciones. Él No importa quién escriba las tarjetas de tareas, pero todos en el equipo deberían revíselas y tenga la oportunidad de dar su opinión. Reconocemos que las tareas pueden Se pueden agregar tareas a medida que comenzamos a codificar, pero reconocer la mayoría de las tareas y estimarlas durante la reunión le da al equipo una buena idea de lo que implica.

La historia de Lisa

Cuando nuestro equipo está listo para comenzar a escribir tarjetas de tareas, los programadores generalmente crean tarjetas de tareas de codificación. Los evaluadores escriben tarjetas de tareas de prueba al mismo tiempo.

Normalmente empiezo con una tarjeta para escribir casos de prueba de alto nivel. Pregunto a los programadores si la historia se puede probar detrás de la GUI y escribo tarjetas de tareas de prueba en consecuencia. Esto generalmente significa una tarjeta de prueba para "Escribir casos de prueba de FitNesse" y una tarjeta de tarea de desarrollador para "Escribir dispositivos de FitNesse", a menos que los dispositivos ya existan. A veces, todas las pruebas detrás de la GUI se pueden cubrir más fácilmente en pruebas unitarias, por lo que siempre es bueno preguntar si este es el caso.

Ponemos todo lo que el equipo necesita recordar durante la iteración en una tarjeta de tarea. "Mostrar la interfaz de usuario a Anne" o "Enviar archivos de prueba a Joe" aparecen en el guión gráfico junto con todas las demás tareas.

Calculo las tarjetas de tareas de prueba a medida que avanza y le pido al equipo comentarios sobre las tarjetas y las estimaciones. A veces nos dividimos en grupos y cada grupo toma algunas historias y les escribe tarjetas de tareas. Siempre revisamos todas las tarjetas juntas, junto con el tiempo estimado. Si el tiempo de desarrollo es relativamente bajo en comparación con el tiempo de prueba, o viceversa, eso provoca una discusión. Llegamos a un consenso sobre si creemos que se han cubierto todos los aspectos de la historia con tarjetas de tareas. Si todavía hay algunas incógnitas, simplemente posponemos la escritura de las tarjetas de tareas hasta que tengamos la información.

Todas las tarjetas de prueba y desarrollo van en el guión gráfico en la columna de "cosas por hacer". Cualquier miembro del equipo puede registrarse para obtener cualquier tarjeta. Algunas tarjetas de tareas de prueba pasan a la columna "trabajo en progreso" o "hecho" antes de que las tarjetas de codificación comiencen a moverse, de modo que los programadores tengan algunas pruebas para guiar su codificación. A medida que las tarjetas de tareas de codificación se mueven a la columna "listo", las tarjetas para probar la funcionalidad "listo" se mueven a "trabajo en progreso".

—Lisa

Janet utiliza un método similar a este, pero la tarjeta de codificación del programador permanece en la columna "Para probar" hasta que se haya completado la tarea de prueba. Ambos las tarjetas se mueven al mismo tiempo a la columna "Listo".

Tres tarjetas de prueba para Story PA-5 (Figura 17-2), que muestran el costo de envío de 5-días entrega en día según el peso y el destino, que el equipo de Lisa podría escribir son:

Escriba pruebas de FitNesse para calcular el costo del envío de 5 días según el peso y el destino.

Escriba pruebas WebTest para mostrar el costo de envío de 5 días.

Pruebe manualmente mostrando el costo de envío de entrega en 5 días.

Algunos equipos prefieren escribir tareas de prueba directamente en la tarea de desarrollo. tarjetas. Es una solución simple, porque la tarea obviamente no está "terminada" hasta que la prueba ha terminado. Estás intentando evitar un enfoque de "minicascada" en el que las pruebas se realizan al final y el programador siente que ha terminado porque "envió la historia a control de calidad". Vea qué enfoque funciona mejor para su equipo.

Si la historia involucra en gran medida a partes externas o recursos compartidos, escriba la tarea tarjetas para asegurarse de que esas tareas no se olviden y haga que las estimaciones sean lo suficientemente generosas como para permitir dependencias y eventos más allá del control del equipo. Nuestro equipo hipotético que trabaja en la historia de los costos de envío tiene que trabajar con la API de cálculo de costos del transportista.

Probador: "¿Alguien sabe con quién trabajamos en BigExpressShipping para obtener especificaciones sobre su API? ¿Qué les pasamos, sólo el peso y el código postal? ¿Ya tenemos acceso para probar esto?"

Scrum Master: "Joe de BigExpressShipping es nuestro contacto y ya envió este documento especificando el formato de entrada y salida. Todavía necesitan autorizar el acceso desde nuestro sistema de prueba, pero eso debería hacerse en un par de días".

Probador: "Oh, bien, necesitamos esa información para escribir casos de prueba. Escribiremos una tarjeta de prueba solo para verificar que podemos acceder a su API y recuperar el costo de envío. Pero, ¿cómo sabemos que el coste es realmente correcto?"

Scrum Master: "Joe nos ha proporcionado algunos casos de prueba para el peso, el código postal y el costo esperado, para que podamos enviar esas entradas y verificar el resultado correcto. También tenemos esta hoja de cálculo que muestra las tarifas para algunos códigos postales diferentes".

Probador: "Deberíamos dedicar mucho tiempo para asegurarnos de que accedemos a su API correctamente. Voy a poner una estimación alta en esta tarjeta para verificarla usando la API para realizar pruebas. Quizás la tarjeta de desarrollador para la interfaz de la API también debería tener una estimación bastante conservadora".

Al escribir tarjetas de tareas del programador, asegúrese de que las estimaciones de las tareas de codificación incluya tiempo para escribir pruebas unitarias y para todas las pruebas necesarias por parte de los programadores. Una tarjeta para pruebas "de un extremo a otro" ayuda a garantizar que los programadores Al trabajar en tareas diferentes e independientes, verifique que todas las piezas funcionen juntas. Los evaluadores deben ayudar a asegurarse de que todas las tarjetas necesarias estén escritas y que Tienen estimaciones razonables. No querrás realizar estimaciones de segunda mano, pero si las estimaciones de prueba son dos veces más altas que las estimaciones de codificación, podría ser vale la pena hablar de ello.

Algunos equipos siguen probando tareas durante un día de trabajo o menos y no se molestan en escriba las horas estimadas en la tarjeta. Si una tarjeta de tarea sigue disponible después de un día trabajo, el equipo habla sobre por qué sucedió eso y escribe nuevas tarjetas para ir adelante. Esto podría reducir los gastos generales y el mantenimiento de registros, pero si está ingresando tareas en su sistema electrónico, es posible que no sea así. Haz lo que hace sentido para su equipo.

Calcular el tiempo necesario para corregir errores también siempre es complicado. Si los defectos existentes son presentado como historias, es bastante simple. Pero ¿qué pasa con los errores que hay? encontrado como parte de la iteración?

La historia de Janet

Con los nuevos equipos ágiles, descubrí que siempre parecen terminar dedicando tiempo a errores que no estaban asignados como parte de sus estimaciones para las historias. Con el tiempo, los programadores aprenden cuánto tiempo suelen dedicar a corregir errores de una historia y pueden simplemente agregar medio día o un par de horas a sus tareas para ese propósito. Volver a probar las correcciones de errores también agrega tiempo a las estimaciones de los evaluadores.

Hasta que los miembros del equipo se familiaricen con esto, puede ser apropiado realizar un seguimiento del tiempo dedicado a corregir y probar errores por separado. Mi equipo actual agrega una historia en XPlanner con tareas para corregir y probar aquellos errores que no se detectaron de inmediato. Estamos rastreando el tiempo para poder estimar mejor en el futuro.

—Janet

Sin embargo, su equipo elige estimar el tiempo dedicado a reparar defectos durante La iteración, ya sea que se incluya en la estimación de la historia o se realice un seguimiento por separado, asegúrese de que se realice de manera consistente.

Otro elemento a considerar al estimar las tareas de prueba son los datos de prueba. El comienzo de una iteración es casi demasiado tarde para pensar en los datos que necesita prueba con. Como mencionamos en el Capítulo 15, "Actividades del probador en versión o Planificación del tema", piense en los datos de prueba durante la planificación del lanzamiento y pregunte a los clientes para ayudar a identificarlo y obtenerlo. Ciertamente piénsalo mientras te preparas para la siguiente iteración. Cuando comienza la iteración, faltan todos los datos de prueba. Debe crearse u obtenerse, así que no olvide incluirlo en las estimaciones.

La historia de Lisa

Se trabajó un tema relacionado con los estados de cuenta trimestrales de los participes de planes de retiro. Estábamos modificando un trabajo mensual que toma una "instantánea" de la cuenta de cada participante en la fecha especificada. La instantánea se basa en una enorme cantidad de datos en la base de datos de producción, incluidas miles de transacciones diarias. Planeamos con anticipación.

Para la primera iteración, hicimos algunas historias sobre el tema sabiendo que solo podíamos probar algunos casos usando algunos planes de jubilación individuales para los cuales teníamos datos en la base de datos de prueba. También sabíamos que necesitábamos una prueba a mayor escala, con todos los planes de jubilación en la base de datos y al menos un mes completo de datos. Escribimos una tarjeta de tareas para copiar suficientes datos de producción para producir una "instantánea" mensual y nos aseguramos de que los datos se borraran para proteger la privacidad.

Luego planificamos la prueba completa en la siguiente iteración. Estos datos permitieron a los evaluadores encontrar problemas que antes no eran detectables cuando solo había datos parciales disponibles. Fue un buen equilibrio entre datos "suficientes" para realizar la mayor parte de la codificación y la cantidad total disponible a tiempo para verificar la funcionalidad completa. Debido a que el equipo planeó con anticipación, los errores se solucionaron a tiempo para la versión crítica.

—Lisa

Decidir la carga de trabajo

Nosotros, como equipo técnico, controlamos nuestra propia carga de trabajo. A medida que escribimos tareas para cada historia y las publicamos en nuestro storyboard (real o virtual), sumamos las horas estimadas o comprobamos visualmente la cantidad de tarjetas. ¿Cuánto trabajo podemos asumir? En XP, no podemos exceder la cantidad de puntos de historia que completamos en la última iteración. En Scrum, nos comprometemos con un conjunto de historias basadas en el tiempo real que creemos que necesitamos para completarlas.

El equipo actual de Lisa tiene varios años de experiencia en su proceso ágil y descubre que a veces pierden tiempo escribiendo tarjetas de tareas para historias que tal vez no tengan tiempo de hacer durante la iteración. Comienzan con suficientes historias para mantener a todos ocupados. A medida que las personas comienzan a liberarse, obtienen más historias y planifican las tareas relacionadas. Es posible que tengan algunas historias listas "en cubierta" para traerlas tan pronto como terminen las iniciales. Esto suena fácil, pero es difícil de hacer hasta que haya aprendido lo suficiente como para tener más confianza en el tamaño de las historias y la velocidad del equipo, y saber qué puede y qué no puede hacer su equipo en un período de tiempo determinado y en circunstancias específicas.

Su trabajo como evaluador es asegurarse de que se asigne suficiente tiempo a las pruebas y recordarle al equipo que las pruebas y la calidad son responsabilidad de todo el equipo. Cuando el equipo decide cuántas historias pueden entregar en la iteración, la pregunta no es "¿Cuánta codificación podemos terminar?" sino "¿Cuánta codificación y pruebas podemos completar?" Habrá ocasiones en las que una historia será fácil de codificar, pero las pruebas llevarán mucho tiempo. Como evaluador, es importante que solo acepte en la iteración tantas historias como se puedan probar.

Si tienes que comprometerte, hazlo de forma conservadora. Siempre es mejor incluir otra historia que tener que abandonar una. Si tiene historias de alto riesgo que son difíciles de estimar, o algunas tareas son desconocidas o necesitan más investigación, escriba tarjetas de tareas para una historia adicional o dos y téngalas listas para incorporarlas a mitad de la iteración.

Como equipo, siempre haremos nuestro mejor esfuerzo. Necesitamos recordar que ninguna historia está terminada hasta que se prueba, así que planifíquela en consecuencia.

HISTORIAS COMPROBABLES

Cuando estés viendo historias y los programadores empiecen a pensar en la implementación, piensa siempre en cómo puedes probarlas. Un ejemplo contribuye en gran medida a "probar la capacidad de prueba". ¿Qué impacto tendrá en mis pruebas? La Parte III, "Los cuadrantes de pruebas ágiles", ofrece muchos ejemplos de cómo

diseñar la aplicación para permitir pruebas efectivas. Esta es su última oportunidad para pensar en la capacidad de prueba de una historia antes de comenzar la codificación.

La historia de Janet

Un equipo con el que trabajé me habló de los problemas que tuvieron en la versión anterior.

El equipo estaba reescribiendo el primer paso de un proceso de varios pasos. Lo que no anticiparon fue que cuando comenzó el desarrollo del nuevo paso, el resto del proceso se interrumpió. No se pudieron realizar pruebas sobre ningún otro cambio en esa iteración hasta que se haya completado todo el primer paso.

La capacidad de prueba no se había considerado al planificar la historia. En la siguiente versión, cuando decidieron reescribir el segundo paso, aprendieron de su error anterior. Los programadores crearon un botón adicional en la página que permitió a los evaluadores llamar a la página nueva (en proceso) o a la página anterior para permitirles probar otras historias.

Recuerde preguntar: "¿Cómo podemos probar esto?" si no es obvio para ti.

—Janet

Durante la planificación de la iteración, piense qué tipo de variaciones necesitará Probar. Eso puede generar otras preguntas.

La historia de Janet

Durante una reunión de planificación de iteraciones en la que estuve, los programadores comenzaron a hablar sobre la implementación y a hacer dibujos en la pizarra para mostrar lo que estaban pensando.

Lo pensé un momento y me pregunté: "¿Se puede hacer de forma más sencilla?

Las permutaciones y combinaciones para probar la implementación propuesta harán que las pruebas sean horribles".

Los programadores lo pensaron durante un par de minutos y sugirieron una alternativa que no sólo satisfacía las necesidades del cliente, sino que era más sencilla y fácil de probar.

Fue una combinación beneficiosa para todos.

—Janet

Cuando la capacidad de prueba sea un problema, conviértalo en un problema que debe resolver el equipo. Equipos que comienzan su planificación escribiendo tarjetas de tareas de prueba probablemente tenga una ventaja aquí, porque mientras piensan en sus tareas de prueba, preguntarán cómo se desarrolló la historia. se puede probar. ¿Se puede probar alguna funcionalidad detrás de la GUI? Es posible hacer las pruebas de cara al negocio a nivel de unidad? Todo equipo ágil debería ser prueba de pensamiento primero. Mientras su equipo escribe tarjetas de tareas de desarrollador para una historia, piense sobre cómo probar la historia y cómo automatizar las pruebas. Si los programadores aún no tienen el hábito de codificar TDD o automatizar pruebas unitarias, intente

escribir una tarjeta de tareas “XUnit” para cada historia. Escribir tarjetas de tareas de programación para cualquier accesorio de automatización de pruebas que sea necesario. Piensa en la aplicación cambios que podrían ayudar con las pruebas, como propiedades de tiempo de ejecución y API.

La historia de Lisa

La aplicación en la que trabajo tiene muchas actividades que dependen de la fecha y la hora. Los programadores agregaron una propiedad de servidor de ejecución a la aplicación web para establecer la fecha del servidor. Puedo especificar una anulación de fecha y hora y, cuando el servidor se inicia, se comporta en consecuencia. Esto permite iniciar procesos mensuales o trimestrales con una simple anulación. Esta propiedad ha ayudado a probar una amplia variedad de historias.

Markus Gärtner [2008] nos dijo que su equipo tiene una propiedad similar, “DATE_OFFSET”. contado en “días de avance”. Sin embargo, esto sólo fue utilizado por los componentes Java de la aplicación donde reside la lógica empresarial. Los sistemas back-end en C y C++ no utilizan el desplazamiento de fecha, lo que causó un problema.

—Lisa

Si tiene problemas similares porque otros equipos están desarrollando partes del sistema, escriba una tarjeta de tarea para discutir el problema con el otro equipo y venga.

Llegar a una solución coordinada. Si trabajar con el otro equipo no es una opción, reserve tiempo para pensar en otra solución. Como mínimo, tenga en cuenta las limitaciones, ajuste las estimaciones de las pruebas en consecuencia y administre el riesgo asociado.

La historia de Lisa

Iniciamos un proyecto para reemplazar el sistema de respuesta de voz interactiva (IVR) de nuestra empresa, que permite a los participantes del plan de jubilación obtener información de cuentas y administrar cuentas por teléfono. Contratamos a otra empresa para escribir el sistema en Java, con la intención de que nuestro equipo lo mantuviera después de un cierto período de tiempo.

Pasamos algún tiempo pensando en qué pruebas serían necesarias y cómo realizarlas. Presumiblemente, el contratista probaría cosas como la funcionalidad de conversión de texto a voz, pero tuvimos que proporcionar procedimientos almacenados para recuperar los datos apropiados de la base de datos.

Nuestro primer paso fue negociar con el contratista para entregar pequeñas porciones de características en forma iterativa, de modo que pudieran probarse a medida que avanzaba el proyecto y el trabajo se distribuyera uniformemente a lo largo de la vida del contrato. Decidimos probar los procedimientos almacenados utilizando dispositivos FitNesse y exploramos las opciones. Nos decidimos por PL/SQL para acceder a los procedimientos almacenados. A un programador se le encomendó la tarea de ponerse al día con PL/SQL para abordar la automatización de las pruebas.

El equipo apuntó a un enfoque paso a paso. Al asignar mucho tiempo a las tareas al principio, permitimos las pronunciadas curvas de aprendizaje involucradas.

Curiosamente, el contratista entregó una compilación inicial para la primera iteración, pero no pudo entregar los incrementos de código para las siguientes iteraciones. Terminamos cancelando el contrato y posponiendo el proyecto hasta que pudimos encontrar una solución mejor. Al obligar al contratista a trabajar en incrementos, descubrimos de inmediato que no podía cumplir. ¿Qué pasaría si les hubiéramos dejado tomar seis meses para redactar la solicitud completa? Probablemente no hubiera terminado bien. Aprovechamos lo que aprendimos para investigar un mejor enfoque.

—Lisa

Cuando se embarque en algo nuevo para el equipo, como un nuevo marco de plantillas o una biblioteca de informes, recuerde incluirlo como un riesgo en su plan de prueba. Con suerte, su equipo consideró la capacidad de prueba antes de elegir un nuevo marco o herramienta y seleccionó uno que mejoró su capacidad para prueba. Sea generoso con las estimaciones de sus tareas de prueba con todo lo nuevo, incluidos los nuevos dominios, porque hay muchas incógnitas. A veces nuevo El conocimiento del dominio o la nueva tecnología significa una curva de aprendizaje pronunciada.

COLABORA CON CLIENTES

Trabajar en estrecha colaboración con los clientes, o representantes de los clientes, como los analistas funcionales, es una de nuestras actividades más importantes como evaluadores ágiles. Mientras comienzas En la iteración, la colaboración con sus clientes también se acelerará. Este Es el momento de realizar todas esas buenas actividades descritas en el Capítulo 8, “Pruebas comerciales que respaldan al equipo”. Pida ejemplos a los clientes, pregunte preguntas abiertas sobre la funcionalidad y el comportamiento de cada historia, tienen discusiones en la pizarra y luego convertir esos ejemplos en pruebas para impulsar la codificación.

Incluso si el propietario de su producto y/u otros clientes explicaron las historias antes y durante la planificación de la iteración, a veces es útil repasarlas brevemente una vez más cuando comienza la iteración. Es posible que no todos lo hayan escuchado. antes, y el cliente podrá tener más información.

La historia de Lisa

Comenzamos a escribir pruebas de aceptación de alto nivel el primer día de la iteración. Debido a que repasamos todas las historias con el propietario del producto el día antes de la iteración y escribimos pruebas de aceptación del usuario como equipo para las historias más complejas, tenemos una idea bastante clara de lo que se necesita. Sin embargo, el acto de escribir más casos de prueba a menudo plantea nuevas preguntas. Repasamos las pruebas de alto nivel y cualquier pregunta que tengamos con el propietario del producto, quien también ha estado pensando más en las historias.

Un ejemplo de esto fue una historia que involucraba un archivo de distribuciones monetarias a los participantes del plan que retiraban dinero de sus cuentas de jubilación. Este archivo se envía a un socio que utiliza la información para emitir cheques al participante. Los montos en algunos de los registros no se conciliaban correctamente en el sistema del socio y el socio solicitó una nueva columna con un monto que les permitiera hacer una conciliación.

Después de la reunión de planificación de la iteración, a nuestro propietario de producto le preocupó que la nueva columna no fuera la solución adecuada y planteó sus dudas en la reunión de revisión de la historia. Él y un evaluador estudiaron el problema más a fondo y descubrieron que, en lugar de agregar una nueva cantidad, era necesario cambiar un cálculo. En realidad, esta era una historia más amplia, pero abordaba un problema central con las distribuciones. El equipo discutió la historia más amplia y escribió nuevas tarjetas de tareas. Valió la pena tomarse un poco de tiempo para discutir más a fondo la historia, porque la comprensión inicial resultó ser errónea.

—Lisa

Una buena comunicación suele requerir trabajo. Si no aprovecha suficientes oportunidades para hacer preguntas y revisar casos de prueba, programe reuniones periódicas para hacerlo. Si no hay mucho que discutir, las reuniones se irán.

rápidamente. El tiempo en una reunión para una discusión profunda puede ahorrar codificación y tiempo de prueba más tarde, porque estará más seguro de los requisitos.

PRUEBAS Y EJEMPLOS DE ALTO NIVEL

Queremos pruebas de "panorama general" para ayudar a los programadores a comenzar de la manera correcta. dirección en una historia. Como siempre, recomendamos comenzar con ejemplos y convirtiéndolos en pruebas. Tendrás que experimentar para ver cuántos detalles hay. apropiado en el nivel de prueba de aceptación antes de que comience la codificación. El equipo de Lisa tiene descubrió que las pruebas de alto nivel extraídas de ejemplos son lo que necesitan para comenzar fuera de una historia.

Las pruebas de alto nivel deben transmitir el propósito principal detrás de la historia. Que puede incluir ejemplos de conducta tanto deseada como no deseada. Para nuestro anterior Story PA-5 (Figura 17-2) que solicita mostrar el costo de envío para la entrega en 5 días según el peso y el destino del pedido, nuestras pruebas de alto nivel podrían incluir:

Verifique que el costo de envío de 5 días se muestre como predeterminado tan pronto como el usuario ingrese una dirección de envío.

Verifique que el costo de envío estimado coincida con el costo de envío en la factura final.

Verifique que el usuario pueda hacer clic en un botón para cambiar la dirección de envío y, cuando lo haga, se mostrará el costo de envío actualizado.

Verifique que si el usuario elimina artículos del carrito o agrega artículos al carrito, se muestra la opción de envío actualizada.

Consulte la bibliografía para obtener enlaces a más información sobre pruebas gráficas y desarrollo basado en modelos.

No te limites a las palabras de una página wiki cuando escribas textos de alto nivel. pruebas. Por ejemplo, una matriz de prueba como la que se muestra en la Figura 15-7 podría trabajar mejor. Algunas personas expresan las pruebas gráficamente, utilizando dibujos de flujo de trabajo y fotografías. Brian Marick [2007] tiene una técnica para dibujar pruebas gráficas que se puede convertir en scripts de prueba de Ruby. El desarrollo basado en modelos proporciona otra forma de expresar el alcance de alto nivel de una historia. Los casos de uso son otra posible vía para expresar el comportamiento deseado en el nivel del "panorama general".

Una imagen vale mas que mil palabras

El dicho "Una imagen dice más que mil palabras" también se puede aplicar a casos de prueba y validaciones de pruebas.

Paul Rogers [2008] ha estado experimentando con algunas ideas interesantes en torno a esto y explica el enfoque de su equipo ante este problema en el siguiente recuadro. La figura 17-3 muestra el modelo de interfaz de usuario que describe.

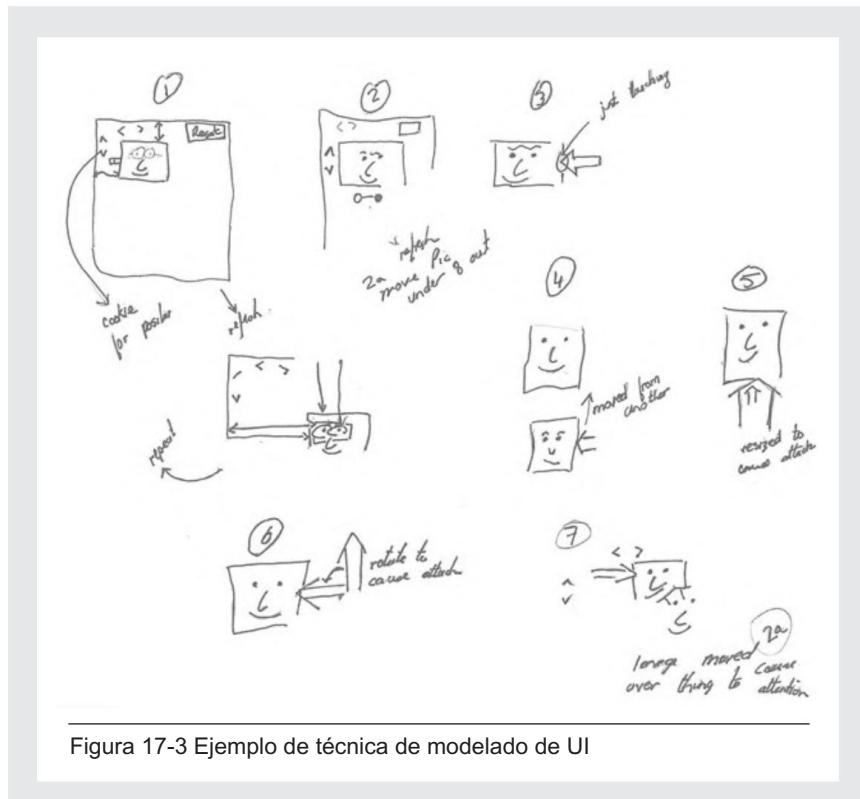
La aplicación en la que trabajo es de naturaleza muy gráfica. Permite al usuario modificar una página web agregando "mejoras fotográficas" como gafas, sombreros o bocadillos a las imágenes, o resaltando el texto en la página web con un efecto de rotulador.

Existe un complejo conjunto de reglas comerciales sobre qué adiciones se pueden aplicar a las imágenes, cómo y dónde se fijan y cómo se pueden rotar. Para explicar las pruebas de estas reglas, fue mucho más sencillo dibujar un boceto de una página web típica con los diferentes tipos de complementos y agregar pequeñas notas a cada imagen.

El resaltado de texto también planteó muchos desafíos. Las más problemáticas eran las áreas donde el resaltado de texto cubría sólo una parte de una etiqueta HTML. Para describir lo que se debería esperar en muchas situaciones diferentes, creamos diferentes páginas web y las imprimimos.

Usando marcadores de lápiz reales, resaltamos las áreas que esperábamos que se mostraran resaltadas después de comenzar y terminar en ciertas áreas. De esta manera, tuvimos una prueba de regresión fácil de leer.

Las herramientas de baja tecnología pueden eliminar el misterio del diseño de aplicaciones complejas. Encuentre formas de expresar las reglas comerciales de la manera más simple posible y compártalas con todo el equipo.



Vea el modelo de muestra de los cambios en la interfaz de usuario en el Capítulo 16, "Comience a trabajar" dibuje los cambios y publíquelos en la página wiki que describe el informe.

Consulte el Capítulo 9, "Kit de herramientas para empresas Pruebas que respaldan al equipo", para obtener algunas ideas sobre herramientas para recopilar y comunicar requisitos.

Las maquetas pueden transmitir requisitos para una interfaz de usuario o informar de forma rápida y clara. Si un informe existente necesita modificarse, tome una captura de pantalla del informe y utilícelo resaltadores, bolígrafo, lápiz o cualquier herramienta que tenga a mano. Si quieres capturar hacerlo electrónicamente, pruebe el programa Windows Paint u otra herramienta gráfica para dibujar los cambios y publíquelos en la página wiki que describe el informe requisitos.

Los equipos distribuidos necesitan pruebas de alto nivel disponibles electrónicamente, mientras que los equipos ubicados conjuntamente pueden funcionar bien a partir de dibujos en una pizarra, o incluso desde hacer que el cliente se siente con ellos y les indique los requisitos a medida que código.

Lo importante al comenzar la iteración es que aprenda rápidamente los requisitos básicos de cada historia y los exprese en contexto de una manera que Funciona para todo el equipo. Los equipos más ágiles con los que hemos hablado dicen que sus mayores

El problema es comprender cada historia lo suficientemente bien como para transmitir exactamente lo que dice. cliente buscado. Podrían producir código técnicamente libre de errores, pero no coincide del todo con la funcionalidad deseada por el cliente. O pueden terminar Reelaborar mucho una historia durante la iteración a medida que el cliente aclara los requisitos y, como resultado, quedarse sin tiempo para completar otra historia.



Consulte el Capítulo 8, "Presentación empresarial Pruebas que respaldan al equipo", para obtener más información sobre lo que constituye un requisito.

Dedique tiempo y esfuerzo a experimentar con diferentes formas de capturar y Exprese las pruebas de alto nivel de una manera que se ajuste a su dominio y entorno. A Janet le gusta decir que un requisito es una combinación de la historia + conversación + un escenario de usuario o una imagen de apoyo si es necesario + una prueba de entrenamiento o ejemplo.

Revisar con los clientes

Anteriormente en este capítulo hablamos sobre la importancia de una atención constante al cliente. colaboración. Revisar las pruebas de alto nivel con los clientes es una buena oportunidad para reforzar la colaboración y mejorar la comunicación, especialmente para un nuevo equipo ágil. Después de que su equipo tenga el hábito de hablar continuamente sobre historias, requisitos y casos de prueba, es posible que no necesite sentarse e ir sobre cada caso de prueba.

Si su equipo está contratado para desarrollar software, requisitos y casos de prueba podrían ser entregables formales que deba presentar. Incluso si no lo son, es una buena idea proporcionar los casos de prueba en un formato que los clientes puedan fácilmente leer por sí solos y comprender.

Revisando con programadores

Puedes tener todos los diagramas y páginas wiki del mundo, pero si nadie mirándolos, no ayudarán. La comunicación directa siempre es lo mejor. siéntate con los programadores y repasar las pruebas y requisitos de alto nivel. Ir sobre diagramas de pizarra o prototipos de papel juntos. La figura 17-4 muestra una Probador y programador discutiendo un diagrama de cortes finos o hilos a través un flujo de trabajo de usuario. Si estás trabajando con un miembro del equipo en otra ubicación, Encuentre una manera de programar una conversación telefónica. Si los miembros del equipo tienen problemas Al comprender las pruebas y los requisitos de alto nivel, sabrá que debe probar un enfoque diferente la próxima vez.

Los programadores con buen conocimiento del dominio pueden entender bien una historia de distancia y poder comenzar a codificar incluso antes de que se escriban las pruebas de alto nivel. Incluso Por lo tanto, siempre es una buena idea revisar las historias del cliente y del evaluador. perspectiva con los programadores. Su comprensión de la historia podría

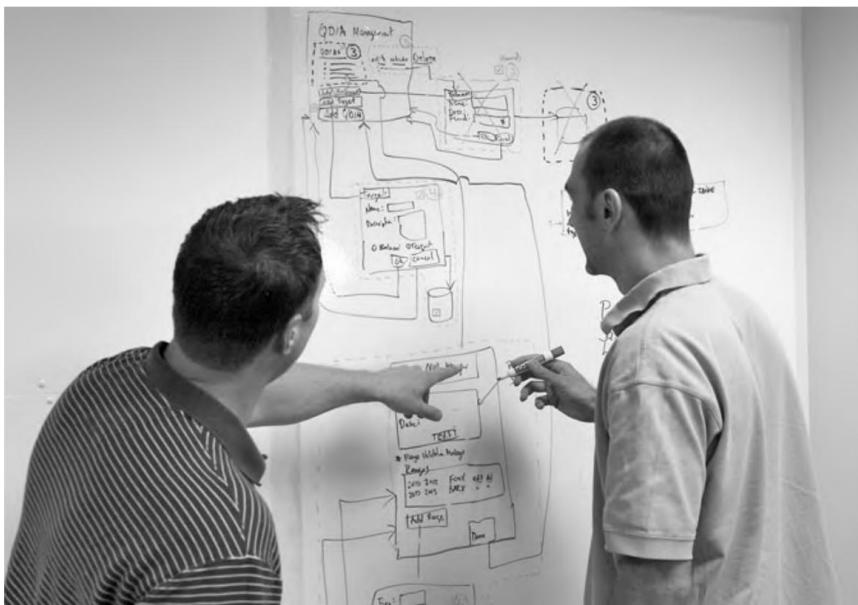


Figura 17-4 Una discusión en la pizarra

■

El Capítulo 2, "Diez principios para los probadores ágiles", presenta el "Poder regla de los Tres".

ser diferente al tuyo y es importante observar las discrepancias. Recordar la regla del "Poder de Tres" y captar un cliente si hay dos opiniones que usted No puedo reconciliarme. Los casos de prueba también ayudan a poner la historia en contexto con el resto. de la aplicación. Los programadores pueden utilizar las pruebas para ayudarles a codificar el historia correctamente. Esta es la razón principal por la que usted quiere hacer esto lo más cerca posible el inicio de la iteración como pueda, antes de que los programadores comiencen a codificar.

No olvides preguntar a los programadores qué creen que te has perdido.

¿Cuáles son las áreas de alto riesgo del código? ¿Dónde creen que las pruebas debería estar enfocado? Obtener una perspectiva más técnica ayudará a diseñar casos de prueba detallados. Si ha creado una matriz de prueba, es posible que desee revisar las áreas impactadas nuevamente también.

Un efecto secundario beneficioso de revisar las pruebas con los programadores es la aprendizaje cruzado que sucede. Usted, como evaluador, está expuesto a lo que son, pensando, y aprenden algunas técnicas para realizar pruebas que no usarían han encontrado de otra manera. Como programadores, pueden comprender mejor qué pruebas de alto nivel no habían considerado.

Casos de prueba como documentación

Casos de prueba de alto nivel, junto con las pruebas ejecutables que escribirá durante el iteración, formará el núcleo de la documentación de su aplicación. Los requisitos cambiarán durante y después de esta iteración, así que asegúrese de que sus casos de prueba ejecutables sean fáciles de mantener. Personas no familiarizadas con el desarrollo ágil

A menudo tenemos la idea errónea de que no existe documentación. De hecho, ágil

Los proyectos producen documentación utilizable que contiene pruebas ejecutables y por lo tanto está siempre actualizado.

La gran ventaja de tener pruebas ejecutables como parte de tus requerimientos documento es que es difícil discutir sus resultados.

La historia de Lisa

Con frecuencia, los propietarios de productos, administradores de planes o gerentes de desarrollo empresarial vienen y me hacen preguntas como: "¿Qué se supone que debe hacer el sistema si alguien presenta un pago de préstamo por cero dólares?" o "¿Por qué no todos los participantes en este plan recibieron una contribución no electiva del 3%?"

Mostrarles una prueba de FitNesse que reproduzca el escenario es mucho más poderoso que simplemente mostrarles requisitos narrativos. Quizás el sistema no fue diseñado como debería haber sido, pero la prueba ilustra cómo funciona realmente, porque podemos ver claramente los resultados de las entradas y operaciones. Esto ha ahorrado muchos argumentos del tipo "Pensé que funcionaba de esta manera".

Si deciden que la funcionalidad, tal como se implementó, es incorrecta, podemos cambiar los resultados esperados de la prueba y escribir una historia para implementar el código para que la prueba pase nuevamente con las nuevas expectativas. No puede hacer eso con un documento de requisitos.

—Lisa

Organizar los casos de prueba y las pruebas no siempre es sencillo. Muchos equipos documentar las pruebas y los requisitos en una wiki. La desventaja de la flexibilidad de un wiki es que puedes terminar con una mezcla de jerarquías. Es posible que usted tenga problemas para encontrar el requisito o ejemplo particular que necesita.

Capítulo 14, "Un Estrategia ágil de automatización de pruebas", tiene más en prueba gestión.

El equipo de Lisa revisa periódicamente su documentación wiki y sus pruebas de FitNesse, y refactoriza la forma en que están organizados. Si tiene problemas para organizar sus requisitos y casos de prueba, reserve algo de tiempo para investigar nuevas herramientas que puedan ayudar. Contratar a un redactor técnico capacitado es una buena manera de realizar su valiosa prueba. casos y ejemplos en un repositorio utilizable de información fácil de encontrar.

RESUMEN

La sesión de planificación de iteraciones marca la pauta para toda la iteración. En esto capítulo, analizamos lo que hacen los evaluadores ágiles para ayudar a iniciar la iteración a un buen comienzo.

Durante la planificación de la iteración, los evaluadores ayudan al equipo a conocer las historias haciendo preguntas y considerando todos los puntos de vista.

Las tarjetas de tareas deben escribirse junto con las tarjetas de tareas de desarrollo y estimarse de manera realista.

Otra forma de abordar las tareas de prueba es escribirlas directamente en las tarjetas de tareas del desarrollador.

Los equipos deben comprometerse con el trabajo para el cual puedan completar todas las tareas de prueba, porque ninguna historia está terminada hasta que esté completamente probada.

El inicio de una iteración es la última oportunidad para garantizar que las historias sean comprobables y que se proporcionen datos de prueba adecuados.

Los evaluadores colaboran con los clientes para explorar historias en detalle y escribir casos de prueba de alto nivel para permitir a los programadores comenzar a codificar.

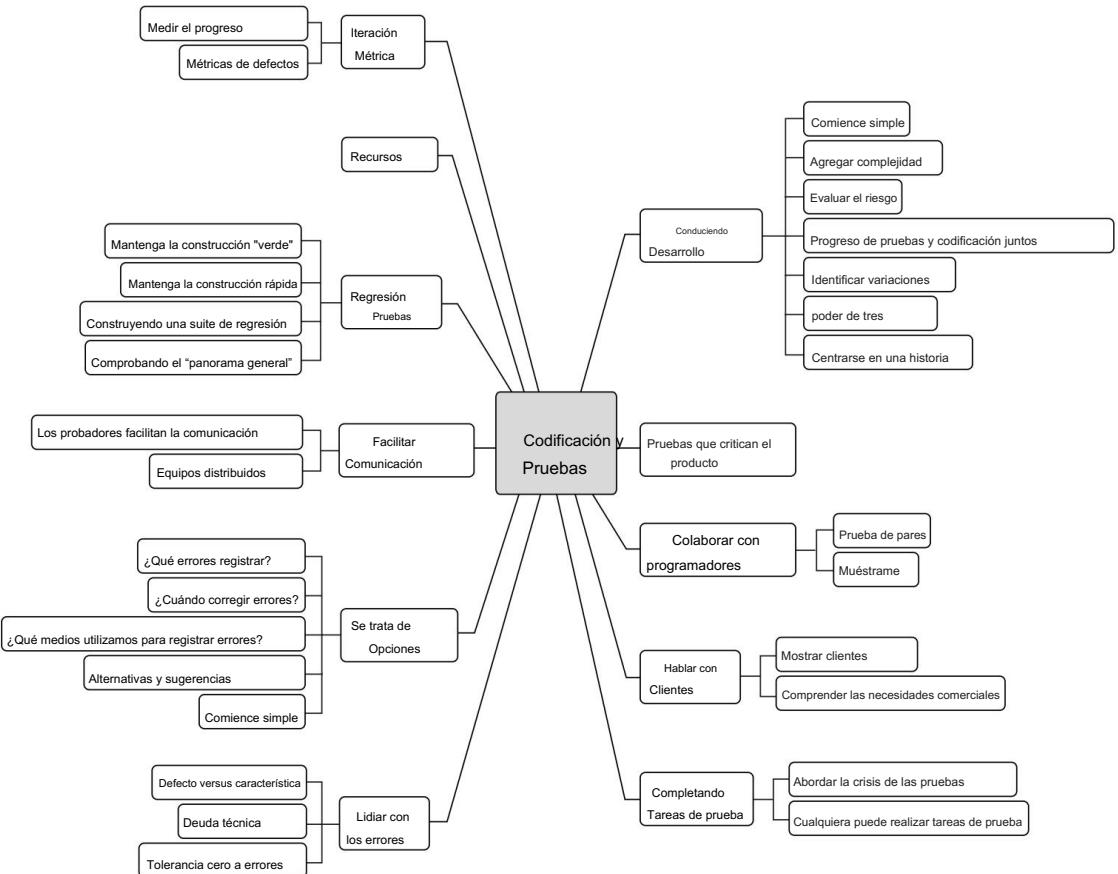
Los evaluadores revisan las pruebas y requisitos de alto nivel con los programadores para asegurarse de que se estén comunicando bien.

Las pruebas forman el núcleo de la documentación de la aplicación.

Esta página se dejó en blanco intencionalmente.

Capítulo 18

CODIFICACIÓN Y PRUEBAS



Nuestro ágil evaluador ha ayudado a planificar el lanzamiento, dimensionar las historias adecuadamente y asegurarse de que sean comprobables. Ella, junto con colegas del equipo de clientes y desarrollo, ha convertido ejemplos de comportamiento deseado para cada historia en pruebas de aceptación de usuarios de alto nivel. Ella y su equipo han reunido los recursos y la infraestructura necesarios para ofrecer valor empresarial. Ahora, los miembros del equipo tomaron tarjetas de tareas y comenzaron a escribir código. ¿Qué hacen los evaluadores a continuación, especialmente antes de que las historias estén listas para ser probadas?

IMPULSANDO EL DESARROLLO

El comienzo de la codificación es un buen momento para empezar a escribir pruebas detalladas. El pruebas de alto nivel escritas antes de la iteración, o en los primeros días de la misma, Proporciona suficiente información para que los programadores comiencen su propio desarrollo basado en pruebas. Ahora tenemos un poco de margen para respirar, pero si no lo hacemos Si se mueve rápidamente, la codificación podría adelantarse a las pruebas y salir mal dirección.

Ahora es el momento de empezar a escribir pruebas ejecutables que ilustren los detalles. sobre una historia para que el desarrollo avance sin problemas y Ayude a que las pruebas sigan el ritmo de la codificación. Al igual que las pruebas de alto nivel, nos basamos en detalles pruebas sobre ejemplos proporcionados por los clientes.

En este punto, principalmente estamos escribiendo pruebas que serán automatizadas, pero también estamos Pensando en las importantes pruebas exploratorias que debemos realizar como codificación. esta completado.

Comience simple

Como evaluadores, nos distraemos fácilmente con olores de código interesantes y casos extremos. Sin embargo, si utilizamos pruebas para guiar la codificación, debemos comenzar con lo básico. Escribe la prueba del camino feliz más simple que puedas para demostrar que el núcleo La funcionalidad funciona.

Capítulo 14, "Un Agile Automation Strategy", ofrece sugerencias para seleccionar las herramientas adecuadas.

¿Por qué pruebas ejecutables? Estamos trabajando en un calendario extremadamente apretado y Ni los programadores ni los probadores tienen tiempo para detenerse y ejecutar el manual. pruebas una y otra vez. Tienen tiempo para hacer clic en un botón y ejecutar una prueba automatizada. Esta prueba debe fallar de una manera que haga que la causa sea tan obvia como sea posible. Lo ideal sería darles estas pruebas a los programadores para que podría ejecutarlos mientras codifican. Ésa es una de las razones por las que elegir el producto adecuado El marco de automatización es muy importante.

Para algunas historias, la automatización de las pruebas puede llevar mucho tiempo. Al mantener el La primera prueba es sencilla: usted se concentra en diseñar la solución de automatización. Cuando la prueba simple funciona, vale la pena dedicar tiempo a pruebas más complejas. casos.

Destacamos la importancia de la automatización, pero Janet ha trabajado con equipos que han utilizado con éxito pruebas manuales en forma de listas de verificación u hojas de cálculo para brindar a los programadores la información que necesitan para comenzar. Sin embargo, Para tener éxito a largo plazo, estas pruebas deben automatizarse.

Agregue complejidad

Tan pronto como la prueba del camino feliz funcione, comience a agregar más casos de prueba. Agregue condiciones de límites y bordes. Las pruebas pueden mostrar que los programadores entendieron mal un requisito, o pueden mostrar que un evaluador lo hizo, o tal vez el verdadero significado del requisito evadió a todos. Lo importante es que todos hablen de ello y se pongan manos a la obra.

A medida que los evaluadores piensan en nuevos escenarios para validar con pruebas ejecutables, también piensan en escenarios potenciales para pruebas exploratorias manuales. Tome nota de estos para su posterior seguimiento.

Recuerde el propósito de estas pruebas. Deberían proporcionar ejemplos que indiquen a los programadores qué código escribir. A medida que el código evoluciona, sus pruebas pueden desafiarlo más, pero resista la tentación de seguir inmediatamente los olores hasta los casos extremos. Primero, haga funcionar lo básico. Si piensa en más casos basándose en algún análisis de riesgo, siempre puede agregar pruebas adicionales más adelante.

Evaluar el riesgo

Los evaluadores han utilizado el análisis de riesgos para ayudar a priorizar las pruebas durante mucho tiempo, y la consideración del riesgo ya está integrada en el desarrollo ágil. Las historias de alto riesgo pueden obtener estimaciones de mayor tamaño, y los equipos consideran el riesgo al priorizar las historias durante la planificación del lanzamiento y la iteración.

Un análisis rápido de riesgos puede ayudarle a decidir qué pruebas realizar primero y dónde centrar sus esfuerzos. Nunca tenemos tiempo para probarlo todo y podemos utilizar el análisis de riesgos para determinar cuántas pruebas son suficientes.

Si tiene una historia realmente compleja, es posible que desee comenzar enumerando todos los riesgos potenciales relacionados con la historia. Estos no se limitan a la funcionalidad. Considere la seguridad, el rendimiento, la usabilidad y otras "ilidades". A continuación, para cada elemento, califique el impacto en el negocio si ocurriera, usando una escala del 1 al 5 (o cualquier escala que funcione para usted): 1 es un impacto bajo, 5 es un impacto negativo crítico.

Ahora, considere la probabilidad de que ocurra cada ítem, usando la misma escala: 1 para nada probable que suceda y 5 para ítems que probablemente sucederán.

Multiplique las dos calificaciones para obtener la calificación de riesgo total para cada elemento. Esto facilita la selección de las áreas en las que su equipo debe centrar sus esfuerzos de prueba primero. Los elementos de bajo riesgo pueden dejarse para el final o, debido a que su impacto es bajo o es muy poco probable que ocurran, es posible que no se aborden en absoluto.

Tu dominio hace una gran diferencia aquí. Si está probando software que ejecuta En los marcapasos, probablemente necesites cubrir todos los riesgos con tus pruebas sin importa cuán bajos o improbables sean. Si estás probando una web interna de empresa aplicación para que la utilicen algunos expertos capacitados en la materia, es posible que capaz de omitir escenarios que son poco probables o que tienen una solución alternativa obvia.

Considera la historia de la Figura 18-1.

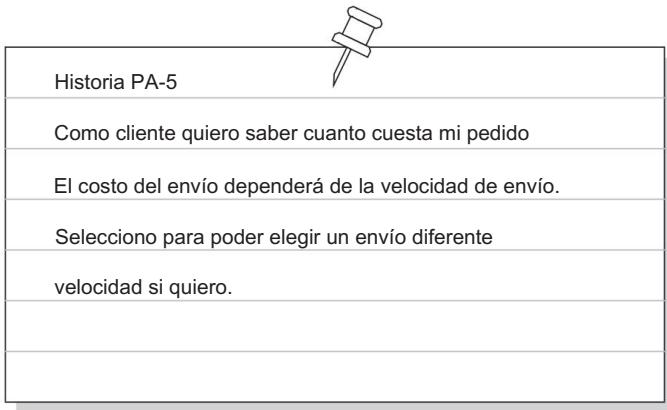


Figura 18-1 Historia sobre velocidades de envío

La Figura 18-2 muestra una posible evaluación de riesgos para esta historia de costos de envío.

# Artículo	Riesgo de probabilidad de impacto
1 Costo incorrecto mostrado	4 2 8
2 El usuario no puede elegir una opción de envío diferente	5 1 5
3 El artículo no es elegible para la opción de envío seleccionada, pero se permite la selección	3 26
4 El costo estimado no coincide con el costo real en verificar	3 4 12
5 Código postal no válido ingresado y no detectado por la validación	4 14
6 El usuario no puede entender las reglas de opciones de envío	2 3 6
7 El usuario no puede cambiar la dirección de envío	5 2 10
8 El usuario cambia la dirección de envío, pero el costo no cambiar en consecuencia	5 4 20

Figura 18-2 Ejemplo de evaluación de riesgos

El artículo 8 es el artículo de mayor riesgo, por lo que queremos asegurarnos de probar el cambio de direcciones de envío y verificar los costos actualizados. Es posible que deseemos automatizar un

prueba de un extremo a otro con este escenario. No estamos demasiado preocupados por el punto 5; tal vez ya hayamos probado la validación de nuestro código postal y nos sintamos bien así que no necesitamos probarlo más. Incluso puedes tener un artículo de muy bajo riesgo. que usted decidió no probar.

La historia suele ser una buena maestra. Tome nota de los problemas anteriores y asegúrese de que no vuelva a suceder.

Codificación y pruebas progresan juntas

En este punto de la iteración, la codificación y las pruebas continúan de la mano.

Probadores, programadores, expertos en bases de datos y otros miembros del equipo colaboran para desarrollar las historias, siguiendo las pautas proporcionadas por los ejemplos. y pruebas. Diferentes miembros del equipo pueden aportar su experiencia particular, pero todos se sienten responsables de que cada historia esté terminada. Todo ellos aprenden sobre la historia y aprenden unos de otros a medida que avanza el trabajo.

Veamos cómo podría trabajar un equipo en la historia de los costos de envío en la Figura 18-1.

Patty Programmer toma una tarjeta de tarea para codificar el costo de envío estimado cálculos. Ella ya entiende bastante bien la historia gracias a discusiones anteriores, pero puede mirar las páginas wiki o el reverso de la tarjeta de la historia donde los evaluadores escribieron una narrativa que describe el propósito de la historia, algunos ejemplos de cómo debería funcionar y algunas pruebas de alto nivel para hacer Seguro que tiene una buena idea de por dónde empezar. Tammy Tester ve esa codificación El trabajo ha comenzado y comienza a escribir casos de prueba detrás de la GUI por el costo. cálculos.

Durante la planificación, el equipo había acordado comenzar calculando el costo de envío de 5 días en función de la dirección de envío y el peso del artículo. Los artículos sólo pueden ser enviado dentro de América del Norte continental, pero esa validación se realizará en la capa de presentación, por lo que las pruebas de cálculo de costos solo pueden asumir valores válidos Los destinos se consideran para la entrada. Están utilizando una API de cálculo de costos. proporcionado por el socio de envío, y Tammy le pregunta a Patty dónde encontrar los algoritmos para que ella misma pueda calcular el costo y poder escribir las pruebas. tammy escribe el caso de prueba más simple que se le ocurre en su prueba detrás de la GUI herramienta. Lo mostramos como una tabla simple en la Figura 18-3.

Peso 5 libras	Código postal de destino 80104	Costo 7.25
------------------	-----------------------------------	---------------

Figura 18-3 Prueba de camino feliz simple

Patty aún no ha terminado el código que haría pasar esta prueba, así que Tammy comienza a trabajar en otra tarea de prueba para la historia, configurando el entorno de prueba para que funcione con el sistema de prueba del socio de envío.

Identificar variaciones

Debido a que esta historia y la prueba son tan sencillas, Patty y Tammy no discuta el diseño de la prueba y modifíquelo como lo harían en historias más complejas. Tampoco han necesitado hacerle más preguntas al propietario del producto todavía. Empanada llama a Tammy para mostrarle que la prueba simple ya está funcionando. tammy escribe más casos de prueba, probando diferentes pesos y destinos dentro los Estados Unidos. Todos funcionan bien. Intenta con un código postal canadiense y la prueba obtiene una excepción. Ella le muestra esto a Patty, quien se da cuenta de que la API El valor predeterminado son los códigos postales de EE. UU. y requiere un código de país para los códigos de Canadá y México. Todavía no había escrito ninguna prueba unitaria para otros países. Revisan las entradas de prueba y Patty se asocia con Paul Programmer para cambiar el código que llama a la API. Ahora la prueba se parece a la Figura 18-4.

Peso	Código postal de destino	Código de país	Costo
5 libras	80104	ANEGTOS	7.25
5 libras	T2J 2M7	ESO	9.40

Figura 18-4 Prueba de camino feliz revisada

Este sencillo ejemplo ilustra el ir y venir iterativo entre la codificación y las pruebas. Diferentes equipos adoptan diferentes enfoques. patty y tammy podría emparejarse tanto en la codificación como en las pruebas. Tammy podría emparejarse con Paul para escribir el dispositivo para automatizar la prueba. Tammy podría estar en una oficina remota, usando una herramienta de colaboración en línea para trabajar con Patty. Patty podría escribir el La historia ejecutable se prueba a sí misma y luego escribe el código para que funcionen. practicando el desarrollo basado en pruebas de historias reales. El punto es que las pruebas y La codificación es parte de un proceso de desarrollo en el que todos los miembros del equipo. participar.

Tammy puede continuar identificando nuevos casos de prueba, incluidos casos extremos y condiciones límite, hasta que sienta que todas las áreas de riesgo han sido cubiertas por el Cantidad mínima y variedad de casos de prueba. Ella podría probar con el más pesado. Artículo disponible en el sitio web enviado al destino más caro. Ella Podría probar tener una gran cantidad del mismo artículo. Algunos casos extremos pueden ser tan improbable que no se molesta con ellos, o decide hacer una prueba pero una vez aprobado, no lo incluye en el conjunto de regresión. Algunas pruebas podrían ser es mejor hacerlo manualmente después de que haya una interfaz de usuario disponible.

poder de tres

Patty ha escrito pruebas unitarias con Hawaii como destino de envío, pero Tammy cree que sólo los destinos continentales son aceptables. Ninguna de ellos están seguros de si los destinos de apartados postales militares son aceptables. Van a ver a Polly, propietaria del producto, para preguntarle qué piensa. Están usando el Poder de tres. Cuando surgen desacuerdos o preguntas, tener tres puntos de vista diferentes es una forma efectiva de asegurarse de obtener una buena solución y No tendrás que repetir el tema más tarde. Si uno de los participantes en la discusión no está familiarizado con el tema, los demás tendrán que organizar sus Pensamientos para explicarlo claramente, lo cual siempre es útil. Involucrar a personas en diferentes roles ayuda a garantizar que los cambios en los requisitos no pasen desapercibidos. radar y miembros del equipo sorpresa más tarde.

Cuando surgen problemas inesperados, como siempre ocurre, la regla del Poder de Tres es un gran lugar para comenzar. Es posible que necesite atraer a más personas, o incluso a la todo el equipo, dependiendo de la gravedad o complejidad del problema. ¿Qué pasa si el La API del socio de envío resulta tan lenta que el tiempo de respuesta en el ¿El sitio web será inaceptable? Tanto el equipo de desarrollo como el cliente El equipo necesita explorar rápidamente soluciones alternativas.

Centrarse en una historia

Paul busca una tarea de programación en la que trabajar. Aunque las tareas de la interfaz de usuario para el La historia del costo de envío estimado todavía está en la columna "pendientes" de la tarea. tablero, está más interesado en la historia para eliminar artículos de la tienda carro, entonces toma una de esas tarjetas. Nadie tiene tiempo para empezar a escribir el pruebas ejecutables para esa historia, por lo que se lanza hacia adelante por su cuenta.

Ahora el equipo tiene dos historias en marcha. Realmente no saben cuánto tiempo tomará terminar cualquiera de las historias. Un enfoque mucho mejor sería que Paul comenzar a trabajar en una tarea de UI para la primera historia para que la historia pueda terminarse cuanto antes. Cuando una historia está terminada (lo que significa que todo el código está escrito y probado), sabes exactamente cuánto trabajo queda por hacer: cero. Si ocurriera un desastre y no se terminaron otras historias en esta iteración, hay al menos una completa historia para publicar.

Completar toda la historia no es un concepto de prueba, pero es uno que los evaluadores deben promover y seguir. Si un programador ha comenzado a codificar una historia, asegúrese Alguien también ha comenzado a trabajar en tareas de prueba para esa historia. Este es un acto de equilibrio. ¿Qué pasa si nadie ha escrito ni siquiera pruebas de alto nivel para los elementos eliminados? ¿historia? ¿Quizás esa sea la máxima prioridad en las pruebas? Por lo general, terminar una historia debe ser el objetivo antes de que el equipo pueda pasar a la siguiente historia.

A menos que el equipo sea muy pequeño, siempre hay más de una historia en progreso. en cualquier momento dado. Puede que sea más difícil, pero intenta concentrarte en terminar uno. historia a la vez. Patty está a punto de concluir la historia de los costos de envío y Paul tiene Pasó a la historia de eliminación de elementos. Patty se encuentra con un problema y no está segura. Cómo resolverlo. Paul la ayuda a terminar el código para que Tammy pueda terminarla. pruebas exploratorias y pueden marcar la historia como "terminada". Ahora tienen una mejor idea de cuánto les queda para terminar esta iteración (o al menos, cuánto tiempo les queda para terminar esta iteración). mucho en lo que todavía no tienen que trabajar).

A veces, se pueden hacer varias historias diferentes al mismo tiempo si un programador y un evaluador se unen para completar cada historia juntos. Esto funciona si el Las historias son pequeñas e independientes. Lo que no desea ver es que los programadores comiencen a codificar sin que se completen las tareas de prueba al mismo tiempo.

PRUEBAS QUE CRITICAN EL PRODUCTO

Tan pronto como estén disponibles fragmentos de código comprobables y las pruebas automatizadas que guió su paso de codificación, tómese el tiempo para explorar la funcionalidad más profundamente. Pruebe diferentes escenarios y obtenga más información sobre el comportamiento del código. Debería tener tarjetas de tareas para pruebas que critiquen el producto, tanto de cara al negocio como a la tecnología. La historia no estará "terminada" hasta que se completen todos estos tipos de pruebas.

Esto se vuelve más importante cuando todas las tareas, excepto las pruebas, están completas durante un historia. Ahora deberías poder probar desde un extremo del hilo de la historia hasta el final. otro extremo, con todas las variaciones intermedias. No pospongás esta prueba. Tú Puede encontrar requisitos que estaban en la historia pero que se omitieron en las pruebas que impulsaron el desarrollo y, por lo tanto, faltan en el código. Ahora es el momento de escribir esas pruebas y códigos que faltan. Llene todos los vacíos y agregue más valor mientras el El equipo todavía está centrado en la historia. Hacer esto más tarde costará mucho más.

Capítulo 10,
"Presentación empresarial
Pruebas que critican el
producto" y el Capítulo
11, "Pruebas
tecnológicas que critican
el producto", le ayudarán
a asegurarse de cubrir
todas las pruebas
necesarias que critican
el producto.

Tenga en cuenta que parte de lo que aprenda al probar la historia final puede considerarse "agradable de tener", tal vez haciendo que la funcionalidad sea más fácil de usar o más rápida. elementos que no formaban parte de la historia original. Consulte con su cliente. Si hay tiempo para agregarlo en la iteración y la empresa puede usar el extra valor, adelante. Estas adiciones son mucho más económicas de agregar ahora. pero no lo hagas poner en peligro otras historias al dedicar demasiado tiempo a agregar "bling" que no tiene un gran retorno de la inversión.

Si sus pruebas exploratorias llevan al equipo y a los clientes a darse cuenta de que Las historias no cubrían una funcionalidad significativa, escriba nuevas historias para iteraciones futuras. Mantenga un estricto control sobre el "desplazamiento del alcance" o su equipo no tendrá tiempo para entregar el valor que planeó originalmente.

Las pruebas tecnológicas para criticar el producto a menudo se realizan mejor durante codificación. Este es el momento de saber si el diseño no escala o si hay agujeros de seguridad.

COLABORA CON PROGRAMADORES

Nuestra viñeta que describe un equipo que escribe y utiliza pruebas detalladas para impulsar la codificación muestra cuán estrechamente colaboran los evaluadores y programadores. Esto continúa a medida que avanzan la codificación y las pruebas. Trabajar juntos mejora la capacidad del equipo para entregar el producto correcto y ofrece muchas oportunidades para transferir habilidades. Los programadores aprenden nuevas formas de realizar pruebas y serán mejores en las pruebas. su propio código mientras lo escriben. Los evaluadores aprenden más sobre el proceso de codificación y cómo las pruebas adecuadas pueden hacerlo más fácil.

Prueba de pares

Paul Programmer ha completado la interfaz de usuario para la historia de opciones de envío estimadas, pero aún no la ha registrado. Le pide a Tammy que venga a sentarse. con él y demuestra cómo el usuario final ingresaría la dirección de envío durante el proceso de pago. El costo de envío estimado se muestra a la derecha lejos. Tammy cambia la dirección de envío y ve aparecer el nuevo costo. Ella ingresa un código postal que no coincide con el resto de la dirección y ve el Aparece el mensaje de error correspondiente. La interfaz de usuario les parece buena a ambos, así que Paul revisa el código y Tammy continúa con su manual exploratorio. prueba del mismo.

A Janet le gusta que el programador "conduzca" durante estas sesiones de prueba en pareja. mientras ella observa lo que sucede. Ella considera que es mucho más eficaz que tomando el control del teclado y el mouse mientras el programador observa.

"Muéstrame"

Tammy está especialmente preocupada por cambiar la dirección de envío y tener recalcular el costo estimado, porque identificaron que era un área de riesgo. Ella descubre que si muestra el costo estimado, pasa a la dirección de facturación página, y luego regresa para cambiar la dirección de envío, los costos estimados no cambies correctamente. Ella hace que Paul venga a observar este comportamiento. Él se da cuenta hay un problema con el almacenamiento en caché de la sesión y vuelve para solucionarlo.

Mostrarle a alguien un problema y resolverlo juntos es mucho más más efectivo que registrar un error en un sistema de seguimiento de defectos y esperar a que alguien para tener tiempo de mirarlo. Es más difícil de lograr si el equipo no comparte la misma ubicación. si equipo

los miembros trabajan en zonas horarias muy diferentes, es aún más difícil. Atenerse a la comunicación más directa disponible para usted. Uno de los compañeros de equipo de Lisa está en una zona horaria adelantada 121/2 horas. Trabaja hasta altas horas de la noche, y cuando Si es necesario, llama a Lisa y trabajan juntos en los resultados de las pruebas y en los ejemplos.

 La bibliografía contiene referencias para lecturas adicionales sobre este tema.

El simple hecho de mostrar la GUI a otra persona puede ayudar a Paul a darse cuenta Ha implementado algún comportamiento erróneo. De manera similar, si Tammy tiene problemas para que su script de prueba GUI funcione, explicar que el problema podría ser suficiente para que ella se dé cuenta de lo que lo está causando. Si no hay nadie disponible para mirar en lo que acaba de codificar o ayudarle a depurar un problema, a veces ayuda explícalo en voz alta. "Agacharse" y "Pensar en voz alta" Son formas sorprendentemente efectivas de resolver sus propios problemas. A Janet le gusta tener su propio patito de goma sentado en su escritorio para recordarse a sí misma que debe pensar antes de preguntar.

HABLAR A LOS CLIENTES

Es sorprendentemente fácil para los miembros del equipo de desarrollo agachar la cabeza. generar historias y olvidarse de mantener informados a los clientes. Además de Consultar a expertos en negocios cuando tenemos preguntas, debemos mostrárselas. lo que hemos entregado hasta ahora.

Con suerte, pudo revisar casos de prueba con los clientes, o con alguien que pudiera representar al cliente, antes de comenzar la codificación. Si no, nunca lo será Demasiado tarde. Para situaciones en las que los clientes necesitan estar más involucrados con el detalles de las pruebas ejecutables, asegúrese de encontrar herramientas de prueba que funcionen para ellas como así como para los miembros del equipo técnico.

Como describimos en los dos últimos capítulos, es posible que ya haya repasado maquetas o prototipos en papel con sus clientes. Si las tareas para simular un informe o una interfaz permanecen en el plan de iteración, recuerde mantener el proceso simple. Por ejemplo, no codifique un prototipo HTML cuando dibuje en un La pizarra funcionará igual de bien. Queremos mantener el proceso lo más simple posible; la simplicidad es un valor fundamental.

Mostrar clientes

Tan pronto como esté lista una interfaz de usuario codificada o un informe, incluso si aún es rudimentario, Si carece de todas las funciones o muestra datos codificados, muéstrela a la persona adecuada. clientes. Nadie puede explicar exactamente lo que quiere de antemano. Ellos Necesita ver, sentir y usar la aplicación para saber si es correcta. Puede que no lo estés

capaz de implementar grandes cambios a mitad de la iteración, pero si comienza temprano, puede haber llegado el momento de realizar pequeños ajustes y sus clientes sabrán qué esperar.

La reunión de revisión de iteraciones es una gran oportunidad para mostrar lo que el equipo entregado y obtenga comentarios para la próxima iteración, pero no espere hasta entonces para obtener comentarios de los clientes. Manténgalos involucrados durante toda la iteración.

Entender el negocio

Aunque nos quedamos atrapados en el rápido ritmo de las iteraciones, también debemos detenernos y tomarse el tiempo para comprender mejor el negocio. Pasa algún tiempo hablando con empresarios sobre sus trabajos y qué aspectos podrían mejorarse con nuevas características del software. Cuanto mejor comprenda el negocio de su cliente, mejor podrá ser para ofrecer un buen producto.

La historia de Lisa

Mi equipo asigna tiempo para que cada miembro del equipo de desarrollo se siente con los miembros del equipo de administración del plan de jubilación mientras realizan su trabajo diario. No sólo entendemos mejor esos trabajos, sino que a menudo identificamos pequeños cambios en la aplicación que facilitarán el trabajo del administrador.

Adiciones simples, como proporcionar un poco de datos adicionales, un filtro de búsqueda adicional o cambiar el orden de una visualización, pueden marcar una gran diferencia en un proceso tedioso y detallado. También documentamos lo que aprendemos con diagramas de flujo y páginas wiki para que otros miembros del equipo puedan beneficiarse.

—Lisa

De hecho, algunos equipos se sientan permanentemente con los empresarios para que puedan estar involucrados con el negocio real a diario.

COMPLETAR TAREAS DE PRUEBA

Los evaluadores ágiles son proactivos. No nos sentamos a esperar que llegue el trabajo.

Los evaluadores que están acostumbrados a un proceso en cascada pueden sentir que no hay nada que hacer. hacer hasta que una historia esté 100% completa. Esto rara vez ocurre durante una iteración ágil.

Trabajar con programadores para que produzcan algún fragmento de código comprobable.

Temprano. El algoritmo de costos de envío presentado anteriormente es un buen ejemplo. Él Se puede probar completamente de forma aislada, sin necesidad de acceder a la base de datos. o la interfaz de usuario. Alternativamente, la interfaz de usuario podría eliminarse con datos codificados antes de que se completen los servicios que acceden a los datos reales, y el comportamiento de la capa de presentación se puede probar por sí mismo.

Peligro: la crisis de las pruebas

Incluso los equipos ágiles experimentados a menudo experimentan una crisis de pruebas al final de una iteración. Tal vez una o dos historias tomaron mucho más tiempo de lo esperado, o un problema de producción le quitó tiempo al desarrollo. ¿Qué sucederá cuando mañana finalice su iteración y su tablero de tareas (real o virtual) todavía esté lleno de tarjetas de prueba?

Si ves esto, reconócelo como un mal olor. Trabaje con el equipo para determinar cuál puede ser el problema. ¿Los programadores no trabajan lo suficientemente estrechamente con los evaluadores? ¿Hubo demasiadas interrupciones?

La forma de abordar este peligro es involucrar a todo el equipo. Recuerde que cualquier miembro del equipo puede registrarse para realizar tareas de prueba. En tu stand-up diario, puedes evaluar si el equipo está en camino de terminar todas las historias. Si hay varias historias en peligro de no completarse, elija una historia para descartarla o reduzca el alcance de una o más historias. Concéntrese en completar una historia a la vez. A medida que se acerca el final de la iteración, es posible que los programadores tengan que dejar de trabajar en nuevas funciones y comenzar a realizar tareas de prueba.

Omitir algunas funciones de una versión es mejor que omitir la versión completa porque no se pudieron completar las pruebas en todas o la mayoría de las historias.

Los programadores del equipo de Lisa automatizan regularmente las pruebas detrás de la GUI en además de pruebas unitarias y de integración. También suelen escribir el funcional casos de prueba detrás de la GUI. A veces escriben la prueba inicial ejecutable del camino feliz para poder coordinar la prueba y el diseño del código; luego un probador agrega más Casos de prueba. Ocasionalmente, escriben todos los casos de prueba funcionales, porque el Los evaluadores no tienen el ancho de banda para cubrir todas las historias de pruebas intensivas.

Todos los miembros del equipo también deben estar dispuestos a asumir tareas de prueba manuales. Si su equipo recién está comenzando y no ha podido abordar las necesidades de automatización Sin embargo, todo el equipo debe planificar el tiempo para ejecutar la prueba de regresión manual. scripts y probar manualmente nuevas funciones. Como puede atestiguar el equipo de Lisa, esto tarea proporciona una gran motivación para aprender a diseñar la aplicación para Facilitar la automatización de pruebas. Otros equipos nos dicen que esto también les funcionó.

TRATAR LOS ERRORES

Hemos conocido a muchos equipos que luchan con la cuestión de cómo realizar un seguimiento errores, o si realizar un seguimiento de ellos. Como escriben Tom y Mary Poppendieck en su libro Implementación del desarrollo de software ajustado: del concepto al efectivo [2006], las colas de defectos son colas de retrabajo y, por tanto, puntos de recogida de

desperdiciar. Algunos equipos simplemente corren errores tan pronto como los descubren. ellos escriben una prueba unitaria para reproducir el error, corregir el código para que pase la prueba, verificar la prueba y la corrección del error, y continúa. Si alguien descifra ese fragmento de código más tarde, la prueba captará la regresión.

El Capítulo 5, "Transición de procesos típicos", explica por qué su equipo puede querer o no utilizar un sistema de seguimiento de defectos.

Otros equipos encuentran valioso documentar los problemas y las soluciones en un sistema de seguimiento de defectos (DTS), especialmente los problemas que no se detectaron hasta después del código. fue lanzado. Incluso pueden buscar patrones en los errores que llegaron a producción y realizar un análisis de la causa raíz para aprender cómo evitar que se produzcan problemas similares. periódico. Aún así, los sistemas de defectos no proporcionan un buen foro para las conversaciones cara a cara. comunicación sobre cómo producir código de mayor calidad.

Lisa y sus compañeros de prueba prefieren hablar con un programador tan pronto como encuentran un problema. Si el programador puede solucionarlo inmediatamente, no es necesario iniciar sesión. el error en cualquier lugar. Si no hay ningún programador disponible inmediatamente para trabajar en el problema y existe la posibilidad de que el error se olvide, escriben una tarjeta para ello o ingréselo en su EDE.

Hemos agregado esta sección a este capítulo porque es aquí cuando te encuentras con el problema. Ha estado escribiendo pruebas primero, pero encuentra problemas a medida que trabaja con el programador. ¿Registros un error? ¿Si es así, cómo? has estado haciendo tus pruebas exploratorias y encontraste un error en una historia que estaba marcada como completada. ¿Registros un error por eso? Analicemos más sobre los defectos y consideremos las opciones que están abiertas para usted y su equipo.

¿Es un defecto o es una característica?

Primero, hablemos de defectos versus características. La vieja pregunta sobre el software desarrollo es, "¿Qué es un error"? Algunas respuestas que hemos escuchado son: Es una desviación de los requisitos o su comportamiento no es el esperado. De hecho, existen algunos defectos realmente obvios, como resultados incorrectos o mensajes de error incorrectos. Pero lo que realmente importa es la percepción que tiene el usuario de la calidad del producto. Si el cliente dice que es un defecto, entonces es un defecto.

En Agile, tenemos la oportunidad de trabajar con los clientes para arreglar las cosas. a su satisfacción. Los clientes no tienen que intentar pensar en todos los posibles Característica y detalle al frente. Está bien que cambien de opinión cuando ven algo.

Al final, ¿realmente importa si es un error o una característica si es necesario solucionarlo? El cliente elige las prioridades y la propuesta de valor. Si la calidad del software

es una prioridad más alta para el cliente que obtener todas las nuevas funciones, entonces deberíamos intentar corregir todos los defectos a medida que los encontramos.

Los clientes del equipo utilizan sus conocimientos para dar el mejor consejo que pueden al equipo en el desarrollo diario. Sin embargo, cuando un producto va a UAT y está expuesto a una base de clientes más grande, siempre habrá solicitudes en forma de errores o nuevas mejoras.

Deuda técnica

Capítulo 6, "El Propósito de las pruebas", explica cómo las pruebas ayudan a gestionar la deuda técnica.

Una forma de pensar en los defectos es como deuda técnica. Cuanto más largo sea un defecto mientras más permanezca en el sistema y pase desapercibido, mayor será el impacto. También es verdad que dejar errores encenados en una base de código tiene un efecto negativo en la calidad del código, intuición del sistema, flexibilidad del sistema, moral del equipo y velocidad. Arreglando uno Un defecto en el código defectuoso puede revelar más, por lo que las tareas de mantenimiento llevan más tiempo.

Tolerancia cero a errores

Janet anima a los equipos con los que trabaja a esforzarse por lograr una "tolerancia cero" con respecto al recuento de errores. A los nuevos equipos ágiles generalmente les cuesta creer que puedan hacerlo. estar hecho. En una organización con la que trabajaba Janet, desafió a cada uno de Los cinco equipos del proyecto para ver qué tan cerca podían llegar de cero errores pendientes al final de cada iteración y cero en el momento del lanzamiento.

Iteraciones sin errores

Jakub Oleszkiewicz, director de control de calidad de NT Services [2008], relata cómo su equipo aprendió a terminar cada iteración sin que los errores se trasladaran a la siguiente.

Creo que todo se reduce a una comunicación excepcional entre los evaluadores, los desarrolladores y los analistas de negocios. La disciplina también fue clave, porque nos fijamos el objetivo de cerrar iteraciones con características completamente desarrolladas, funcionales, desplegables y libres de defectos, mientras nos esforzábamos por evitar caer en una trampa de cascada. Para nosotros, evitar la cascada significaba que teníamos que mantener la alineación con el código y las actividades de prueba; Intentamos planificar las actividades de una iteración para que los casos de prueba de una característica determinada se diseñaran y automatizaran al mismo tiempo que se escribía el código de esa característica. Nosotros Rápidamente descubrimos que estábamos practicando una forma de desarrollo basado en pruebas. No creo que fuera TDD puro, porque en realidad no estábamos ejecutando las pruebas hasta que se registraba el código, sino que estábamos desarrollando las pruebas a medida que los desarrolladores escribían el código, y los desarrolladores nos preguntaban cómo estaban estructuradas nuestras pruebas y qué nuestros resultados esperados fueron.

Por el contrario, preguntamos periódicamente a los desarrolladores cómo estaban implementando una característica determinada. Este tipo de cuestionamiento bidireccional a menudo planteaba inconsistencias en cómo se interpretaban los requisitos y, en última instancia, resaltaba defectos en nuestras interpretaciones antes de que el código se enviara realmente.

Cada mañana durante nuestro Scrum, garantizamos aún más la paridad entre los grupos funcionales dentro del equipo a través de un diálogo simple. La comunicación era ridículamente buena: nos sentábamos uno cerca del otro, a menudo incluso frente al mismo ordenador. Cuando se descubrió un defecto, el desarrollador estaba allí observando, tomando notas y comentando los requisitos. Siempre había un analista de negocios cerca para validar aún más nuestro pensamiento. A menudo, en cuestión de minutos se registraba una resolución, se implementaba en el entorno de prueba y se verificaba.

Tanto los desarrolladores como los evaluadores debían comprometerse con este enfoque, de lo contrario no habría funcionado. Sin disciplina, los desarrolladores podrían haber avanzado fácilmente hacia más funciones y dejar pasar los errores hasta el final del proyecto, arriesgándose a una iteración incompleta. Si no estuvieramos conciudadanos como lo estábamos, la comunicación se habría visto afectada; Probablemente un sistema de seguimiento de errores o un correo electrónico se habrían convertido en nuestro medio principal para comunicar defectos, lo que habría resultado en tiempos de respuesta más largos y una mayor probabilidad de reelaboración.

Como parte de cualquier desarrollo, siempre será necesario hacer concesiones. Su equipo puede decidir lanzarlo con algunos errores pendientes porque se considera Es más importante sacar nuevas funciones que arreglar problemas de bajo nivel. insectos.

TODO SE TRATA DE ELECCIONES

Los equipos han resuelto el problema de cómo manejar defectos en muchos diferentes maneras. Algunos equipos ponen todos sus errores en tarjetas de tareas. Otros equipos han optado por escribir una tarjeta, estimarla y programarla como una historia. Otros más sugieren agregando una prueba para cada error; de esa manera no es necesario registrar el defecto, solo la prueba.

¿Existe una forma correcta? ¡Por supuesto que no! Pero, ¿cómo sabes qué es lo correcto para Tu equipo? Tenemos algunas sugerencias para ayudarle a elegir y decidir qué es bien por ti. Piensa en tu equipo y tu producto y en lo que podría trabajar en su situación. Primero, hablaremos sobre qué defectos debemos registrar, luego hablaremos un poco sobre cuándo debes solucionarlos y, finalmente, veremos qué medios elegir. La combinación correcta dependerá de qué tan avanzado su equipo se encuentra en su viaje ágil y qué tan maduro es su producto.

Decidir qué errores registrar

No es necesario registrar todos los errores, pero los equipos a menudo tienen dificultades para determinar cuáles son. deben registrarse y cuáles no. Recomendamos que

Evite crear un informe de defectos si es posible. Tener una conversación con un verdadero persona primero, y sólo producir un informe de defecto si es realmente un problema real que exige un cambio en el producto o los programadores simplemente no pueden hacerlo bien lejos.

Fallos de pruebas unitarias

No registre fallas en las pruebas unitarias. Si eres parte de un equipo que practica TDD (desarrollo basado en pruebas) y tiene buena cobertura con sus pruebas unitarias, usted Tenga en cuenta que las pruebas fallidas durante la compilación no deben registrarse. Una prueba fallida durante la construcción de integración continua es una señal para que los programadores abordar el problema de inmediato. Registrar estos errores sería redundante y una pérdida de tiempo.

Fallos en pruebas de regresión de nivel superior

Muchos equipos tienen compilaciones que ejecutan pruebas de regresión por encima del nivel de unidad, como pruebas detrás de la GUI y pruebas a través de la GUI. Cuando uno de estos se construye falla, ¿debería registrar el error en un DTS?

La historia de Lisa

Tenemos dos compilaciones, una "compilación en curso" que ejecuta solo pruebas unitarias y una "compilación completa". que ejecuta las pruebas funcionales detrás y a través de la GUI. Cuando la "construcción completa" se rompe, si un desarrollador investiga y aborda el problema de inmediato, como sucede a veces, generalmente no se registra ningún error. El problema se soluciona rápidamente. En otras ocasiones, el fracaso no es sencillo. Uno de los evaluadores investiga, reduce el problema y presenta un error que indica el nombre de la prueba fallida o proporciona pasos manuales para recrear el problema.

En cualquier caso, se escriben pruebas que reproducen el error y se corrige el código para que las pruebas pasen. Las pruebas pasan a formar parte de una de las compilaciones.

—Lisa

Las pruebas fallidas en sí mismas son un tipo de error registrado. Pero a veces, como en En el caso de Lisa, es necesario agregar más información para permitir una respuesta efectiva y solución limpia, por lo que se justifica registrar el defecto.

Errores de historia dentro de la iteración actual

No registre errores que puedan solucionarse inmediatamente, especialmente si, de lo contrario, los registraría en una DTS electrónica. Si su equipo está trabajando estrechamente con

los programadores y está practicando pruebas de pares tan pronto como se completa una historia, le recomendamos encarecidamente que no registre esos errores siempre y cuando el programador los resuelva de inmediato. Cuando notes problemas, habla con el programador y decide si son problemas reales o no. Hable con el cliente si es necesario, pero tome un par de notas para recordar lo que vio y poder ajustar sus pruebas si es necesario.

Si está utilizando fichas para registrar errores, es posible que desee colocar una ficha en el tablero de tareas (o una tarjeta en su tablero electrónico) solo como recordatorio.

Errores posteriores a la iteración (o aquellos que no se pueden solucionar de inmediato)

Registre los errores que no se puedan solucionar de inmediato. Hacemos hincapié en las pruebas tempranas para detectar tantos errores como sea posible mientras los programadores todavía están trabajando en la historia. Sabemos que es más barato solucionarlos cuando se detectan a tiempo; sin embargo, a veces simplemente no los detectamos de inmediato. El programador pasó a otra historia y no puede dejarlo todo para arreglarlo ahora. Ésos son los que son buenos candidatos para la tala. A veces, un "error" es en realidad un requisito omitido y debe manejarse como una historia: estimarse y priorizarse para una iteración futura.

Desde el sistema heredado

Registre los errores que se produzcan en el sistema heredado. Si su producto existe desde hace mucho tiempo, es probable que tenga una serie de errores que han estado acechando en segundo plano esperando ser descubiertos. Cuando los encuentres, tendrás un par de opciones. Si el propietario de su producto cree que vale la pena corregirlos, registre los errores y podrá priorizarlos como parte del trabajo pendiente del producto. Sin embargo, si han existido durante mucho tiempo y no causan problemas, el propietario de su producto puede decidir que no vale la pena solucionarlos. En este caso, no se moleste en registrarlos. De todos modos, nunca se abordarán, así que no pierda el tiempo.

Encontrado en producción

Registre todos los errores de producción. Cuando su aplicación esté en producción, se deben registrar todos los errores encontrados por el cliente. Dependiendo de su gravedad, estos errores pueden corregirse inmediatamente, en el momento de la próxima versión, o se estimarán, priorizarán y se incluirán en su cartera de productos.

Elija cuándo corregir sus errores

Hay tres opciones. Todos los errores que encuentre deben clasificarse para determinar si los corrige ahora, los corrige más tarde o no los corrige en absoluto. Esta clasificación puede ser tan simple como una discusión con el programador para determinar si realmente son

errores en la historia en la que está trabajando. La clasificación puede ser una discusión con el propietario del producto para determinar si debería haber otra historia para la próxima generación. La clasificación también puede ser un proceso formal con los clientes para priorizar qué errores corregir.

Reparalo ahora

Cuantos más errores pueda corregir inmediatamente, menos deuda técnica generará su aplicación y menos inventario de "defectos" tendrá. Los defectos también son más barato repararlos cuanto antes se descubran. En un artículo de la revista iSixSigma, Mukesh Soni [2008] cita un informe de IBM según el cual el coste de reparar un error encontrado después del lanzamiento del producto fue de cuatro a cinco veces más que uno no cubierto durante el diseño, y hasta 100 veces más que uno identificado en el fase de mantenimiento (ver Figura 18-5).

La figura 18-5 muestra una estadística basada en una metodología por fases, pero la estadística sigue siendo válido para el desarrollo ágil. Es más barato arreglar los errores que se encuentran durante el desarrollo que después.

Si se encuentra un defecto durante el desarrollo de una nueva función, o es un efecto secundario de otra corrección de errores, debería solucionarse automáticamente. Pero, como siempre, esto debe aplicarse con prudencia. Por ejemplo, si se encuentra un error que los programadores dicen será difícil de arreglar y puede desestabilizar el producto, se debe llevar a los clientes a priorizar.

Fase/Etapa del Desarrollo del S/W en la que se Encuentra el Defecto

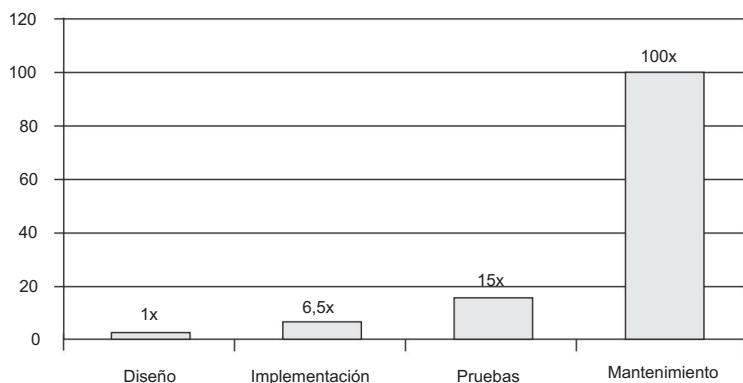


Figura 18-5 Costos relativos para reparar defectos de software (Fuente: Sistemas IBM Instituto de Ciencias)

Si corrige los errores durante el desarrollo, disminuirá la presencia de errores más adelante en el proceso. La velocidad de su equipo puede incluir tiempo para corregir errores. Con el tiempo, los miembros de su equipo tendrán una buena idea de cuánto tiempo dedican a corregir los errores encontrados por los evaluadores de una historia. Ojalá sean pocos. Si su equipo es un equipo nuevo y ágil, es posible que haya bastantes errores que escapen al desarrollo, pero a medida que el equipo se sienta más cómodo con las herramientas y los procesos, la cantidad de errores encontrados disminuirá. Para empezar, intente hacer una estimación de una historia que incluya dos horas o medio día para corregir los errores asociados.

Reparar más tarde

Diferentes equipos tienen diferentes formas de manejar los defectos. Algunos equipos creen que los clientes deben priorizar todos los defectos encontrados antes de incluirlos en la lista para solucionarlos. Creen que depende completamente del cliente determinar si realmente son defectos y, de ser así, si deben repararse.

Nunca arreglar

Su equipo ha reconocido un defecto, pero sabe que no se solucionará. Quizás esa sección de código necesite una reescritura completa más adelante porque la funcionalidad cambiará, o quizás sea simplemente un problema de tan baja prioridad o tan oscuro que es posible que sus clientes nunca lo encuentren. Hay multitud de razones por las que no se soluciona. Si su clasificación determina que este es el caso, le sugerimos que simplemente cierre el error. No lo dejes abierto pretendiendo que algún día lo arreglarás.

Elija los medios que debe utilizar para registrar un error Cuando hablamos de medios, nos referimos a la variedad de formas en que puede registrar un error. Podría ser un sistema de seguimiento de defectos o fichas, o tal vez elija no tener ningún registro físico.

Fichas

Las fichas (ya sean tarjetas reales o virtuales en un sistema de seguimiento y planificación en línea) no dejan mucho espacio para muchos detalles administrativos, pero sí dan gran visibilidad a los temas pendientes cuando se fijan en el guión gráfico, especialmente si son de otro color. Algunos equipos usan impresiones de pantalla y las grapán en el reverso de la tarjeta o escriben los detalles en un archivo de texto, o incluso graban los pasos en forma de audio en una grabadora de voz portátil.

Hay muchas opciones, pero le sugerimos que elija una que contenga suficiente información para guiar a alguien a reproducir un problema o centrar una discusión cuando el programador esté listo para solucionarlo. La tarjeta es tangible. Quinientos errores en un DTS son sólo un número. Una pila de 500 cartas es impresionante.

Utilice tarjetas en las siguientes circunstancias:

Eres un equipo ágil y disciplinado y estás solucionando todos los errores dentro de una iteración.

Quieres que los errores sean visibles para el equipo.

No hay nada que le impida tener fichas y una EDE.

Sistema de seguimiento de defectos

Utilice un DTS en las siguientes circunstancias:

Tu equipo está distribuido.

Debe realizar un seguimiento de los errores con fines de auditoría o capturarlos en notas de lanzamiento.

Tiene errores que escapan a una iteración y debe recordar corregirlos más tarde.

Tiene un sistema heredado con una gran cantidad de defectos.

De una forma u otra, probablemente querrás tener algún tipo de DTS para registrar algunos de los errores. Esto no significa que deba registrarlos todos. Se inteligente sobre cuáles registras.

Ninguno en absoluto

¿Por qué no registrarías un error? La mayoría de los equipos con los que hemos trabajado han establecido reglas por sí mismas de que ningún error se soluciona sin una prueba unitaria. Si también tienes un paquete de automatización funcional, entonces podrá detectar los errores más importantes con ellos. El argumento es que si hay una prueba que detecta el error, no tienes Necesito registrar el error. Todo lo aprendido al corregir el error se capturó en la prueba y el código. Sin embargo, es necesario reconocer que no todas las pruebas son fácil de automatizar.

Utilice pruebas para capturar errores en las siguientes circunstancias:

Su equipo es disciplinado y escribe pruebas para cada error encontrado.

Alternativas y sugerencias para lidiar con errores

A medida que los equipos maduran, encuentran procedimientos que les funcionan. ellos eliminan tareas redundantes. Adquieren más práctica en el uso de tarjetas de cuentos,

tableros y trabajos pendientes de proyectos. Usan pruebas de manera efectiva y aprenden qué errores iniciar sesión y qué métricas tienen sentido para su equipo. En esta sección, compartiremos algunas ideas que otros equipos han encontrado que les funcionan.

Establecer reglas

Establezca reglas como: "El número de tarjetas rosas (bichos) nunca debe ser mayor que diez a la vez". Vuelve a visitarlos cada vez que tengas una retrospectiva del equipo. Si Su tasa de defectos está disminuyendo, no se preocupe. Si la tendencia es la contraria, gasta tiempo analizando la causa raíz de los errores y creando nuevas reglas para mitigarlos.

Reparar todos los errores

No olvide corregir también los errores de baja prioridad encontrados durante la iteración, ya que tienen un efecto en el desarrollo futuro. En nuestra experiencia, hay Parece haber una fuerte correlación entre "baja prioridad" y "rápido arreglo", aunque no tenemos hechos concretos que lo respalden. Sugerimos detenernos poco a poco, insectos aislados antes de que se conviertan en insectos grandes y enredados.

Combinar errores

Si encuentra muchos errores en un área, piense en combinarlos en una mejora o historia.

La historia de Janet

Cuando comencé a trabajar en WestJet, encontré muchos pequeños problemas con la aplicación móvil. La aplicación funcionó correctamente, pero estaba confundido acerca del flujo. Sólo encontré estos problemas porque era nuevo y no tenía percepciones previas.

El equipo decidió agrupar los temas que había planteado y considerar el tema completo como una nueva historia. Después de estudiar el problema completo con todos los detalles conocidos, el resultado final El resultado fue una característica sólida. Si los errores se hubieran solucionado poco a poco, el efecto no habría sido tan bonito.

—Janet

Trátelo como una historia

Si un "error" es una funcionalidad que realmente se pierde, elija escribir una tarjeta para el error. y programarlo como una historia. Estas historias se estiman y priorizan al igual que cualquier otra historia. Tenga en cuenta que es posible que las historias de errores no reciban tanta atención como las nuevas historias de usuarios en la cartera de productos. También lleva tiempo crear el historia, priorizarla y programarla.

El atraso oculto

Antony Marcano, autor de www.TestingReflections.com, señala que mientras las historias de usuarios y sus pruebas de aceptación describen el comportamiento deseado, los informes de defectos describen el mal comportamiento. Detrás de cada mala conducta hay una conducta deseada, muchas veces no definida previamente. Por lo tanto, detrás de cada informe de defecto puede haber una historia de usuario oculta. Explica sus experiencias.

En el Capítulo 5, “Procesos típicos de transición”, mencionamos la teoría de Antony Marcano. publicación de blog sobre los sistemas de seguimiento de defectos como un trabajo pendiente oculto en los equipos ágiles. Antony comparte sus ideas sobre cómo sacar a la luz ese secreto.

Las publicaciones de XP sugieren que si encuentra un error, debe escribir una prueba automatizada que lo reproduzca. Muchos equipos presentan un informe de error y luego escriben una prueba automatizada por separado. He descubierto que esto da como resultado una duplicación de esfuerzos y, por lo tanto, un desperdicio. Cuando escribimos un informe de error, indicamos los pasos, lo que debería haber sucedido (expectativa) y lo que realmente sucedió (contra-expectativa). Una prueba automatizada le dice lo mismo: los pasos, las expectativas y ejecutarla por primera vez deberían demostrar la anti-expectativa. Cuando puede escribir una prueba de aceptación automatizada tan fácilmente como escribe un informe de error, la prueba se comunica tanto como el informe de error, sus trabajos pendientes y los guiones gráficos le permiten administrar el trabajo involucrado en solucionarlo, entonces, ¿por qué escribir? ¿Un informe de error por separado?

Las métricas de errores son todo lo que queda. Las métricas de errores se utilizan tradicionalmente para ayudar a predecir cuándo el software estará listo para su lanzamiento o resaltar si la calidad está mejorando o empeorando. En los enfoques de prueba primero, en lugar de decirnos si la calidad está mejorando o empeorando, nos dice qué tan buenos éramos en la predicción de las pruebas, es decir, qué tan grandes eran las brechas en nuestro pensamiento original. Esta es información útil para retrospectivas y se puede lograr simplemente etiquetando cada prueba con detalles de cuándo se identificó: elaboración de la historia, exploración posterior a la implementación o en producción. En cuanto a predecir cuándo podremos lanzarlo (cuando estemos completando software de “calidad liberable” en cada iteración), este trabajo se maneja mediante gráficos de quemado/quemado y similares.

Con un nuevo proyecto en el que estaba trabajando, sugerí que comenzáramos a usar un sistema de seguimiento de errores cuando la necesidad de uno fuera apremiante. Capturamos el resultado de las pruebas exploratorias realizadas dentro de la iteración como pruebas automatizadas en lugar de informes de errores. Determinamos si la prueba pertenecía a la historia actual, a otra historia o si estas pruebas inspiraron nuevas historias. Gestionamos estas historias como lo haríamos con cualquier otra historia y utilizamos gráficos de evolución para predecir cuánto alcance se alcanzaría al final de la iteración. Al final ni siquiera configuramos un sistema de seguimiento de errores.

Sin embargo, existe una diferencia entre las historias de usuarios típicas y las historias de usuarios inspiradas en errores. Anteriormente, nuestras historias y pruebas solo trataban de comportamientos faltantes (es decir, características que sabemos que queremos implementar en el futuro).

Ahora, también comenzaron a incluir información resumida sobre el mal comportamiento en nuestra historia de usuario propuesta para ayudar al cliente a priorizarla mejor. Por ejemplo:

Como un registrado, se me ~~impide~~ ~~adopta~~ ~~el~~ ~~caso~~ deseado, si mi contraseña se ingresa usando el caso adivinar mi ~~contraseña~~ ~~de~~ ~~acceso~~ ~~a~~ ~~esta~~ ~~contraseña~~, nadie puede

acceso

El cliente entendió que "en lugar de" significaba "eso es algo que sucede actualmente", lo cual es un mal comportamiento y no simplemente un comportamiento aún por implementar.

Al utilizar este enfoque de solo prueba para capturar errores, he notado que las historias inspiradas en errores se priorizan más que las historias de usuarios de nuevas funciones, mientras que antes a menudo prestaban más atención a las "nuevas funciones interesantes" en la cartera de productos. que los malos comportamientos descritos en el seguimiento de errores. Fue entonces cuando me di cuenta de que los sistemas de seguimiento de errores están esencialmente ocultos, o ~~en~~ ~~los~~ ~~atrasos~~ secretos.

En algunos equipos, sin embargo, ocurre lo contrario. Las políticas de corrección de todos los errores pueden prestar más atención a los errores a expensas de características nuevas quizás más importantes en el trabajo pendiente principal.

Ahora, si estoy entrenando a un equipo a mitad de un proyecto, les ayudo a encontrar formas mejores y más rápidas de escribir pruebas automatizadas. Les ayudo a utilizar esas mejoras al escribir pruebas automatizadas derivadas de errores. Les ayudo a encontrar la historia adecuada (nueva o existente) y les ayudo a aprovechar la información agregada útil para las retrospectivas. Con el tiempo, llegan a la misma conclusión que yo: el seguimiento de errores tradicional comienza a parecer un desperdicio y redundante. Es entonces cuando deciden que ya no quieren ni necesitan un trabajo atrasado oculto.

Si los errores simplemente se registran en un DTS, es posible que se pierda información importante del proyecto. Cuando redactamos pruebas de aceptación para impulsar el desarrollo, tendemos a centrarnos en el comportamiento deseado. Aprender sobre el comportamiento no deseado a partir de un defecto y convertirlo en historias es una adición vital para producir la funcionalidad adecuada.

Pegatinas azules, verdes y rojas

Cada equipo necesita determinar el proceso que funciona para él y cómo hacer ese proceso es fácilmente visible. La siguiente historia trata sobre un proceso que funcionó. para Janet.

Hace unos años, trabajé en un sistema heredado con muchos errores ya registrados en el sistema antes de que se introdujera Agile. Uno de los desarrolladores insistió en que no utilizaría un sistema de seguimiento de defectos. Él creía firmemente que

fueron una pérdida de tiempo. Sin embargo, los evaluadores necesitaban que se registraran los defectos porque eran muchos.

El equipo llegó a un compromiso que funcionó para todos. Los errores que se encontraron durante las pruebas en pareja con los programadores no se registraron porque se solucionaron de inmediato. Todos los demás se registraron en el DTS. Los errores que debían corregirse en la iteración actual se registraron en tarjetas rosas con el resumen y el número del error y luego se colocaron en el guión gráfico. Todos los demás pasaron a formar parte de la cartera de productos.

Los programadores podían observar los detalles del sistema pero también pedían a los evaluadores más información, si era necesario. Debido a que los temas estaban en el guión gráfico, se convirtieron en parte de las discusiones y enfrentamientos diarios. Cuando se solucionaba un error, los programadores escribían la solución y cualquier información adicional en el reverso de la tarjeta.

Pusieron una pegatina azul en la tarjeta para que los evaluadores supieran que estaba lista para la prueba. Una etiqueta verde significaba que se había verificado que se había arreglado y una etiqueta roja significaba que no lo estaba. arreglado y necesitaba más trabajo. Por supuesto, hubo muchas conversaciones entre los evaluadores y los programadores. James, uno de los programadores, y yo nos divertimos mucho con un error que simplemente no se solucionaba. Al final, la tarjeta parecía tener una oruga: azul, roja, azul, roja, azul y finalmente verde.

Todos estábamos muy emocionados cuando ese error fue eliminado.

Los evaluadores solucionaron errores y realizaron la mayor parte de la administración, porque el DTS era su requisito. Después de un tiempo, los programadores comenzaron a ingresar lo que arreglaron. en el sistema de seguimiento de defectos porque era más fácil que escribir en la tarjeta. El equipo siguió usando las tarjetas debido a la visibilidad. Fue fácil ver de un vistazo cuántos errores pendientes había en la iteración o en el trabajo pendiente.

—Janet

Este enfoque funcionó para este equipo porque había mucha disciplina en el equipo, y la mayoría de los errores nuevos se corrigieron en la iteración si eran parte de la funcionalidad nueva o modificada. Los únicos errores que se incluyeron en el trabajo pendiente Eran errores heredados que se consideraban de bajo riesgo.

Comience simple

Sugerimos utilizar un sistema lo más simple posible y aplicar la complejidad lo más posible. requerido. El código producido primero como prueba está, según nuestra experiencia, bastante libre de errores para cuando se registre. Si encuentra muchos errores en el código nuevo, su El equipo necesita descubrir por qué y tomar medidas. Intenta acortar el ciclo de codificar, integrar y probar para que los programadores obtengan información inmediata sobre la calidad del código. Quizás sea necesaria alguna sección con errores del código heredado. que ser rediseñado antes de que sume a su equipo en una deuda técnica. Tal vez tú Es necesario trabajar más estrechamente con los expertos empresariales para comprender la funcionalidad deseada.

Más sobre retro
perspectivas en el
Capítulo 19,
"Conclusión de la iteración".

Otra idea podría ser crear una lista continua de "iniciar, detener y continuar" para que Puede recordar algunos de los problemas durante la retrospectiva de la iteración.

FACILITA LA COMUNICACION

La reunión diaria ayuda a los equipos a mantener la comunicación estrecha que necesidad. Todos los miembros del equipo aprenden el estado actual de las tareas e historias, y pueden ayudarse mutuamente con los obstáculos. A menudo, los programadores de audición describen Las tareas en las que están trabajando proporciona una pista de que pueden haber entendido mal. los requisitos del cliente. Eso señala la necesidad de una discusión grupal después del stand-up. Si un evaluador necesita ayuda con un problema de prueba que ha surgido, Podría pedirle al equipo que se quede después del stand-up para hablar sobre ello. Las tareas perdidas son A menudo se identifican durante las reuniones y se pueden escribir nuevas tarjetas en el acto.

El stand-up es un buen momento para observar el progreso. Utilice gráficos grandes y visibles, como como guiones gráficos, gráficos de evolución y otras señales visuales para ayudar a mantener el enfoque y conocer tu estado. Si el final de la iteración se acerca y la codificación Si una historia parece "estancada", levante una bandera roja y pregunte al equipo qué se puede hacer. al respecto. Quizás un poco de emparejamiento o ayuda adicional haga que las cosas funcionen. Lisa ha notado a menudo cuando quedan muchas pruebas por hacer y el tiempo se acaba. Ella Pide ayuda para tomar el relevo. Todo el equipo se centra en lo que hay que hacer. hacer para completar cada historia y habla sobre el mejor enfoque.

Cuando los equipos utilizan un medio electrónico para realizar un seguimiento de las historias, existe una Tendencia a olvidar el storyboard. Janet descubre que tener ambas cosas puede parecer parece una duplicación de esfuerzos, pero la visibilidad del progreso para el equipo supera con creces la sobrecarga adicional de escribir las tarjetas de tareas y moverlas según sea necesario. están completos. Tener el guión gráfico le da a su equipo enfoque durante la stand-ups o cuando estás hablando con alguien fuera del equipo sobre tu progreso.

Los probadores facilitan la comunicación

Los evaluadores pueden ayudar a que la iteración avance sin problemas ayudando a hacer Asegúrese de que todos se comuniquen lo suficiente. Habla con los programadores cuando Empiece a trabajar en una historia y asegúrese de que la comprendan. Lisa descubre que puede escribir todas las pruebas y ejemplos que quiera en la wiki del equipo, pero si nadie se molesta en leerlos, no ayudan. En caso de duda, ella se acerca requisitos y pruebas con el programador que recoge las tarjetas de tareas.

Los programadores siempre tendrán preguntas mientras desarrollan una historia, incluso si comprender bien el negocio y la historia. Es mejor si hay un cliente disponible.

responder preguntas, porque esa es la comunicación más directa. Probadores no debería interponerse en eso; sin embargo, hemos observado que los expertos en negocios a veces tienen problemas para explicar un requisito, o un programador simplemente tiene una idea equivocada y no puede ponerse en sintonía con el cliente. El poder de tres se aplica aquí. Los evaluadores pueden ayudar a los clientes y programadores a encontrar un lenguaje común.

Una pequeña competencia amistosa

Gerard Meszaros, conocido entrenador ágil y autor de [2007], compartió xUnidadPrueba Patrones esta historia sobre un equipo con el que estaba trabajando y cómo un juego resolvió un problema de comunicación.

Estábamos teniendo problemas para lograr que los desarrolladores hablaran con los empresarios sobre sus suposiciones. Cuando hablaban, el evaluador a menudo quedaba fuera del circuito. El evaluador a veces discutía algo con la empresa pero nunca se lo transmitía al desarrollador. Nuestra directora de proyecto, Janice, decidió intentar cambiar el comportamiento mediante una competencia amistosa.

Todos los desarrolladores recibieron fichas de póquer azules con una "D" escrita en ellas. Todos los evaluadores recibieron una ficha roja con una "T", y los empresarios recibieron fichas amarillas con una "B". Cada vez que alguien se reunía con un homólogo de otra zona, podía intercambiar un chip con cada persona. El objetivo era conseguir los juegos de chips más completos: TBD. El ganador recibió un trofeo TBD hecho a medida y decorado con los tres tipos de fichas. ¡El resultado final fue que todos estaban mucho más interesados en reunirse porque obtendrían más fichas!

Encuentre formas creativas de lograr que los programadores y expertos en negocios hablen y acuerden los requisitos. Si un juego de fichas de póquer les hace hablar, aprovechelo.

Facilitar la comunicación normalmente implica dibujar en una pizarra, simular interfaces, enumerar otras áreas que podrían verse afectadas o trabajar en ejemplos reales. Siempre que la comunicación parezca llegar a un callejón sin salida o la confusión sea rampante, pida un nuevo ejemplo y concéntrese en él.

La historia de Lisa

Cuando los participantes de un plan de jubilación quieren retirar dinero de sus cuentas, entran en juego muchas reglas complejas de adquisición de derechos y regulaciones gubernamentales. La situación empeora si el participante ha retirado dinero en el pasado. Al trabajar en una historia para calcular el saldo adquirido de un participante, todos los miembros de mi equipo tenían ideas diferentes sobre el algoritmo correcto, a pesar de que el propietario del producto había trabajado

varios ejemplos al comienzo de la iteración. Mi compañero de pruebas, Mike, le pidió al propietario del producto que trabajara en un nuevo ejemplo, y varios programadores y evaluadores se unieron a la sesión. Les tomó un par de horas bastante tortuosas escribir números y diagramas de flujo en una pizarra, pero finalmente llegaron a la fórmula correcta y todos estaban en la misma página.

—Lisa

Trabaje con tantos ejemplos como necesite hasta que el equipo comprenda suficientes aspectos diferentes del sistema. Pruebe con un formato diferente si no funciona. Por ejemplo, si los dibujos dibujados en la pizarra no son suficientes para entender la historia, pruebe con hojas de cálculo o algún otro formato que le resulte familiar. los expertos en negocios.

Capítulo 9, "Kit de herramientas para empresas-Enfrentando pruebas que Apoyen el Team", habla sobre algunas herramientas que pueden ayudar a los equipos distribuidos.

Equipos distribuidos

Como hemos señalado en otros capítulos, tener miembros del equipo en diferentes ubicaciones y diferentes zonas horarias significa que tienes que trabajar más duro en la comunicación. Los teléfonos, el correo electrónico y la mensajería instantánea constituyen los elementos básicos de la comunicación, pero Cada vez se desarrollan mejores herramientas de colaboración.

La historia de Lisa

Uno de los programadores de nuestro equipo, que también es gerente, se mudó a la India. Nanda trabaja allí hasta altas horas de la noche, por lo que está disponible para el equipo de Denver por las mañanas. Tiene un teléfono celular con un número de teléfono local de Denver, por lo que es fácil hablar con él por teléfono, así como por mensajes instantáneos y correo electrónico. Programamos reuniones en las que discutimos historias, como reuniones de estimación, sesiones de lluvia de ideas y planificación de iteraciones, temprano en la mañana para que él pueda participar. Aunque el equipo no puede ser tan productivo como lo éramos cuando estábamos ubicados juntos, aún podemos beneficiarnos de la experiencia en el dominio y el profundo conocimiento del software de Nanda.

Si Nanda contrata más miembros del equipo en India, es posible que tengamos que abordar cuestiones más complejas, como la coordinación de la integración y las construcciones. Podemos considerar soluciones técnicas más sofisticadas a los problemas de comunicación.

—Lisa

Necesitará experimentar para ver qué funciona para su equipo distribuido. Usar retrospectivas para evaluar si la colaboración y la comunicación necesitan mejorar y pensar en formas de mejorar. Usted, como evaluador, puede tener mucho de experiencia en ayudar con proyectos de mejora de procesos. Solo piensa en mejorar la comunicación como una de esas necesidades de mejora continua.

La historia de un probador remoto

A veces, los evaluadores son los miembros remotos del equipo. Erika Boyer de iLevel by Weyerhaeuser vive en la costa este y trabaja con un equipo en Denver. ella es Probadora de profesión, pero en su equipo todas las tareas están en juego. Podría escribir accesorios para automatizar una prueba de FitNesse o asociarse con un programador para escribir código de producción. Poder ponerse en contacto con las personas cuando las necesita es un problema. Si no recibe respuesta cuando envía un mensaje instantáneo a un compañero de trabajo, llama por teléfono; Cada área de trabajo en la oficina de Denver tiene un teléfono. Es no es infalible, porque todos podrían estar en la sala de descanso de una fiesta de despedida y olvidarse de decírselo. Los equipos en diferentes ubicaciones deben hacer un esfuerzo especial para mantenerse informados unos a otros.

Debido a que Erika comienza a trabajar unas horas antes de la reunión diaria del equipo, necesita trabajo que pueda hacer sola durante ese tiempo. Trabaja con los miembros del equipo que llegan temprano en Denver y conversa con otros programadores al final del día sobre el trabajo que hará a la mañana siguiente.

Erika puede ver las tareas del equipo utilizando una herramienta en su intranet que muestra cada tarea, su estado y su porcentaje de finalización. Con algunas adaptaciones adicionales, el equipo (que tiene otros miembros remotos) puede mantener una buena comunicación.

Incluso desde la distancia, Erika ha podido transferir habilidades de prueba a los programadores, pero ha descubierto que piensan de manera diferente a los evaluadores. Su equipo utiliza estas diferentes perspectivas a su favor al rotar todo tipo de tareas entre todos los miembros del equipo.

Los equipos exitosos mantienen a los miembros remotos "informados" y comparten habilidades y experiencia. Los equipos distribuidos enfrentan desafíos adicionales para completar con éxito las actividades de prueba, pero algunos ajustes menores, la consideración por parte de todos los miembros del equipo y buenas herramientas de comunicación ayudan a garantizar que los evaluadores remotos puedan ser productivos.

Todos necesitamos poder comunicarnos bien entre nosotros para nuestros proyectos. para triunfar. Cuando los equipos están en diversas ubicaciones geográficas, es posible que tengan trabajar el doble para estar en contacto constante.

PRUEBAS DE REGRESIÓN

A menos que esté en un equipo que recién esté comenzando sus esfuerzos de automatización, tiene pruebas de regresión automatizadas que cubren historias de iteraciones anteriores. Con suerte, estos se están ejecutando como parte de un proceso de construcción continuo, o al menos parte de un proceso de construcción diario. Si no es así, pídale a su equipo que implemente esto. infraestructura crítica sea una prioridad, y haga una lluvia de ideas con ellos sobre cómo esto podría estar hecho. Planifique el tiempo en la próxima iteración para iniciar un proceso de construcción.

Mantenga la construcción "verde"

Los programadores deben ejecutar todas las pruebas unitarias automatizadas antes de incorporar código nuevo. Sin embargo, las pruebas unitarias pueden fallar en la compilación continua, ya sea porque alguien olvidó ejecutarlas antes del check-in o debido a una diferencia en el entorno de ejecución o IDE. Tenemos pruebas unitarias por una razón, por lo que cada vez que una falla, la máxima prioridad del equipo (aparte de un problema de producción espectacular) debe ser solucionarlo y hacer que la compilación funcione nuevamente.

Los equipos adoptan diferentes enfoques para asegurarse de que su construcción siga siendo "verde". El equipo de Lisa tiene un proceso de compilación que envía los resultados por correo electrónico después de cada compilación. Si la compilación falla, la persona que registró la falla generalmente la soluciona de inmediato. Si no está claro por qué falló la compilación, los miembros del equipo se reunirán para investigar. Su ScrumMaster tiene un juguete de peluche que coloca sobre el escritorio de la persona que "rompió la construcción", como recordatorio visual de que debe arreglarse de inmediato.

Algunos equipos utilizan un semáforo, un orbe ambiental, una herramienta de monitoreo de compilación GUI u otra forma visual electrónica para mostrar el estado de la compilación. Cuando las luces se ponen rojas, es hora de detener el nuevo desarrollo y arreglar la construcción. Otra técnica es hacer que aparezca una pantalla emergente en el IDE de todos mostrando que la compilación falló y la ventana emergente no desaparecerá hasta que hagas clic en "Ok, arreglaré la compilación". Diviértete un poco con ello, pero mantener la compilación en funcionamiento es un asunto serio.

En casos extremos, es posible que tenga que comentar temporalmente una prueba fallida hasta que pueda diagnosticarse, pero esta es una práctica peligrosa, especialmente para un equipo novato. Todos los miembros del equipo deben dejar de hacer lo que están haciendo si es necesario hasta que la compilación funcione nuevamente.

Mantenga la construcción rápida

La compilación debe proporcionar comentarios inmediatos, así que sea breve. Si la compilación tarda más que la frecuencia promedio de registros de código, las compilaciones comienzan a acumularse y los evaluadores no pueden obtener el código que necesitan probar. La pauta de XP para el tiempo de construcción es de diez minutos [Fowler, 2006]. El equipo de Lisa intenta que la construcción dure menos de ocho minutos, porque se registran con mucha frecuencia.

Las pruebas que tardan demasiado, como las pruebas que actualizan la base de datos, las pruebas funcionales por encima del nivel de unidad o los scripts de prueba de GUI, deben ejecutarse en un proceso de compilación independiente. Si el equipo tiene hardware limitado, es posible que tengan que ejecutar la compilación "completa" con el conjunto completo de pruebas por la noche y la compilación "continua" que solo tiene pruebas unitarias continuamente durante las horas de trabajo. Vale la pena invertir en tener una compilación "completa" continua e independiente con todos los conjuntos de pruebas de regresión. El equipo de Lisa recibe retroalimentación cada 90 minutos de su construcción "completa", y esto ha

ha demostrado ser invaluable para evitar problemas de regresión. Este conjunto secundario de pruebas no impiden que un programador revise su código.

Construyendo una suite de regresión

Durante la iteración, estás automatizando nuevas pruebas. Tan pronto como estos pasen, agregue a la suite de regresión, según corresponda. Quizás no necesites todas las ventajas de la permutación incluido en el conjunto de regresión, y desea mantener las suites de regresión lo suficientemente rápido como para proporcionar comentarios oportunos. Como cada historia es completado, las pruebas que confirman su funcionalidad deben incluirse en el conjunto de regresión y ser parte del ciclo de compilación regular.

Las propias pruebas de regresión deben estar bajo algún tipo de control de versiones. Es mejor mantenerlos en el mismo sistema de control de código fuente que el código de producción. De esa manera, cuando etiquete el código para su lanzamiento en producción, la etiqueta También contiene todas las versiones de las pruebas que funcionaron con el código. En Como mínimo, mantenga una copia de seguridad diaria del código de prueba.

Cuando se agregan pruebas al conjunto de regresión, su propósito cambia. Ya no existen para ayudar a impulsar el desarrollo y no se espera que lo hagan. encontrar nuevos errores. El único propósito en la vida es detectar cambios inesperados o efectos secundarios en el sistema.

Comprobando el “panorama general”

Con suerte, escribiste tarjetas de tareas para probar la historia en el contexto de una aplicación más amplia y pruebas de regresión en otras partes del sistema para garantizar que la nueva historia funcione. no ha tenido un efecto negativo. Es posible que haya automatizado algunas de esas pruebas de un extremo a otro, como en el ejemplo del Capítulo 12, "Resumen de los cuadrantes de prueba".

Pero a veces, incluso si tiene un gran conjunto de pruebas de regresión, las pruebas exploratorias manuales pueden ser apropiadas. La historia no estará “terminada” hasta que usted también haya completado estas tareas.

RECURSOS

Al iniciar la iteración, asegúrese de que los entornos de prueba, los datos de prueba y Existen herramientas de prueba para acomodar las pruebas de las historias de esta iteración. Esperanza-

Has anticipado plenamente estas necesidades, pero es posible que algunos requisitos sólo se vuelvan obvios cuando empiezas a trabajar en una historia. Colaborar con la base de datos expertos, administradores de sistemas y otros miembros del equipo para configurar cualquier infraestructura adicional necesaria.

Es posible que haya contratado recursos externos para esta iteración para ayudar con el rendimiento, la usabilidad, la seguridad u otras formas de prueba. Incluirlos en Stand-ups y discusiones con los clientes según sea necesario. Empareja con ellos y ayudarles a comprender los objetivos del equipo. Esta es una oportunidad para elegir adquirir nuevas habilidades.

Métricas de iteración

Capítulo 15, "Actividades del probador en la planificación de lanzamientos o temporadas", habla de métricas útiles para mantener.

En el Capítulo 5, "Transición de procesos típicos", hablamos un poco sobre los propósitos de las métricas, sino porque las métricas son fundamentales para comprender cómo sus actividades de codificación y prueba están progresando, profundizaremos en ellas más aquí. Sepa qué problema está tratando de resolver antes de comenzar a medir puntos de datos y realizar todo el trabajo de analizar los resultados. En esta sección, Cubriremos algunas de las mediciones típicas que los equipos recopilan a través de la iteración.

Medir el progreso

Necesita alguna forma de saber cuánto trabajo ha completado su equipo en cualquier punto de la iteración y una idea de cuánto trabajo queda por hacer. Tú Necesito saber cuándo se vuelve obvio que algunas historias no se pueden completar. y el equipo necesita un Plan B. Gráficos de evolución de iteraciones y estimaciones versus El tiempo real para las tareas son ejemplos utilizados para medir el progreso del equipo. Que puede o no proporcionar valor para su equipo en particular.

Los tableros de historias o de tareas son una buena forma visual de conocer el estado de la iteración, especialmente si se utiliza un código de colores. Si todavía hay demasiadas tarjetas de tareas de prueba en el campo "para hacer" o no se han movido suficientes tarjetas de tareas de codificación a "Listo" o "Probado", es hora de que el equipo piense en formas de asegurarse de que todas las pruebas estén completadas. Tal vez algunos miembros del equipo necesiten dejar de codificar y comenzar a realizar tareas de prueba, o tal vez una historia o una parte menor crítica de una historia necesite posponerse hasta la próxima iteración para que se puedan realizar pruebas de todas las demás historias. estar terminado.

Esto se puede lograr tanto con guiones gráficos virtuales como físicos.

Sea creativo con sus efectos visuales para que los problemas sean visibles al instante. Recuerde que ninguna historia está "terminada" hasta que se prueba en todos los niveles apropiados.

Los equipos pueden tener otros criterios para determinar cuándo está "terminada" una historia, como si ha sido revisado por pares o se han completado las pruebas de regresión automatizadas. En el guión gráfico que se muestra en la Figura 18-6, la columna "Listo" para cada fila de la historia es la columna más a la derecha. La columna justo a la izquierda es la columna "Verificar". La historia no se considera "terminada" hasta que se hayan completado todas las cartas, incluidas las pruebas. tarjetas de tareas, están en la columna "Listo". Un vistazo al tablero es suficiente para saber qué historias están terminadas.

Incluso los equipos que no rastrean el consumo a nivel de tarea pueden hacerlo en la historia.

nivel. Saber cuánto trabajo puede hacer el equipo en cada iteración (su velocidad)

ayuda con el plan de lanzamiento general y la repriorización para cada iteración.

Simplemente puede ser suficiente saber el número de historias completadas en una iteración si tienden a promediar el mismo tamaño. Aunque los planes son, en el mejor de los casos, provisionales, es útil tener una idea de cuántas historias se pueden crear.



Figura 18-6 Story board que muestra historias y tareas de iteración

completarse con una fecha de lanzamiento definitiva o qué historias podrían realizarse en el próximo trimestre.

Métricas de defectos

Hablamos sobre métricas de defectos en el Capítulo 15, "Actividades del probador en versión o Planificación temática" que le brinda algunas ideas de alto nivel sobre qué rastrear.

Recopilar métricas sobre defectos puede llevar mucho tiempo, así que siempre considere el objetivo antes de empezar a medir. ¿Cuál es el propósito de las métricas que usted ¿Le gustaría reunirse? ¿Cuánto tiempo necesitarás para seguir la tendencia antes que tú? ¿Sabes si son útiles?

La contención de defectos es siempre una de las métricas favoritas para capturar. ¿Cuándo se encontró el defecto? En proyectos tradicionales, es mucho más fácil ya que existen requisitos y fases de codificación "difíciles". Cuando todo el equipo es responsable de calidad y todos trabajan juntos en todo momento, es mucho más difícil determinar "cuándo" se inyectó el defecto en el sistema.

Nos gustaría cuestionar la idea de que este tipo de métrica no es necesaria en desarrollo ágil. Sin embargo, si encuentra que se están filtrando muchos errores, Es posible que desees comenzar a rastrear qué tipo de errores son para poder solucionarlos. la causa principal. Por ejemplo, si los insectos se hubieran podido detectar con la unidad pruebas, entonces tal vez los programadores necesiten más capacitación para escribir pruebas unitarias. Si los errores se omiten o se malinterpretan los requisitos, entonces tal vez no Se dedica suficiente tiempo a la planificación de iteraciones o las pruebas de aceptación no están detalladas. suficiente.

Si practica la tolerancia cero con los defectos, entonces probablemente no tenga Es necesario realizar un seguimiento de los defectos durante la codificación y las pruebas. Una simple tarjeta en el El guión gráfico le dará toda la información que necesita.

Independientemente de las métricas que elija medir, opte por la simplicidad.

La historia de Janet

En una organización con la que estuve, hicimos un seguimiento del número de defectos registrados en el DTS en varias versiones. Estos fueron defectos que escaparon a la iteración o se encontraron en el sistema heredado. La figura 18-7 muestra la tendencia durante un año y medio.

Al principio, la cantidad de problemas encontrados justo después de su envío al control de calidad para las pruebas finales fue alto (33 problemas encontrados en un mes). Los clientes encontraron aún más problemas durante la UAT que duró más de dos meses porque no estaban

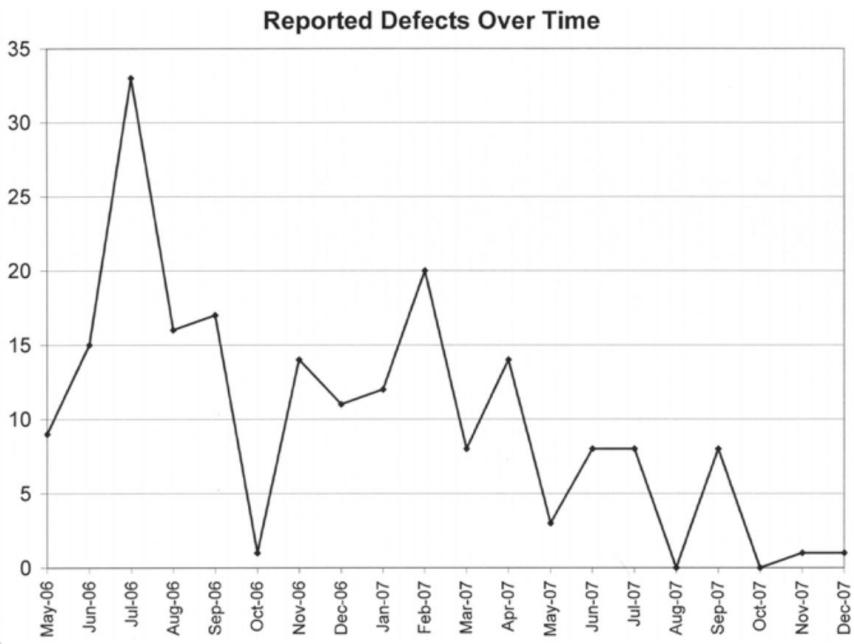


Figura 18-7 Muestra de tendencia de defectos (pero se detuvo después de un tiempo)

confiado en la calidad del lanzamiento. En el mes en el que se reportó cero defectos, recién estábamos comenzando una nueva versión, por lo que no había ninguna funcionalidad nueva que probar. Durante el año siguiente, se registraron cada vez menos defectos y resulta imposible saber dónde se produjo una liberación real con solo observar la tendencia.

Este gráfico se utilizó para mostrar a los clientes que el equipo se estaba volviendo coherente con sus pruebas y lanzamientos. Una vez que el equipo y los clientes tuvieron fe en que los números no aumentarían, las métricas ya no fueron necesarias y se eliminaron.

—Janet

No tenga miedo de dejar de utilizar métricas cuando ya no sean útiles. Si el problema para el que se reunieron inicialmente ya no existe, no hay razón para sigue reuniéndolos.

Es posible que su equipo tenga que proporcionar métricas a los altos directivos o a una Oficina de Gestión de Proyectos (PMO), especialmente si trabaja para una organización grande. Patricio Fleisch, consultor de Accenture que trabajaba como analista funcional en una

Métricas de iteración útiles

Coni Tartaglia, gerente de pruebas de software en Primavera Systems, Inc., explica algunas formas que ha encontrado para lograr métricas de iteración útiles.

Recopilar métricas al final de la iteración es particularmente útil cuando muchos equipos diferentes están trabajando en los mismos lanzamientos de productos. Esto ayuda a garantizar que todos los equipos finalicen la iteración con el mismo estándar de "hecho". Los equipos deben ponerse de acuerdo sobre lo que se debe medir. Lo que sigue son algunos estándares para software potencialmente distribuible [Schwaber 2004] y diferentes formas de juzgar el estado de cada uno.

- Los entregables de Sprint se refactorizan y codifican según los estándares.

Utilice una herramienta de análisis estático. Céntrese en datos que sean útiles y procesables. Decida en cada sprint si es necesaria una acción correctiva. Por ejemplo, utilice una herramienta de código abierto como FindBugs y busque un aumento en cada sprint en la cantidad de problemas de prioridad uno. Corrijalos en consecuencia.

- Los entregables de Sprint se prueban unitariamente.

Por ejemplo, observe los resultados de la cobertura del código en cada sprint. Cuente la cantidad de paquetes con cobertura de prueba unitaria que se encuentren en rangos del 0 % al 30 % (cobertura baja), 31%-55% (cobertura promedio) y 56%-100% (cobertura alta). Los paquetes heredados pueden estar en el rango de cobertura baja, mientras que la cobertura de los paquetes nuevos debería estar en el rango del 56% al 100%, si está practicando un desarrollo basado en pruebas. Es deseable un aumento en el rango de cobertura alta.

- Los entregables de Sprint pasan pruebas de aceptación automatizadas.

Asigne pruebas de aceptación automatizadas a los requisitos de un sistema de gestión de calidad. Al final de la iteración, genere un informe de cobertura que muestre que todos los requisitos seleccionados como objetivos para la iteración han superado las pruebas. Los requisitos que no demuestran la cobertura del examen aprobado no están completos. El mismo enfoque se ejecuta fácilmente utilizando tarjetas con historias en un tablón de anuncios. La intención es simplemente mostrar que las pruebas acordadas para cada requisito o historia se pasan al final del sprint.

- Los entregables de Sprint se integran exitosamente.

Verifique los resultados de las pruebas de compilación de integración continua para asegurarse de que estén pasando. Ejecute otras pruebas de integración durante el sprint. Realice correcciones antes del comienzo de la siguiente iteración. Duda en iniciar una nueva iteración si las pruebas de integración fallan.

- Los entregables de Sprint están libres de defectos.

Los requisitos completados durante la iteración deben estar libres de defectos.

- ¿Se puede enviar el producto en [30] días?

Simplemente hágase esta pregunta al final de cada iteración y continúe con la siguiente iteración según la respuesta.

Métricas como esta son fáciles de recopilar y analizar, y pueden brindar oportunidades valiosas para ayudar a los equipos a corregir su rumbo. También pueden confirmar los estándares de ingeniería que los equipos han implementado para crear software potencialmente distribuible en cada iteración.

compañía de software durante el tiempo que escribimos este libro, nos dio lo siguiente ejemplos de métricas que su equipo proporciona a su PMO.

Números de ejecución de pruebas por piso y área funcional

Estado de automatización de pruebas (número de pruebas automatizadas frente a manuales)

Gráfico de líneas del número de pruebas aprobadas/falladas a lo largo del tiempo

Resumen y estado de cada historia.

Métricas de defectos

La recopilación y presentación de informes de métricas como éstas pueden generar importantes gastos generales.

Busque las formas más sencillas de satisfacer las necesidades de su organización.

RESUMEN

En este punto de nuestra iteración de ejemplo, nuestro evaluador ágil trabaja en estrecha colaboración con programadores, clientes y otros miembros del equipo para producir historias en pequeños

Incrementos de prueba-codificación-revisión-prueba. Algunos puntos a tener en cuenta son:

La codificación y las pruebas son parte de un proceso durante la iteración.

Escríba pruebas detalladas para una historia tan pronto como comience la codificación.

Impulse el desarrollo comenzando con una prueba sencilla; cuando pasen las pruebas simples, escriba casos de prueba más complejos para guiar aún más la codificación.

Utilice técnicas sencillas de evaluación de riesgos para ayudar a centrar los esfuerzos de prueba.

Utilice el “Poder de Tres” cuando los requisitos no sean claros o las opiniones varíen.

Concéntrese en completar una historia a la vez.

Colaborar estrechamente con los programadores para que las pruebas y la codificación estén integradas.

Las pruebas que critican el producto son parte del desarrollo.

Mantenga a los clientes informados durante toda la iteración; permítale revisar temprano y con frecuencia.

Todos los miembros del equipo pueden trabajar en tareas de prueba.

Los evaluadores pueden facilitar la comunicación entre el equipo del cliente y el equipo de desarrollo.

Determine cuál es la mejor opción de "corrección de errores" para su equipo, pero un buen objetivo es intentar no tener errores en el momento del lanzamiento.

Agregue nuevas pruebas automatizadas al conjunto de regresión y prográmelo para que se ejecute con la frecuencia suficiente para proporcionar comentarios adecuados.

Las pruebas exploratorias manuales ayudan a encontrar los requisitos faltantes una vez codificada toda la aplicación.

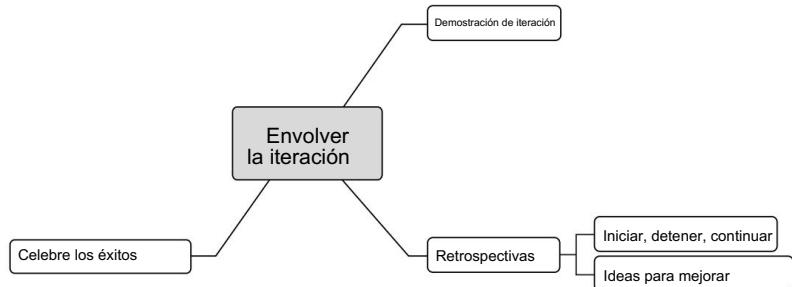
Colabore con otros expertos para obtener los recursos y la infraestructura necesarios para completar las pruebas.

Considere qué métricas necesita durante la iteración; Las métricas de progreso y defectos son dos ejemplos.

Esta página se dejó en blanco intencionalmente.

Capítulo 19

FINALIZAR LA ITERACIÓN



Hemos completado una iteración. ¿Qué hacen los evaluadores mientras el equipo concluye esta iteración y se prepara para la siguiente? Nos gusta centrarnos en cómo nosotros y el resto de nuestro equipo podemos mejorar y ofrecer un mejor producto la próxima vez.

DEMOSTRACIÓN DE ITERACIÓN

Uno de los placeres del desarrollo ágil es la oportunidad de mostrar resultados completos historias a los clientes al final de cada iteración. Los clientes pueden ver un verdadero, Aplicación en vivo y funcional. Pueden hacer preguntas y dar comentarios. Todos los involucrados en el proyecto, tanto desde el punto de vista comercial como técnico, obtienen para disfrutar de una sensación de logro.

En el equipo de Lisa, los evaluadores realizan la revisión de la iteración. entre todo el equipo miembros, generalmente han trabajado en la mayoría de las historias. Tienen una naturaleza papel como proveedores de información y tienen una buena idea de lo que los clientes Necesito saber acerca de la nueva funcionalidad. Hacer que los evaluadores muestren los resultados es una práctica común, aunque no existe una regla estricta. El Los expertos en negocios del equipo también son una buena opción para realizar la demostración. porque tienen la mejor comprensión de cómo el software cumple con los necesidades del negocio y se sentirán más dueños del producto. El Scrum-Master, un programador o un analista de negocios podrían demostrar las nuevas características y a menudo lo hacen. Janet anima a rotar este honor.

Escuchando a los clientes

Pierre Veragen explica cómo su equipo utiliza demostraciones de iteración.

"Nos callamos y escuchamos a nuestros clientes. Se trata de la química de la presentación del grupo. De alguna manera, compartir el momento une los cerebros: miramos las cosas desde una perspectiva diferente. El evento da origen a ideas y conceptos. Algunos mueren mientras habla la siguiente persona; algunos siguen vivos y se convierten en esa gran idea que diferencia el producto".

La demostración es una oportunidad para mostrar las nuevas historias, pero los comentarios que brindan los clientes son la razón más importante para hacerlo.

Cualquier persona puede tomar nota de los comentarios realizados por los clientes mientras participan en la demostración, pero los evaluadores son buenos candidatos. Es posible que observen inconsistencias no detectadas previamente a medida que avanza la demostración. A medida que surgen preguntas, los clientes pueden decidir qué quieren cambiar algo menor, como un texto de ayuda, o algo más grande, como cómo se comporta una característica. Por lo general, se pueden realizar cambios menores en las tareas y abordarlos en la siguiente iteración, pero algunos cambios son lo suficientemente grandes como para convertirlos en historias para planificar lanzamientos futuros.

Las demostraciones de iteración (llamadas revisiones de sprint en el mundo Scrum) son una gran oportunidad para que todos hablen y piensen sobre la aplicación. Llevar ventaja de ello. Las reuniones de revisión suelen ser breves y pueden durar menos de media hora. Si queda tiempo después de mostrar nuevas historias, pregunte a los clientes si han experimentado algún problema con la versión anterior que no hayan tenido informó. ¿Tienen alguna inquietud general, necesitan ayuda para comprender cómo utilizar una función o han surgido nuevos problemas? Por supuesto que puede hablar con los clientes en cualquier momento, pero teniendo a la mayoría de las partes interesadas en la sala con el equipo de desarrollo puede generar ideas interesantes.

RETROSPECTIVAS

El desarrollo ágil significa mejorar continuamente la forma de trabajar, y las retrospectivas son un excelente lugar para comenzar a identificar qué y cómo se puede hacerlo mejor. Recomendamos tomarse un tiempo al final de cada iteración y lanzamiento. Ciclo para mirar atrás y hablar sobre lo que salió bien, lo que no y lo que Quizás le gustaría probar en la próxima iteración. Existen diferentes enfoques para realizar sesiones retrospectivas. No importa qué enfoque utilice, es clave que Cada miembro del equipo se siente seguro, todos son respetados y no hay acusaciones ni culpas.

La idea es mejorar el proceso, paso a paso.

Iniciar, detener, continuar

Un ejercicio común utilizado en retrospectivas de iteraciones es "comenzar, detener, continuar".

El equipo se pregunta: "¿Qué salió bien durante esta iteración pasada? ¿Qué pasó que no debería volver a pasar? ¿Qué podemos empezar a hacer para ayudar?

¿Cosas que no salieron bien? Cada miembro del equipo puede sugerir cosas que comenzar a hacer para mejorar, cosas que dejar de hacer y que no estaban funcionando y cosas que si están funcionando. esa ayuda debe continuar. Un facilitador o ScrumMaster los enumera en una pizarra o una hoja de papel grande. Publíquelos en un lugar donde todos puedan léalos nuevamente durante la iteración. La Figura 19-1 muestra una retrospectiva en curso de "detener, iniciar y continuar". El ScrumMaster (de pie) está escribiendo detenido, comience y continúe con las sugerencias en la hoja de papel grande del guión gráfico.



Ágil Retrospectivas:

Hacer grandes a los buenos equipos

[2006] tiene ideas imaginativas para hacer que las retrospectivas sean más productivas (ver la bibliografía).

Algunos equipos inician este proceso con anticipación. Todos los miembros del equipo escriben "comienzo", "detener" y "continuar" elementos en notas adhesivas, y luego, durante la reunión retrospectiva, colocaron las notas adhesivas en la pizarra y las agruparon por tema.

"Comenzar, detener, continuar" es sólo un ejemplo de los términos que podría utilizar. Alguno otras ideas son: "Cosas que salieron bien", "Cosas para mejorar", "Agradable" "Frustrante" y "Intentarlo". Utilice cualquier nombre que funcione para usted. Puede ser



Figura 19-1 Una retrospectiva en progreso

Es difícil recordar las últimas dos semanas, y mucho menos un lanzamiento completo, si eso es lo que cubre su retrospectiva. Investiga diferentes enfoques creativos para reflexionar sobre las experiencias de tu equipo.

Aquí hay una lista de muestra de "detener, comenzar, continuar" del equipo de Lisa:

Comenzar:

Enviándonos las historias del próximo sprint antes.

No realice un procesamiento lento de un solo registro. Piense en cada llamada de servicio como una llamada remota.

Comuníquese cualquier cambio en la base de datos a todos.

Detener:

Aceptar historias sin requisitos completos.

Continuar:

Ejecutar pruebas de FitNesse para el código en el que estás trabajando.

Documentar lo que surgió en reuniones o discusiones informales.

Comunicarse mejor entre nosotros.

Mostrando maquetas temprano.

Haciendo un desarrollo impulsado por FitNesse.

Si la lista de elementos de "iniciar, detener y continuar" es larga, es una buena idea elegir uno o dos en los que centrarse para la nueva iteración. Para priorizar los elementos, dé a cada miembro del equipo "n" votos que pueda asignar a los elementos. Las diez personas del equipo de Lisa obtienen cada una tres votos y pueden aplicarlos todos a un elemento si consideran que es el más importante, o pueden votar por dos o tres elementos diferentes. Los elementos con mayor número de votos se anotan como elementos de enfoque. Janet también ha tenido éxito con esta forma de priorizar.

Además de los elementos de "iniciar, detener, continuar", el equipo puede simplemente escribir tarjetas de tareas para las acciones que se llevarán a cabo en la siguiente iteración. Por ejemplo, si la compilación en curso es demasiado lenta, escriba una tarjeta para "realizar la compilación en curso en menos de diez minutos".

En la siguiente iteración, tómate un tiempo para observar uno o dos elementos de enfoque que deseas mejorar. Al final de esa iteración, realice un punto de control para ver si mejoró. Si no, pregunte por qué. ¿Deberías probar algo diferente? ¿Sigue siendo importante? Podría ser que haya perdido importancia o que realmente no fuera importante en el panorama general. Si pensaba que había mejorado un área problemática y ésta resurge, tendrá que decidir hacer algo al respecto o dejar de hablar de ello.

Hemos descubierto que las retrospectivas son una forma sencilla y muy eficaz de equipos para identificar y abordar problemas. La reunión retrospectiva es perfecta. oportunidad de plantear cuestiones relacionadas con las pruebas. Plantear los problemas de forma objetiva, manera no culpabilizante. El equipo puede discutir cada problema, cuál podría estar causándolo y escribir algunas ideas para solucionarlo.

Ideas para mejoras

Echemos un vistazo a algunos de los elementos que llegaron a la lista para mejorar. Muchas veces, un equipo identificará problemas realmente importantes pero nunca los seguirá. levantarse y hacer algo al respecto. Por ejemplo, tal vez se descubran muchos errores a nivel de unidad después de que los programadores hayan afirmado que la codificación era incorrecta. completo.

El equipo puede decidir que los programadores no están cubriendo suficiente código con pruebas unitarias. Podrían escribir un elemento de acción para ejecutar la herramienta de cobertura de código antes de registrar código nuevo, o comenzar a escribir una tarjeta de tareas de “pruebas unitarias” para cada historia para asegurarse de que estén completos. Quizás el equipo no terminó todos los Probar tareas de automatización antes de que finalice la iteración. Mientras discuten el problema, el equipo descubre que las pruebas ejecutables iniciales eran demasiado complejas y Primero deben concentrarse en escribir y automatizar una prueba simple, o emparejarse para una mejor diseño de prueba. Asegúrese de que los elementos de acción sean concretos.

Los equipos ágiles intentan resolver sus propios problemas y establecer pautas que les ayuden a mejorar. Las acciones dirigidas a un problema pueden ayudar con otros. Cuando el equipo de Lisa tuvo problemas para terminar las historias y probarlas durante cada iteración, se le ocurrieron varias reglas en el transcurso de algunas retrospectivas:

Finalice los casos de prueba de alto nivel para todas las historias antes del cuarto día de la iteración.

Entregue una historia para probar antes del cuarto día de la iteración.

Concéntrate en terminar una historia a la vez.

El 100 % de las funciones deben registrarse antes del cierre del horario laboral del penúltimo día de la iteración.

Estas reglas hicieron más que ayudar al equipo a finalizar las tareas de prueba. ellos facilitaron un flujo y un ritmo que ayudaron al equipo a trabajar a un ritmo constante y sostenible en el transcurso de cada iteración.

Comience la próxima reunión retrospectiva revisando los elementos de acción para ver qué Los artículos fueron beneficiosos. El equipo de Lisa pone caras felices, tristes o neutrales al lado de

elementos para indicar si el equipo los probó y encontró que tuvieron éxito. El equipo debe descubrir las razones detrás de las caras tristes. fueron algunos elementos ¿Simplemente olvidado? ¿Las limitaciones de tiempo impidieron que el equipo intentara una nueva actividad? ¿Pareció que después no era tan buena idea? Estas discusiones podrían llevar a cambiar el elemento de mejora o evolucionarlo a uno nuevo.

Cuando las acciones de mejora se convierten en un hábito para el equipo, ya no Ya no es necesario escribirlos en la lista de "detener, iniciar y continuar". Elementos de "inicio" que funcionan bien se pueden mover a la columna "Continuar". Algunas ideas no funcionan, o resultan innecesarios, y también se pueden eliminar de la lista para la siguiente iteración.

Consulte sus ideas de mejora y elementos de acción durante la iteración. Correo colóquelos en un lugar (en una pared o en línea) donde todos los vean con frecuencia. la de lisa El equipo a veces revisa la lista durante una reunión de pie a mitad de iteración. Si piensa en nuevas ideas de mejora durante la iteración, escríbalas. posiblemente incluso en la lista existente, para que no lo olvide en la próxima iteración.

Es una buena idea realizar un seguimiento de las cosas que se interponen en su camino durante la iteración. Mantenga un registro de impedimentos pendientes en algún gráfico grande y visible. Hablar sobre los impedimentos en cada iteración y escribir tarjetas de tareas o tomar medidas para eliminarlos.

Un enfoque para la mejora de procesos

Rafael Santos, vicepresidente de desarrollo de software y jefe de ScrumMaster de Ultimate Software, y Jason Holzer, jefe de PSR (rendimiento, seguridad, confiabilidad) Architect, nos explicó que sus equipos encontraron ineficaces las retrospectivas que utilizaban el modelo "detener, iniciar y continuar". Hicieron listas de "parar, comenzar y continuar", pero no proporcionaron suficiente enfoque para abordar los problemas.

En cambio, ScrumMaster mantuvo una acumulación de impedimentos y el equipo descubrió que funcionaba mejor que las retrospectivas. Los impedimentos pueden estar relacionados con pruebas o herramientas.

También hacen un mapeo del flujo de valor para encontrar el mayor "tiempo de espera" y utilizan los "cinco porqués" de Toyota para comprender qué impedimento es el mayor o qué restricción debe abordarse.

Un ejemplo compartido fue que en un equipo con tres programadores y un evaluador, el mayor problema era un cuello de botella en las pruebas. Rafael preguntó al equipo qué hace el evaluador y escribió esos elementos en una pizarra. Luego preguntó a los programadores cuáles de esas cosas en el tablero no podían hacer. Sólo había un elemento que sentían que no podían manejar. Esto ayudó a los programadores.



Consulte la bibliografía para obtener buenos recursos para el desarrollo lean. prácticas.

Comprenda cómo todos los miembros del equipo de desarrollo, no solo los evaluadores, podrían ser responsables de las tareas de prueba. Este fue un ejercicio muy efectivo.

Enfoques creativos como este ayudan a los nuevos equipos ágiles a afrontar desafíos de pruebas difíciles. Las retrospectivas son un buen ambiente para experimentar.

Utilice las retrospectivas como una oportunidad para plantear cuestiones relacionadas con las pruebas y obtener la Todo el equipo pensando en posibles soluciones. Nos han sorprendido gratamente las ideas innovadoras que surgen de todo un equipo centrado en cómo mejorar su funcionamiento.

CELEBRAR LOS ÉXITOS

Las prácticas de desarrollo ágil tienden a moderar los altibajos que existen en procesos más tradicionales o caóticos. Si su equipo de cascada finalmente logra sacar un lanzamiento después de un ciclo de un año que termina en un estresante ciclo de reparación y prueba de dos meses, todos pueden estar listos para celebrar el evento con una gran fiesta, o simplemente podrían colapsar durante un par de semanas. Los equipos ágiles que lanzan lanzamientos cada dos semanas tienden a permanecer en su codificación normal y Probando el ritmo, comenzando con el siguiente conjunto de historias después de dibujar lo suficiente. Respire para sostener una revisión de iteración y una retrospectiva. Esto es lindo, pero tú Sé lo que dicen sobre todo trabajo y nada de juego.

Asegúrese de que su equipo se tome al menos un poco de tiempo para darse una palmadita en la espalda y reconocer sus logros. Incluso los pequeños éxitos merecen una recompensa. El disfrute es un valor ágil vital y un poco de motivación ayuda a su equipo a continuar en su camino exitoso. Por alguna razón, esto puede resultar difícil de hacer. Muchos ágiles Los equipos tienen problemas para tomarse el tiempo para celebrar el éxito. A veces estás ansioso para comenzar con la siguiente iteración y no se tomen el tiempo para felicitarse por los logros anteriores.

El equipo de Lisa finaliza una iteración cada dos jueves y realiza su retrospectiva, revisión de iteración y lanzamiento al día siguiente. Después de sus reuniones En conclusión, suelen participar en algo que llaman "diversión del viernes". Este a veces consiste en jugar una trivia tonta o un juego de mesa, salir a beber o jugar una partida de minigolf. Tener la oportunidad de relajarse y Reírse mucho tiene un beneficio adicional para la formación de equipos.

Para hitos más importantes, como un gran lanzamiento o lograr un objetivo de cobertura de prueba, el toda la empresa tiene una fiesta para celebrar, trayendo comida de catering o saliendo

a un restaurante el viernes por la tarde. Esta es una buena recompensa y un reconocimiento para todos, tanto en el equipo comercial como en el técnico.

Si el suyo es un nuevo equipo ágil, motívese premiando los pequeños logros. Celebre el creciente número de pruebas unitarias que se pasan en cada compilación.

Ooooh y aaah sobre el gráfico que muestra el incendio real que coincide con el incendio proyectado. Suena una campana cuando las pruebas unitarias rotas en la compilación sean arreglado (está bien, ese puede ser molesto, pero reconócelo de alguna manera).

Celebre también sus éxitos individuales. Felicita a tu compañero de trabajo por completar la primera línea base de prueba de desempeño del proyecto. Dale a tu DBA un Estrella de oro por implementar un sistema de respaldo de producción. Date un regalo para resolver ese difícil problema de automatización de pruebas. Lleva galletas a tu Próxima reunión con los clientes. Reconoce al programador que te dio un Arnés de JavaScript que aceleró las pruebas de algunas validaciones de GUI. Usa tu imaginación.

La caja de zapatos del grito

Nos encanta la idea de celebración que nos dio Megan Sumrell, una entrenadora y entrenadora ágil. Ella compartió esto con una sesión de pruebas ágiles de Open Space en Agile 2007.

Celebrar los logros es algo que me apasiona en los equipos. En un proyecto reciente, implementamos Shout-Out Shoe-box. Tomé una caja de zapatos vieja y la decoré. Luego, simplemente hice un corte en la parte superior de la tapa para que la gente pudiera poner sus agradecimientos en la caja. El box está abierto a todo el equipo durante el transcurso del sprint.

Cada vez que los miembros del equipo quieran dar un “agradecimiento” a otro miembro del equipo, pueden escribirlo en una tarjeta y ponerla en la caja. Pueden ser desde alguien que le ayude con una tarea difícil hasta alguien que vaya más allá de su deber. Si ha distribuido miembros del equipo, anímelos a enviar sus agradecimientos por correo electrónico a su ScrumMaster, quien luego también podrá incluirlos en el cuadro.

Al final de nuestra demostración, alguien del equipo se levanta y lee todas las cartas de la caja. Esto es incluso mejor si tiene otras partes interesadas en su demostración. De esa manera, las personas de su equipo obtienen reconocimiento público por su trabajo frente a una audiencia más amplia. También puedes incluir pequeños obsequios para la gente.

Puede que sea un cliché, pero las pequeñas cosas pueden significar mucho. Shout-Out Shoebox es una excelente manera de reconocer el valor que aportan los diferentes miembros del equipo.

Tomarse el tiempo para celebrar los éxitos le permite a su equipo dar un paso atrás y obtener una nueva perspectiva y renovar su energía para seguir mejorando su producto.

Brinde a los miembros del equipo la oportunidad de apreciar las contribuciones de los demás. No caer en una rutina en la que todos tienen la cabeza gacha trabajando todo el tiempo.

En el desarrollo ágil, tenemos la oportunidad de detenernos y obtener una nueva perspectiva en el final de cada iteración corta. Podemos hacer correcciones menores en el rumbo, decidir pruebe una nueva herramienta de prueba, piense en mejores formas de obtener ejemplos de los clientes o identifique la necesidad de un tipo particular de experiencia en pruebas.

RESUMEN

En este capítulo, analizamos algunas actividades para concluir la iteración o lanzamiento.

La revisión de la iteración es una excelente oportunidad para obtener comentarios y aportaciones del equipo del cliente.

Las retrospectivas son una práctica fundamental para ayudar a su equipo a mejorar.

Observe todas las áreas en las que el equipo puede mejorar, pero concéntrese en una o dos a la vez.

Encuentre una manera de tener en cuenta los elementos de mejora durante la iteración.

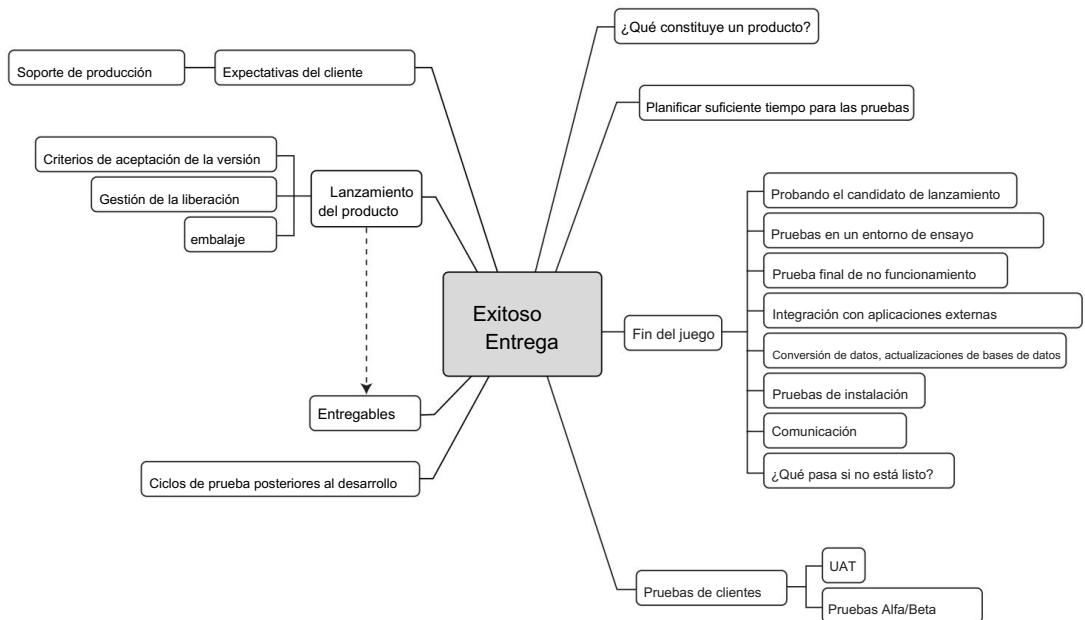
Celebre los éxitos grandes y pequeños y reconozca las contribuciones de diferentes roles y actividades.

Aproveche la oportunidad después de cada iteración para identificar los obstáculos relacionados con las pruebas y pensar en formas de superarlos.

Esta página se dejó en blanco intencionalmente.

Capítulo 20

ENTREGA EXITOSA



En este capítulo, compartimos lo que usted, como evaluador, puede hacer para ayudar a su equipo y a su organización a entregar con éxito un producto de alta calidad. Se pueden utilizar el mismo proceso y herramientas para productos retractilados, soluciones personalizadas o productos de desarrollo interno. Los evaluadores ágiles pueden realizar contribuciones únicas que ayuden tanto al cliente como al equipo de desarrollo a definir y producir el valor que la empresa necesita.

¿QUÉ HACE UN PRODUCTO?

Muchos de los libros sobre desarrollo ágil hablan sobre el ciclo de desarrollo real, pero omiten hablar de lo que constituye un producto y lo que se necesita para entregarlo con éxito. No basta con codificar, probar y decir que es

hecho. Es como comprar algo en una tienda: si hay un gran servicio para llevar con la compra, ¿cuántas más probabilidades hay de que regrese y compre allí? ¿de nuevo?

La historia de Janet

Estaba hablando con mi amigo Ron, que compra y vende monedas. A lo largo de los años, ha desarrollado una muy buena reputación en la industria y ha rechazado a posibles clientes porque está muy ocupado.

Cuando le pregunté cuál era su secreto, dijo: "No es un secreto. Simplemente trabajo con mis clientes para que se sientan cómodos y establezca una relación de confianza con ellos. Al final, tanto yo como mi cliente debemos estar contentos con el trato. Sólo hace falta un cliente descontento para romper mi reputación".

Los equipos ágiles pueden aprender de la experiencia de Ron. Si tratamos a nuestros clientes con respeto y les entregamos un producto con el que estén satisfechos, tendremos una buena relación con ellos, con suerte durante muchos años.

—Janet

Nuestro objetivo es entregar valor al negocio de manera oportuna. no queremos sólo para cumplir con los requisitos pero también para deleitar a nuestros clientes. Antes de volver a arrendar, queremos asegurarnos de que todos los entregables estén listos y pulidos. adecuadamente. Con suerte, comenzó a planificar con anticipación para cumplir no solo con los requisitos del código pero para planificar la capacitación, la documentación y todo que se necesita para fabricar un producto de alto valor.

Ajuste y acabado

Coni Tartaglia, gerente de pruebas de software de Primavera Systems, Inc., explica los entregables de "ajuste y acabado".

Es útil tener una lista de verificación de "Ajuste y acabado". A veces encajar y terminar Los artículos no están listos para ser incluidos en el producto hasta cerca del final. Puede que sea necesario reconstruir partes del producto para incluir elementos como nuevas ilustraciones, licencias o acuerdos legales, firmas digitales para archivos ejecutables, fechas de derechos de autor, marcas comerciales y logotipos.

Es útil ensamblarlos durante la última iteración de desarrollo completo e incorporarlos al producto mientras se ejecutan los ciclos de compilación de integración continua para que no se necesiten compilaciones adicionales más adelante.

El valor empresarial es el objetivo del desarrollo ágil. Esto puede incluir lotes más allá del código de producción. Los equipos deben planificar todos los aspectos de la entrega del producto.

Imagínese en medio de la preparación de su lanzamiento para la producción. Acabas de terminar tu última iteración y estás concluyendo tu última historia. prueba. Su suite de regresión automatizada se ha estado ejecutando en cada nueva compilación, o al menos en cada compilación nocturna. Lo que hagas ahora dependerá de cuán disciplinado haya sido tu proceso. Si has estado manteniendo la “tolerancia cero” para los insectos, probablemente estés en bastante buena forma.

Si eres uno de esos equipos que cree que puedes dejar los errores hasta el final para solucionarlo, probablemente no esté en tan buena forma y es posible que necesite introducir una iteración para “endurecerlo” o corregir errores. No recomendamos esto, pero si su equipo tiene muchos errores pendientes que se han introducido durante el ciclo de desarrollo, es necesario solucionarlos antes de llegar al final. juego. Descubrimos que los equipos nuevos tienden a caer en esta trampa.

Además, hay muchos componentes variados para cualquier versión, algunos en el software, algunos no. Tienes clientes que necesitan instalar y aprender a usar las nuevas características. Piensa en todos esos elementos que son críticos para un lanzamiento exitoso, porque es hora de atar todos esos cabos sueltos y perfeccionar tu producto.

Bob Galen, un entrenador ágil y experto en finales, observa que es posible que el desarrollo ágil no se haya filtrado en todos los rincones de la organización. Él señala: “Los evaluadores ágiles pueden servir como conductor o facilitador cuando se trata de entrega física del software”.

PLANIFICAR TIEMPO SUFICIENTE PARA LAS PRUEBAS

Dado que las pruebas y la codificación son parte de un proceso de desarrollo ágil, recomendamos Preferimos no hacer planes especiales para pasar más tiempo en las pruebas, pero en la vida real puede que necesite algo de tiempo extra.

La mayoría de los equipos acumulan cierta deuda técnica, a pesar de las mejores intenciones, especialmente si trabajan con código heredado. Para mantener la velocidad, su equipo puede Es necesario planificar una iteración de refactorización a intervalos regulares para agregar pruebas, actualizar herramientas y reducir la deuda técnica. El equipo de Lisa realiza un sprint de refactorización aproximadamente cada seis meses. Si bien la empresa no suele recibir ninguna información directa beneficios al final de un sprint de refactorización, los expertos en negocios entienden que estos sprints especiales resulten en una mejor cobertura de pruebas, una base sólida para futuros desarrollo, deuda técnica reducida y una mayor velocidad general del equipo.

Algunos equipos recurren a iteraciones de “refuerzo”, en las que dedican tiempo únicamente a encontrar y corregir errores, y no introducen ninguna funcionalidad nueva. Este es un último recurso para mantener sólida la aplicación y su infraestructura. Es posible que los equipos nuevos necesiten una iteración adicional para completar las tareas de prueba y, de ser así, presupuestan tiempo para ello en el plan de lanzamiento.

Utilice retrospectivas y otras prácticas de mejora de procesos para aprender formas de integrar las pruebas y la codificación para que el código producido en cada iteración esté listo para producción. Cuando se logre ese objetivo, trabaje para garantizar que todos los días esté disponible una versión estable que pueda lanzarse a producción. Los miembros del equipo de Lisa pensaron que este era un objetivo inalcanzable en los días en que luchaban por conseguir una versión estable antes del lanzamiento, pero solo pasaron un par de años antes de que casi todas las versiones fueran dignas de lanzamiento.

Cuando tu construcción sea estable, estarás listo para ingresar al “Fin del juego”.

EL FINAL DEL JUEGO

¿Cuál es el final del juego? Hemos escuchado a personas decir muchas cosas sobre el momento justo antes de la entrega, pero el “final del juego” parece encajar mejor. Es el momento en que el equipo aplica los toques finales al producto. Estás poniendo los puntos en las íes y cruzando las t. Es el último tramo antes de la meta de entrega. No pretende ser un ciclo de corrección de errores, porque para entonces no debería tener ningún error pendiente, pero eso no significa que no tenga uno o dos que corregir.

Es posible que tenga grupos en su organización que no involucró en su planificación anterior. Ahora es el momento de trabajar estrechamente con las personas que administran los entornos de ensayo y producción, los administradores de configuración, los administradores de bases de datos fuera de su equipo y todos los que desempeñan un papel en el traslado del software del desarrollo al ensayo y la producción. Si no estuvo trabajando con ellos temprano esta vez, considere hablar con estas personas durante sus próximas sesiones de planificación de lanzamiento y manténgase en contacto con ellos durante todo el ciclo de desarrollo.

Bob Galen nos cuenta que los evaluadores de su equipo se han asociado con el grupo de operaciones que gestiona los entornos de puesta en escena y producción. Debido a que el grupo de operaciones es remoto, considera que contar con la orientación del equipo ágil es particularmente valioso.

Siempre hay pruebas a nivel de sistema que no se pueden automatizar o que no vale la pena automatizar. La mayoría de las veces, su entorno de puesta en escena es el único lugar

donde puede realizar algunas pruebas de integración a nivel de sistema o carga a nivel de sistema y pruebas de estrés. Le sugerimos que dedique algo de tiempo después del desarrollo para este tipo de tareas de acabado. No codifique hasta el final.

Planifica todo el tiempo que necesites para el final del juego. Janet ha descubierto que el tiempo necesario para el final del juego varía según la madurez del equipo. y el tamaño de la aplicación. Puede ser que solo falte un día para terminar las tareas adicionales, pero puede ser una semana o, a veces, todo un total iteración de dos semanas. El equipo del ejemplo utilizado en el Capítulo 12, "Resumen de los cuadrantes de prueba", programó dos semanas porque era un proceso complejo. sistema que requirió un poco de configuración y prueba del sistema.

La historia de Lisa

Cuando trabajaba en un equipo que desarrollaba aplicaciones para un cliente, teníamos que seguir el cronograma de lanzamiento del cliente. Las pruebas con otras partes del sistema más grande solo fueron posibles durante ciertos períodos de dos semanas, cada seis u ocho semanas. Nuestro equipo completó dos o tres iteraciones, terminando todas las historias de cada una como si estuvieran lanzando cada iteración.

Luego ingresamos a una ventana de prueba donde pudimos coordinar las pruebas del sistema con otros equipos de desarrollo, ayudar al cliente con UAT y planificar el lanzamiento real. Esto constituyó nuestro final.

—Lisa

Si tiene una organización grande, es posible que tenga diez o quince equipos que desarrollen software para productos individuales o para áreas separadas de funcionalidad para la misma aplicación. Es posible que todas estas áreas o productos deban lanzarse juntos, por lo que es necesario un final integrado. Esto no significa que usted Dejar la integración para el final. Coordinación con los otros equipos. Será fundamental durante todo el ciclo de desarrollo y, si tiene un sistema de integración de prueba, le recomendamos que se asegure de haber intentado integrarlo mucho antes del final del juego.

También es posible que tengas consideraciones más allá de tu equipo, por ejemplo, trabajar con software entregado por equipos externos a nivel empresarial.

Utilice este tiempo del final del juego para realizar algunas pruebas exploratorias finales. Da un paso atrás y Mire todo el sistema y realice algunos escenarios de un extremo a otro. Tales pruebas confirmar que la aplicación está funcionando correctamente, darle mayor confianza en el producto y proporcionar información para la próxima iteración o lanzamiento.

Probando el candidato de lanzamiento

Recomendamos que las pruebas de regresión automatizadas se realicen contra cada candidato de liberación. Si sigue nuestra recomendación de ejecutar automáticamente pruebas de regresión continuamente en cada nueva compilación, o al menos diariamente, ya hecho esto. Si algunas de tus pruebas de regresión son manuales, necesitarás planificar el tiempo para esos o podrían no terminarse. Una evaluación de riesgos basada en cambios hecho en cada compilación determinará qué pruebas deben ejecutarse si hay más más de un candidato de liberación.

Prueba en un entorno de ensayo

Ya sea que esté utilizando procesos de desarrollo tradicionales o ágiles, un entorno de preparación que imite la producción es vital para las pruebas finales antes del lanzamiento, así como para las pruebas finales antes del lanzamiento. en cuanto a probar el proceso de lanzamiento en sí. Como parte del objetivo final, su aplicación debe implementarse en el escenario de la misma manera que lo haría en producción. o como lo harían sus clientes en sus entornos. En muchas organizaciones que Janet ha visto, el entorno de puesta en escena suele ser compartido entre múltiples proyectos, y la implementación debe programarse como parte de la planificación del lanzamiento. Considere de antemano cómo manejar las dependencias, integrándose con otros equipos que utilizan el entorno de prueba y trabajan con terceros externos fiestas. Puede parecer una planificación de pruebas “tradicional”, pero es posible que esté lidiando con equipos que no han adoptado el desarrollo ágil.

Aunque la metodología ágil promueve la integración continua, a menudo es difícil integrarla con productos de terceros u otras aplicaciones fuera del ámbito de su proyecto. control. Los entornos de prueba pueden tener mejores controles para que las aplicaciones externas puedan conectarse y tener acceso a entornos de prueba de terceros. Los entornos de prueba también se pueden utilizar para pruebas de carga y rendimiento, simulacros implementaciones, pruebas de conmutación por error y pruebas de regresión manual y pruebas funcionales exploratorias. Siempre existen diferencias de configuración entre entornos, por lo que su entorno de prueba es un buen lugar para probarlas.

Pruebas finales no funcionales

Las pruebas de carga deben programarse a lo largo del proyecto en piezas específicas de la aplicación que estás desarrollando. Si su entorno de puesta en escena es alto demanda, es posible que no pueda realizar pruebas de carga completas del sistema hasta el final del juego.

En este momento, debería poder realizar pruebas de confiabilidad de larga duración en todos la funcionalidad del producto. Compruebe si hay fallos y degradación del rendimiento. con carga normal. Cuando se haga en el momento del lanzamiento, debería ser sólo una confirmación final.

Las pruebas de recuperación y tolerancia a fallos se realizan mejor en su entorno de prueba además, porque los entornos de prueba normalmente no cuentan con la configuración necesaria. Para estos mismos motivos, es posible que solo puedas probar ciertos aspectos de seguridad. Un ejemplo es https, una conexión http segura mediante conexiones seguras cifradas. Algunas organizaciones pueden optar por tener los certificados necesarios en únicamente su entorno de puesta en escena. Otros ejemplos son la agrupación en clústeres o la replicación de datos. Asegúrese de involucrar a todas las partes que deben participar en esta prueba.

Integración con aplicaciones externas

Su equipo puede ser ágil, pero otros equipos de productos de su organización, o Los terceros con los que trabaja su equipo pueden no serlo.

La historia de Janet

En una organización con la que trabajé, el socio externo que aprobó las tarjetas de crédito tenía una cuenta de prueba que podía usarse, pero solo era accesible desde el entorno de prueba.

Para realizar pruebas durante el desarrollo, se crearon resguardos de prueba para devolver resultados específicos según el número de tarjeta de crédito utilizado. Sin embargo, esto no fue suficiente porque el tercero a veces cambiaba la funcionalidad por su parte que no estábamos consciente de. Las pruebas con terceros reales fueron fundamentales para el éxito del proyecto y son una parte clave del objetivo final.

—Janet

Coordinar con mucha antelación con otros equipos de productos o socios externos. que tienen productos que necesitan integrarse con su producto. Si usted tiene identificado estos riesgos tempranamente y realizó tantas pruebas iniciales como fue posible, el Las pruebas realizadas durante el final del juego deben ser solo una verificación final. Sin embargo, Siempre hay sorpresas de último momento, por lo que es posible que tengas que estar preparado para realizar cambios en su aplicación.

Herramientas como simuladores y objetos simulados utilizados para realizar pruebas durante el desarrollo. puede ayudar a aliviar algunos de los riesgos, pero cuanto antes pueda realizar pruebas con dispositivos externos aplicaciones, menor es el riesgo.

Conversión de datos y actualizaciones de bases de datos

Mientras desarrollamos una aplicación, cambiamos campos, agregamos columnas en la base de datos eliminamos las obsoletas. Los diferentes equipos abordan esto de diferentes maneras. Algunos equipos recrean la base de datos con cada nueva compilación. Esto funciona para nuevos aplicaciones, porque no existen datos existentes. Sin embargo, después de una aplicación existe en producción y tiene datos asociados, este enfoque no funcionará.

Una aplicación debe considerar los datos que forman parte del producto. Al igual que con En gran medida del desarrollo ágil, se requiere un esfuerzo conjunto de los expertos en bases de datos, programadores y evaluadores del equipo para garantizar la publicación exitosa de los cambios en la base de datos. Janet ha visto un par de tácticas diferentes para manejar datos. conversión y compatibilidad con versiones anteriores. Los scripts de bases de datos se pueden crear mediante los desarrolladores o administradores de bases de datos a medida que el equipo realiza cambios. Estos Los scripts se convierten en parte de la compilación y se prueban continuamente. Otra opción es para que el equipo ejecute "diffs" en la base de datos después de todos los cambios en la base de datos ha sido hecho.

Si es un evaluador, solicite ayuda al administrador/desarrollador de su base de datos. El equipo se asegura de que los esquemas se mantengan coherentes entre los entornos de producción, pruebas y preparación. Encuentre una manera de garantizar que todos los cambios Lo realizado en los entornos de prueba se realizará en la puesta en escena y la producción. entornos durante el lanzamiento. Mantenga los esquemas coincidentes (excepto el nuevos cambios aún en desarrollo) en términos de nombres de columnas, activadores, restricciones, índices y otros componentes. La misma disciplina se aplica a La codificación y las pruebas también deben aplicarse al desarrollo y desarrollo de bases de datos. mantenimiento.

La historia de Lisa

Recientemente, se lanzó un error a producción porque a algunos de los esquemas de prueba, incluido el utilizado por las pruebas de regresión, les faltaba una restricción. Sin la restricción implementada, el código no falló. Esto desencadenó un esfuerzo para garantizar que se ejecuten exactamente los mismos scripts de actualización en cada esquema para realizar cambios para una versión determinada.

Resultó que los diferentes esquemas de prueba tenían pequeñas diferencias, como columnas antiguas que aún permanecían en algunos o columnas en diferente orden en diferentes esquemas, por lo que no era posible ejecutar el mismo script en todos los entornos. Nuestro administrador de base de datos dirigió un gran esfuerzo para recrear todos los esquemas de prueba para que fueran perfectamente compatibles con la producción. Crea un script en cada iteración con todos los cambios necesarios en la base de datos y ejecuta ese mismo script en el entorno de ensayo y producción cuando lo lanzamos. Esto parece simple, pero es fácil pasar por alto diferencias sutiles cuando te concentras en ofrecer nuevas funciones.

—Lisa

Automatizar las migraciones de datos mejora su capacidad para probarlas y reduce la posibilidad de error humano. Las herramientas de bases de datos nativas como SQL, procedimientos almacenados, herramientas de importación de datos como SQL*Loader y bcp, scripts de shell y archivos de comandos de Windows se pueden utilizar para la automatización porque se pueden clonar. y alterarse fácilmente.

No importa cómo se crean o crean los scripts de conversión y actualización de la base de datos. mantenidos, es necesario probarlos. Una de las mejores maneras de garantizar que todos los Los cambios se han capturado en los scripts de actualización es utilizar los datos del cliente. si está disponible. Los clientes tienen la costumbre de utilizar la aplicación de forma extraña y maneras maravillosas, y los datos no siempre son tan limpios como nos gustaría. Si el El equipo de desarrollo limpia la base de datos y pone restricciones adicionales a un columna, la aplicación en el sitio del cliente podría explotar tan pronto como La consulta toca un dato que no coincide con las nuevas restricciones. Tú Debe asegurarse de que los cambios que haya realizado sigan siendo compatibles con los datos existentes.

La historia de Lisa

Mi equipo utiliza el entorno de prueba para probar los scripts de actualización de la base de datos. Después de ejecutar los scripts, realizamos pruebas manuales para verificar que todos los cambios y conversiones de datos se hayan completado correctamente. Algunos de nuestros scripts de prueba de GUI cubren un subconjunto de escenarios de regresión. Esto nos da confianza a la hora de lanzar a producción, donde nuestra capacidad de realizar pruebas es más limitada.

—Lisa

Al planificar una conversión de datos, piense en la limpieza de datos como parte del Estrategia de mitigación. Tiene la oportunidad de tomar los datos que se ingresaron de algunas de las formas "extrañas y maravillosas" que mencionamos antes y masajéelo o manipúlelo para que se ajuste a las nuevas limitaciones. Este tipo de El trabajo puede llevar mucho tiempo, pero a menudo vale la pena en términos de mantener la integridad de los datos.

No todo el mundo puede hacer una simulación suficientemente buena de los datos de producción en el entorno de puesta en escena. Si los datos de un cliente no están disponibles, una estrategia de mitigación es tener una UAT en el sitio del cliente. Otra forma de mitigar el riesgo es Intento evitar actualizaciones a gran escala y publicarlas en etapas más pequeñas. Desarrollar nuevas Funcionalidad en paralelo con la funcionalidad anterior y uso de una propiedad del sistema. para "encender" uno u otro. La antigua funcionalidad puede seguir funcionando en producción hasta que se complete la nueva funcionalidad. Mientras tanto, las pruebas pueden realizarse en el nuevo código en cada iteración. Se pueden crear nuevas columnas y tablas. agregado a las tablas de producción sin afectar el código antiguo para que los datos Se minimiza la migración o conversión para la versión final.

Pruebas de instalación

Las organizaciones suelen tener un equipo separado que implementa la producción o crea el conjunto de productos. Estos miembros del equipo deberían tener la oportunidad de

practique la implementación exactamente como lo harían para la producción. Si usan el despliegue a la puesta en escena como su campo de pruebas, pueden resolver cualquiera de los problemas mucho antes de que se entreguen al cliente.

Probar instalaciones de productos también puede significar probar varias instalaciones de productos reactualizados a diferentes sistemas operativos o hardware. Cómo
¿Se comporta el producto? ¿Hace lo que se espera? ¿Cuánto tiempo deberá estar inactivo el sistema para su instalación? ¿Podemos implementar sin interrumpir el servicio? ¿Podemos hacer que la experiencia del usuario sea lo más placentera posible?

La historia de Janet

Hace un tiempo tuve una experiencia que no fue tan agradable y me llevó a desear que alguien hubiera probado y solucionado el problema antes de que yo lo encontrara. Compré una computadora portátil nueva y quería transferir mi licencia para una de mis aplicaciones a la nueva computadora. Venía con una versión de prueba de la misma aplicación, por lo que la transferencia debería haber sido fácil, pero la nueva PC no reconoció la clave del producto y seguía diciendo que no era válida. Llamé al servicio de soporte y después de un poco de diagnóstico, me informaron que se consideraban productos diferentes, por lo que la clave no funcionaba.

Dos horas más de soporte y el problema se solucionó. Se tuvo que eliminar la versión de prueba, reinstalar una versión anterior, volver a ingresar la clave y cargar todas las actualizaciones desde la compra original. Cuánto más fácil hubiera sido para el equipo de desarrollo probar ese escenario y ofrecer al cliente un mensaje informativo que dijera: "La versión de prueba no es compatible con su clave de producto". Un mensaje como ese me habría permitido descubrir el problema y resolverlo yo mismo en lugar de quitarle el tiempo a la persona de soporte.

—Janet

Tómese el tiempo que necesite para determinar cuáles son sus requisitos para las pruebas. instalación. Al final valdrá la pena si satisface a tus clientes.

Comunicación

La comunicación constante entre los diferentes miembros del equipo de desarrollo es siempre importante, pero es especialmente crítico a medida que finalizamos el lanzamiento. Tener reuniones de pie adicionales, si es necesario, para asegurarse de que todo esté listo para el liberar. Escriba tarjetas para las tareas de lanzamiento si existe alguna posibilidad de que se pueda realizar algún paso. olvidado.

La historia de Lisa

Mi equipo lanza después de cada iteración. Por lo general, hacemos una reunión rápida la última tarde del sprint para tocar la base e identificar cualquier cable suelto. Antes de que el equipo tuviera mucha práctica con los lanzamientos, escribimos tarjetas de tareas de lanzamiento como "ejecutar script de actualización de la base de datos en preparación" y "verificar las actualizaciones de la base de datos en producción".

Con más experiencia en la implementación, ya no necesitamos esas tarjetas a menos que tengamos un nuevo miembro del equipo que pueda necesitar un recordatorio adicional. Sin embargo, nunca está de más tener tarjetas para tareas de lanzamiento.

—Lisa

Recordatorios de tareas, ya sea que estén en un plan de implementación completo o simplemente escritas en tarjetas de tareas como lo hace el equipo de Lisa, a menudo son necesarias. En implementaciones simples, una pizarra funciona bien.

¿Qué pasa si no está listo?

Al realizar un seguimiento constante del progreso en muchas formas, como compilaciones, pruebas de regresión suites, guiones gráficos y gráficos de avance, un equipo generalmente sabe de antemano cuándo tiene problemas en un lanzamiento. Hay tiempo para dejar historias y leer solo. Aún así, pueden ocurrir desastres de último momento. ¿Qué pasa si la máquina de construcción se estropea? ¿El último día de la iteración? ¿Qué pasa si la base de datos de prueba falla y el final ¿No se pueden completar las pruebas? ¿Qué pasa si no se detecta un error espectacular hasta la prueba funcional final?

Recomendamos encarecidamente no agregar días adicionales a una iteración, porque comer en la siguiente iteración o desarrollo de lanzamiento. Un equipo experimentado Puede ser lo suficientemente flexible para hacer esto, pero puede descarrilar un nuevo equipo. Aun así, tiempos desesperados exigen medidas desesperadas. Si publicas cada dos semanas, Es posible que simplemente pueda omitir la versión real y presupuestar el tiempo para la siguiente iteración para corregir los problemas y terminar, y liberar en la siguiente cita agendada. Si las tareas de prueba se posponen o ignoran y la liberación Adelante, plantea este problema al equipo. ¿Cambiaron las necesidades de pruebas? ¿O el equipo se está arriesgando y sacrificando la calidad para cumplir con una fecha límite? El equipo debe recortar el alcance del lanzamiento si la fecha de entrega está fijada y está en peligro.

Si su ciclo de lanzamiento es más largo, más de tres meses, debe saberlo en con antelación si su liberación está en peligro. Probablemente hayas planeado un final juego de al menos dos semanas, que será sólo para la validación final. Cuando usted Si tiene un ciclo de lanzamiento más largo, tendrá más tiempo para determinar lo que desea. debería hacer, ya sea eliminar la funcionalidad o cambiar el horario.

Si su organización requiere que cierta funcionalidad se publique en un formato fijo El día y los fallos de última hora amenazan el lanzamiento, evalúa tus alternativas. Vea si puede continuar con el mismo ciclo de desarrollo pero retrasar el lanzamiento. sí mismo durante un día o una semana. Quizás el fragmento de código ofensivo pueda respaldarse temporalmente y un parche hecho más tarde. Los clientes tienen la última palabra. en lo que funcionará para el negocio.

La historia de Lisa

En las raras ocasiones en que nuestro equipo se ha enfrentado al problema de los obstáculos de último minuto, hemos utilizado diferentes enfoques según la situación. Si hay

No hay nada crítico que deba publicarse en este momento, a veces nos saltamos el lanzamiento y publicamos dos iteraciones el siguiente día de lanzamiento. Si es necesario introducir algo crítico, retrasamos el lanzamiento uno o dos días. A veces podemos seguir adelante y liberar lo que tenemos y lanzar un parche al día siguiente. En una ocasión, decidimos tener una iteración especial de una semana para corregir los problemas, lanzarlos y luego volver al cronograma normal de iteraciones de dos semanas.

Después de más de cuatro años de practicar el desarrollo ágil, tenemos una versión estable casi el 100 % del tiempo y nos sentimos seguros de poder lanzarla cuando sea necesario. Necesitábamos mucha disciplina y mejora continua en nuestro proceso para sentir que un enfoque más flexible podría funcionar para nosotros. Es

También es bueno poder lanzar una valiosa funcionalidad temprano, si es posible. Lo que hemos trabajado duro para evitar es caer en una espiral mortal en la que nunca podremos lanzar a tiempo y siempre estamos tratando de ponernos al día.

No te castigues si no puedes liberarlo a tiempo. Tu equipo está haciendo lo mejor que puede. Dedique tiempo a analizar por qué se retrasó o se comprometió demasiado y tome medidas para evitar que vuelva a suceder.

—Lisa

Trabaje para evitar una situación de "no ir" con buena planificación, estrecha colaboración y conducir la codificación con pruebas y realizar pruebas a medida que codifica. Si su seguimiento muestra el la liberación podría estar en peligro, elimine la funcionalidad que no se puede finalizar, si posible. Si sucede algo malo e inesperado, no entre en pánico. Involucrar al todo el equipo y el equipo del cliente, y realizar una lluvia de ideas sobre la mejor solución.

PRUEBAS DEL CLIENTE

Hay un par de formas diferentes de involucrar a sus clientes para obtener su aprobación o retroalimentación. Las pruebas de aceptación del usuario pueden ser bastante formales, con aprobaciones del negocio. Significa aceptación de una liberación. Alfa o La prueba beta es una forma de obtener comentarios sobre un producto que desea lanzar, pero que no está del todo listo.

UAT

Las pruebas de aceptación del usuario (UAT) son importantes en aplicaciones personalizadas de gran tamaño así como aplicaciones internas. Lo realizan todos los departamentos comerciales afectados para verificar la usabilidad del sistema y confirmar la funcionalidad comercial existente y nueva (énfasis en la nueva) del sistema. Tus clientes son los

aquellos que tienen que vivir con la aplicación, por lo que deben asegurarse de que trabaja en su sistema y con sus datos.

En capítulos anteriores hemos hablado a menudo de cómo involucrar a los clientes. Temprano, pero en esos momentos, las pruebas se realizan en características específicas en desarrollo. La UAT generalmente se realiza después de que el equipo decide que la calidad es lo suficientemente buena para liberar. A veces, sin embargo, el cronograma dicta el ciclo de lanzamiento. Si ese es el caso, intente mover el ciclo UAT hacia arriba para que se ejecute en paralelo con su juego final. La aplicación debe ser lo suficientemente estable para que su equipo pueda implementarla en el sistema de prueba del cliente al mismo tiempo que lo implementan en la etapa de preparación.

La historia de Janet

En un equipo al que me uní, los clientes eran muy exigentes. De hecho, el más exigente que jamás había visto. Siempre pedían una semana completa de UAT solo para asegurarse de tener tiempo para probarlo todo. Habían preparado casos de prueba y los comprobaron todos, incluido todo el contenido, tanto en inglés como en francés. Los errores más importantes incluyeron errores ortográficos, como la falta de acento en el contenido francés. Con el tiempo, a medida que ganaron más confianza en nuestros lanzamientos y encontraron cada vez menos errores, relajaron sus demandas pero aún querían una semana, en caso de que no pudieran hacerlo de inmediato. Su grupo empresarial estaba muy ocupado.

Llegó un lanzamiento que empujó la línea de tiempo. Nos retuvieron hasta la fecha de lanzamiento, pero no pudimos incorporar todas las funciones y dejamos dos semanas para el final del juego. Hablamos con los usuarios comerciales y decidimos reducir el final del juego a una semana; los usuarios empresariales realizarían su UAT mientras el equipo del proyecto terminaba las pruebas y la limpieza del sistema. La única razón por la que pudimos hacer esto fue por la confianza que el cliente tenía en nuestro equipo y la consistencia.

Tenencia de nuestros lanzamientos.

La buena noticia fue que, una vez más, la UAT no encontró problemas que no pudieran esperar hasta la próxima versión.

—Janet

La Figura 20-1 muestra una línea de tiempo de ejemplo con una UAT normal al final del ciclo de liberación. El equipo comienza a trabajar en la próxima versión, planifica la versión y comienza la primera iteración con todos los miembros del equipo listos para comenzar.

Trabajar con los clientes para que comprendan el proceso, su función y lo que se espera de ellos. Si la UAT no es fluida, entonces hay posibilidades de que será necesario un alto nivel de apoyo. Un equipo de pruebas de clientes experimentado puede Tienen casos de prueba definidos, pero la mayoría de las veces sus pruebas son ad hoc. Los clientes pueden Abordar sus pruebas como si estuvieran haciendo su trabajo diario, pero probablemente centrarse en la nueva funcionalidad. Esta es una oportunidad para observar cómo las personas

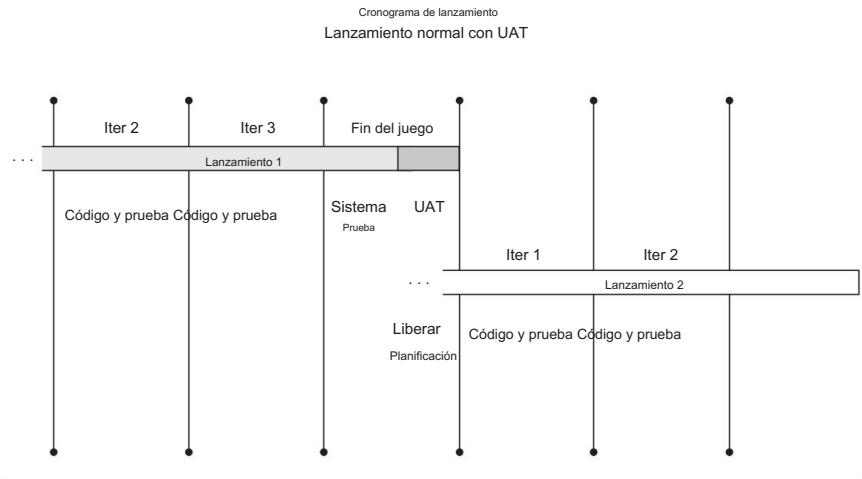


Figura 20-1 Cronograma de lanzamiento con UAT

utilizar el sistema y obtener comentarios de ellos sobre lo que funciona bien y lo que las mejoras les ayudarían.

Los evaluadores pueden brindar soporte a los clientes que realizan la UAT revisando las pruebas ejecutadas y los defectos registrados, y realizando un seguimiento de los defectos hasta su finalización. A ambos nos ha resultado útil ofrecer a los clientes que participan en la realización de UAT con un informe de todas las pruebas realizadas durante el desarrollo, junto con el resultados. Eso les ayuda a decidir dónde centrar sus propias pruebas.

Pruebas Alfa/Beta

Si es una organización que distribuye software a una gran base de clientes, es posible que no tenga una UAT formal. Es mucho más probable que incorpores pruebas alfa o beta. Su equipo querrá recibir comentarios sobre nuevas funciones de sus clientes reales, y este es un mecanismo para hacerlo. La prueba alfa es Distribución temprana de nuevas versiones de software. Porque es probable que haya algunos errores importantes, debes elegir a tus clientes sabiamente. Si eliges este método de comentarios de los clientes, asegúrese de que sus clientes comprendan sus role. Las pruebas alfa sirven para obtener comentarios sobre las funciones, no para informar errores.

Las pruebas beta están más cerca de la UAT. Se espera que el lanzamiento sea bastante estable y realmente se puede utilizar. Puede que no esté "listo para el horario de máxima audiencia" para la mayoría de los clientes, pero muchos clientes pueden sentir que vale la pena correr el riesgo por las nuevas funciones. personalizado

Los usuarios deben comprender que no se trata de una autorización formal y que usted pidiéndoles que prueben su producto e informen errores.

Como evaluador, es importante comprender cómo ven los clientes el producto, porque puede afectar la forma en que realiza la prueba. Las pruebas alfa y beta pueden ser las únicas tiempo que puede interactuar con los usuarios finales, así que aproveche la oportunidad de saber qué tan bien el producto satisface sus necesidades.

CICLOS DE PRUEBAS POST-DESARROLLO

Si trabaja en una organización grande o está desarrollando un componente de una organización grande, sistema complejo, es posible que necesite reservar tiempo para realizar pruebas una vez finalizado el desarrollo completo. A veces, las pruebas UAT, o la coordinación de las pruebas, no son tan sencillas como podría ser, por lo que la línea de tiempo se alarga. Entornos de prueba que incluyen pruebas.

Es posible que las versiones de todos los sistemas de producción solo estén disponibles para pequeñas empresas programadas. ventanas de tiempo. Es posible que necesite coordinar sesiones de prueba con equipos que trabajan en otras aplicaciones que interactúan con la suya. Cualquiera sea el motivo, necesitas tiempo de prueba adicional que no incluye a todo el equipo de desarrollo.

La historia de Lisa

Trabajé en un equipo desarrollando componentes de aplicaciones internas y externas para un gran cliente de telecomunicaciones. Sólo pudimos acceder al entorno de prueba completo a intervalos programados. Los lanzamientos también estuvieron muy programados.

El equipo de desarrollo trabajó en iteraciones de dos semanas. Podría lanzarse al entorno de prueba solo después de cada tercera iteración. En ese momento, hubo un ciclo de prueba de aceptación del usuario e integración del sistema de dos semanas, seguido del lanzamiento.

Alguien de mi equipo necesitaba dirigir la fase de pruebas post-desarrollo. Mientras tanto, los desarrolladores estaban comenzando una nueva versión con nuevas funciones y necesitaban un evaluador que los ayudara con ese esfuerzo.

El equipo tuvo que hacer un esfuerzo especial para asegurarse de que alguien en el rol de probador siguiera cada lanzamiento de principio a fin. Por ejemplo, trabajé de principio a fin. en la versión 1. Shauna asumió el rol de tester cuando el equipo comenzó a trabajar en la primera iteración de la versión 2, mientras yo coordinaba las pruebas del sistema y la UAT en la versión 1. Shauna permaneció como tester principal para la versión 2, mientras yo asumía que rol para la versión 3.

—Lisa

La Figura 20-2 muestra un cronograma de ejemplo en el que se amplió la UAT. Este podría suceder por varias razones y es posible que el problema no siempre se resuelva. UAT. La mayor parte del equipo está listo para empezar a trabajar en la próxima versión, pero a menudo

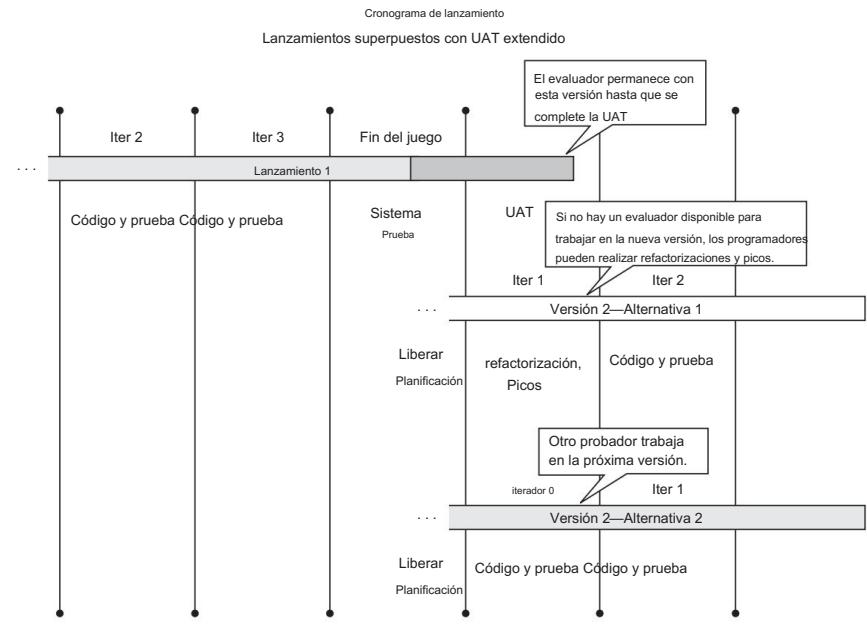


Figura 20-2 Cronograma de lanzamiento: enfoque alternativo con UAT extendida

un evaluador todavía está trabajando con los clientes, completando las pruebas finales. A veces también participará un programador. Hay un par de opciones. Si el equipo es lo suficientemente grande, probablemente pueda iniciar la siguiente versión mientras un par de miembros del equipo trabajan con la versión existente (Versión 2: Alternativa 2 en la Figura 20-2). Si tiene un equipo pequeño, es posible que deba considerar una Iteración 0 con programadores realizando refactorizaciones o picos (experimentos) en nuevas funcionalidades para que el evaluador que trabaja con el cliente no se quede atrás (Versión 2: Alternativa 1 en la Figura 20-2).

Sea creativo al enfrentar las circunstancias impuestas a su equipo por las realidades de su proyecto. Si bien los planes rara vez funcionan como se espera, planificar con anticipación aún puede ayudarlo a asegurarse de que cuenten con las personas adecuadas para entregar el producto de manera oportuna.

ENTREGABLES

En la primera sección de este capítulo hablamos sobre lo que constituye un producto. La respuesta a esto dependerá en realidad de la audiencia: ¿quién acepta el producto y cuáles son sus expectativas?

Si sus clientes necesitan cumplir con los requisitos de cumplimiento de SOX (Sarbanes-Oxley), habrá ciertos entregables que serán necesarios. Por ejemplo, un cliente con el que Janet trabajó consideró que los resultados de las pruebas deberían documentarse minuciosamente.

e hizo de los resultados de las pruebas uno de sus puntos de medición de cumplimiento SOX, mientras que un Diferentes clientes no midieron los resultados de las pruebas en absoluto. Trabajar con cumplimiento personal de auditoría para identificar las necesidades de informes al comenzar un proyecto.

¿Cuánta documentación es suficiente? Janet siempre hace dos preguntas antes respondiendo a esa pregunta: "¿Para quién es?" y "¿Para qué lo usan?" Si no hay respuestas adecuadas a esas preguntas, entonces considere si la La documentación es realmente necesaria.

Los entregables no siempre son para el cliente final y no siempre están en la forma de software. Hay muchos clientes internos, como los miembros del equipo de soporte de producción. ¿Qué necesitarán para facilitar su trabajo?

Los diagramas de flujo de trabajo pueden ayudarlos a comprender las nuevas funciones. Lo harán Probablemente le gustaría saber si existen soluciones alternativas para que puedan ayudar. clientes a través de problemas.

A Janet a menudo le preguntan sobre la cobertura de prueba del código, generalmente por parte de la gerencia. ¿Qué parte de la aplicación se está probando mediante pruebas unitarias o regresión? pruebas? El problema es que el número en sí mismo es sólo un número, y hay Hay tantas razones por las que puede ser alto o bajo. Además, la cobertura del código no indica informarle sobre funciones que podrían haberse pasado por alto y para las cuales aún no existe ningún código. La audiencia de un producto como la cobertura del código no debe ser la dirección, sino el equipo mismo. Se puede utilizar para ver qué áreas del código no están siendo probado.

La formación también podría considerarse un producto entregable. Muchas aplicaciones requieren Sesiones de formación personalizadas para clientes. Es posible que otros solo necesiten estar en línea ayuda o un manual de usuario. La formación podría determinar el éxito de su producto, entonces es importante considerarlo. El equipo de Lisa a menudo escribe tarjetas de tareas para un evaluador o el propietario del producto para asegurarse de que los materiales y sesiones de capacitación sean organizado. Algunas personas pueden sentir que el entrenamiento no es trabajo de los evaluadores ni de nadie más. en el equipo de desarrollo. Sin embargo, los equipos ágiles pretenden trabajar lo más estrechamente posible posible con el negocio. Los evaluadores a menudo tienen la experiencia en el dominio para poder para al menos identificar la capacitación que podría ser necesaria para funciones nuevas o actualizadas. Incluso si la capacitación no es responsabilidad del evaluador, puede plantear el problema si el la empresa no está planificando sesiones de formación.

Muchos equipos ágiles tienen redactores técnicos como parte del equipo que escribe en línea. ayuda o formularios electrónicos de documentación. Incluso se incluye una aplicación

videos de capacitación para ayudar a comenzar, y diferentes miembros del equipo fueron Los entrenadores. Es responsabilidad del equipo crear un producto exitoso.

Entregables que no son software

Coni Tartaglia, gerente de pruebas de software de Primavera Systems, Inc., reflexiona sobre lo que ha funcionado para su equipo al entregar elementos que no son código pero que son necesarios para un lanzamiento exitoso.

Aparte del software, ¿qué ofrece el equipo? Es útil tener una conversación con personas ajenas al equipo de desarrollo que puedan estar interesadas en esta pregunta. Grupos como Legal, Marketing de Productos, Capacitación y Atención al Cliente querrán contribuir a la lista de entregables.

Una vez que se llega a un acuerdo sobre lo que se entrega, puede comenzar el ensamblaje de los componentes y la función de Gestión de lanzamientos puede proporcionar confirmación de la entrega mediante la ejecución de una lista de verificación de lanzamiento. Si la versión es una actualización de un producto existente, los evaluadores pueden verificar los entregables de versiones anteriores para asegurarse de que no quede nada crítico fuera del paquete de actualización. Los entregables pueden incluir avisos legales, documentación, traducciones y software de terceros que se proporcionan como cortesía a los clientes.

Los equipos ágiles están aportando valor, no sólo software. Trabajamos junto con el equipo del cliente para mejorar todos los aspectos del producto.

No existen reglas estrictas sobre lo que se debe entregar con el producto. Piense en los entregables como algo que agrega valor a su producto. ¿Quién debería ser el destinatario del entregable y cuándo tiene más sentido entregarlo?

LANZAMIENTO DEL PRODUCTO

Cuando hablamos de lanzar el producto, nos referimos a ponerlo a disposición de al cliente en cualquier formato que adopte. Su organización podría tener un sitio web que se actualiza o una aplicación personalizada que se entrega a un pocos clientes grandes. Tal vez el producto esté reenviado y entregado a millones de PC en todo el mundo o descargados de Internet.

Criterios de aceptación de la versión

¿Cómo sabes cuando has terminado? Los criterios de aceptación son tradicionales. forma de definir cuándo aceptar el producto. Los criterios de desempeño pueden tener

ser reunidos. Los capturamos para cada historia al comienzo de cada iteración, y También podemos especificarlos para conjuntos de características más grandes cuando comenzamos un tema o épico. Los clientes pueden establecer criterios de calidad como un cierto porcentaje de código cubiertos por pruebas automatizadas, o que ciertas pruebas deben pasar. Líneas de pedido como tener cero errores críticos, o cero errores con impacto grave en el sistema, son a menudo forma parte de los criterios de liberación. Los clientes deben decidir cómo saber cuándo hay suficiente valor en el producto. Los evaluadores pueden ayudarlos a definir publicar criterios que logren sus objetivos.

Los equipos ágiles trabajan para alcanzar el espíritu de los objetivos de calidad, no sólo la letra. No reducen la gravedad de los errores a media para poder decir que logró el criterio de ausencia de errores de alta gravedad. En cambio, con frecuencia miran observar las tendencias de errores y pensar en formas de garantizar que no se produzcan errores de alta gravedad en producción.

Su nivel de calidad debe negociarse con su cliente por adelantado para que no hay sorpresas desagradables. La aceptación pone a prueba a tu equipo y a tu Los clientes definidos, utilizando ejemplos reales, deben servir como hitos para progreso hacia la liberación. Si su cliente tiene una tolerancia muy baja a los errores, y el 100% de esas pruebas de aceptación deben pasar, su velocidad de iteración debería tener eso en cuenta. Si las nuevas características son más importantes que correcciones de errores, bueno, tal vez envíe errores.

Una historia de "cocción" en varios niveles

Bob Galen, entrenador ágil y autor de sus Finales de software , explica como equipos, define los criterios de aceptación de la versión y evalúa si han sido cumplido.

Me uní a varios equipos ágiles nuevos en los últimos años y he visto un patrón común dentro de esos equipos. Mi equipo actual hace un trabajo maravilloso al establecer criterios a nivel de historia de usuario o característica, básicamente definir criterios de aceptación. Hemos trabajado duro para perfeccionar nuestros criterios de aceptación. Inicialmente fueron desarrollados a partir de los Product Owners' y a menudo eran bastante ambiguos y mal definidos. Los evaluadores decidieron que realmente podían ayudar a los clientes a perfeccionar sus pruebas para que fueran mucho más relevantes, claras y comprobables. Esa colaboración resultó ser una victoria significativa a nivel de historia y los propietarios de producto realmente valoraron el compromiso y la ayuda.

Muy a menudo, los evaluadores también automatizarían las pruebas de aceptación de la historia del usuario, ejecutándolas durante cada sprint pero también demostrando la aceptación general durante la revisión del sprint.

Sin embargo, un problema que tuvimos fue lograr que este mismo nivel de claridad para que el "listo" a nivel de historia se extendiera más allá de las historias individuales.

Descubrimos que a menudo, cuando nos acercábamos al final de un Sprint o al final del juego de un lanzamiento, teníamos expectativas abiertas de lo que se suponía que el equipo debía lograr dentro del sprint. Por ejemplo, entregaríamos historias que fueron probadas exhaustivamente "en lo pequeño"; es decir, se probó la funcionalidad de esas historias, pero no se integraron en nuestro entorno de ensayo para realizar pruebas más amplias. Eso no formaba parte de nuestro "entendimiento", pero las partes interesadas externas tenían esa expectativa sobre los resultados de los equipos.

La forma en que los equipos resolvieron este problema fue considerar nuestros criterios como un conjunto de objetivos rectores de varios niveles que envuelven cada fase, por así decirlo, del desarrollo ágil. Un ejemplo de esto se muestra en la Tabla 20-1.

Definir el nivel de cocción en estos niveles individuales ha demostrado funcionar para nuestros equipos y ha mejorado significativamente nuestra capacidad para cuantificar y cumplir con todas las expectativas de nuestros clientes. Tenga en cuenta que existe una conexión entre todos los criterios, por lo que definir en un nivel realmente ayuda a definir los demás. A menudo comenzamos en el nivel de Criterios de publicación y avanzamos "hacia atrás".

El desarrollo ágil no funciona si las historias, iteraciones o lanzamientos no lo son "hecho." "Estar listo" incluye pruebas, y las pruebas son a menudo lo que se pospone cuando hay poco tiempo. Asegúrese de que sus criterios de éxito en todos los niveles incluyan todas las pruebas necesarias para guiar y validar el desarrollo.

Tabla 20-1 Diferentes niveles de cocción

Actividad	Criterios	Ejemplo
Trabajo en equipo básico Productos	Criterios de cocción	Emparejamiento o inspecciones de código antes del check-in o del desarrollo, ejecución y aprobación de pruebas unitarias.
Nivel de historia de usuario	Pruebas de aceptación	Desarrollo de pruebas de aceptación basadas en FitNesse con el cliente Y su ejecución y aprobación exitosas
Sprint o iteración Nivel	Criterios de cocción	Definir un objetivo de Sprint que aclare el desarrollo de funciones y todas las dependencias externas asociadas con un sprint.
Nivel de lanzamiento	Criterios de liberación	Definir un conjunto amplio de condiciones (artefactos, actividades de prueba o niveles de cobertura, resultados/métricas, colaboración con otros grupos, cumplimiento de niveles de cumplimiento, etc.) que, de cumplirse, significarían que la liberación podría ocurrir.

Cada proyecto, cada equipo, cada negocio es único. Los equipos ágiles trabajan con el expertos en negocios para decidir cuándo están listos para entregar el software a producción. Si la fecha límite de publicación está fijada en piedra, la empresa tendrá que modificar alcance. Si hay suficiente flexibilidad para lanzar cuando el software tenga suficiente valor, los equipos pueden decidir cuándo se han cumplido los criterios de calidad y el software puede pasar a producción.

Compilaciones desafiantes de candidatos a lanzamiento

El equipo de Coni Tartaglia utiliza una lista de verificación para evaluar cada versión candidata a versión. La lista de verificación podría especificar que la compilación candidata a la versión:

- Incluye todas las características que proporcionan valor comercial para el lanzamiento, incluido el arte, los logotipos, los acuerdos legales y la documentación.
- Cumple con todos los criterios de aceptación de construcción.
- Tiene pruebas de que todas las pruebas acordadas (aceptación, integración, regresión).
- No tiene informes de defectos abiertos

El equipo de Coni desafía el software que podrían enviar con un conjunto final de inspecciones y "pruebas de aceptación de lanzamiento" o "RATS" acordadas. Ella explica:

La frase clave es "pruebas acordadas". Al acordar las pruebas de antemano, el alcance de la lista de verificación de liberación queda bien definido. Incluya pruebas de extremo a extremo a nivel de sistema en RATS y seleccione de la lista de pruebas de compatibilidad, lo que realmente desafiará la versión candidata a versión. Las pruebas de rendimiento también pueden incluirse en las RAT. Acuerde de antemano el contenido de las suites de automatización, así como un subconjunto de pruebas manuales para cada RAT.

Acuerde de antemano qué pruebas se repetirán si una RAT logra provocar el fallo de una versión candidata a versión. Si el software ha sobrevivido a varias iteraciones de pruebas de regresión automatizadas ejecutadas continuamente, superar estos desafíos finales debería ser muy sencillo.

La definición de los criterios de aceptación depende en última instancia de los clientes. Los evaluadores se encuentran en una posición única para ayudar al cliente y a los equipos de desarrollo a acordar los criterios que optimizan la calidad del producto.

El desarrollo de software tradicional funciona en largos períodos de tiempo, con plazos fijados con mucha antelación y obstáculos que superar de una fase a la siguiente. El desarrollo ágil nos permite producir software de calidad en pequeños incrementos y lanzarlo como necesario. Los equipos de desarrollo y de clientes pueden trabajar estrechamente para definir y decidir qué publicar y cuándo. Los evaluadores pueden desempeñar un papel fundamental en este proceso de establecimiento de objetivos.

Gestión de la liberación

Muchas organizaciones tienen un equipo de gestión de versiones, pero si no lo tienes, alguien hace el trabajo. Muchas veces en una organización pequeña es el responsable de calidad quien cumple este rol. La persona que dirige la liberación puede tener una liberación reunión de preparación con las partes interesadas para evaluar la preparación.

Una lista de verificación de preparación para el lanzamiento es una excelente herramienta para analizar lo que es importante para su equipo. La intención de esta lista de verificación es ayudar al equipo a determinar objetivamente lo que se completó e identificar los riesgos asociados con no completar una tarea.

Por ejemplo, si no se requiere capacitación porque los cambios realizados en el producto fueron transparentes para el usuario final, entonces el riesgo es bajo. Sin embargo, si hay Hubo cambios significativos en el proceso de creación de un nuevo usuario en el sistema, el riesgo sería muy alto para el soporte de producción o los equipos de ayuda, y puede justificar un retraso. Se deben considerar las necesidades de todas las partes interesadas.

Las notas de la versión son importantes para cualquier lanzamiento de producto. La formalidad de estos depende de la audiencia. Si su producto está dirigido a desarrolladores, entonces un "read El archivo de texto "me" probablemente esté bien. En otros casos, es posible que quieras hacerlos Más formal. Cualesquiera que sean los medios, deben abordar las necesidades de la audiencia. No proporciones mucha información adicional que no sea necesaria.

Cuando Janet recibe un nuevo lanzamiento, una de las primeras cosas que hace es comprobar el versión y todos los componentes. "¿Recibí lo que dijeron que me dieron? ¿Hay instrucciones especiales que debo considerar antes de realizar la instalación, como dependencias o scripts de actualización? Esas son buenas preguntas simples de responder. en las notas de la versión. Otras cosas a incluir son las nuevas características que el cliente debe buscar.

Las notas de la versión deben prestar especial atención a los componentes que no son parte de lo que entregó su equipo de desarrollo, como un archivo de ayuda o un usuario manuales preparados por un equipo diferente. A veces las notas de la versión antiguas se dejan los medios de lanzamiento, que pueden ser útiles o no para el usuario final. Considerar lo que es adecuado para su equipo y su aplicación.

embalaje

Hemos hablado mucho sobre la integración continua. Tendemos a darlo por sentado y olvídense de lo que significa una buena gestión de la configuración. "Construir una vez, implementar varias veces" es parte de lo que nos da confianza cuando lanzamos. Sabemos que la compilación que probamos en la prueba es la misma que probó el cliente

en UAT y es la compilación que lanzamos a producción. Esto es fundamental para una liberación exitosa.

Si el producto está destinado a un cliente externo, la instalación debe realizarse fácil, porque la instalación puede ser el primer vistazo al producto que tiene el cliente. Conozca a su audiencia y su nivel de tolerancia a los errores. ¿Cómo será? ¿Se entregará el producto? Por ejemplo, si se va a descargar de Internet, entonces debería ser una simple descarga e instalación. Si es una gran empresa sistema, entonces tal vez su organización necesite enviar una persona de soporte con el producto para ayudar con la instalación.

EXPECTATIVAS DEL CLIENTE

Antes de lanzar nuevo software a nuestros clientes, será mejor que nos aseguremos de que están listos para ello. Debemos asegurarnos de que sepan qué nuevas funciones esperar. y que tengan algunos medios para hacer frente a los problemas que surjan.

Soporte de producción

Muchas organizaciones tienen un equipo de soporte de producción u operaciones que mantiene el código y brinda soporte a los clientes después de que esté en producción. Si tu La empresa tiene un equipo de soporte de producción, ese grupo es su primer cliente. Convírtelo también en tu compañero. Los equipos de soporte de producción reciben informes de defectos y solicitudes de mejora de los clientes, y pueden trabajar con su equipo para identificar áreas de alto riesgo.

Muy a menudo, el equipo de soporte de producción es el equipo que acepta el lanzamiento. del equipo de desarrollo. Si su organización tiene este tipo de traspaso, Es importante que su equipo de desarrollo trabaje estrechamente con el de producción. equipo de soporte para que la transición sea sin problemas. Asegúrese de que el equipo de soporte de producción comprenda cómo utilizar los archivos de registro del sistema y la mensajería. y sistemas de monitoreo para realizar un seguimiento de las operaciones e identificar problemas rápidamente.

Comprender el impacto en las empresas

Cada vez que una implementación a producción requiere una interrupción, el producto se no disponible para su cliente. Si su producto es un sitio web, esto puede ser un enorme impacto. Si su producto es un producto independiente que debe descargarse en un PC, el impacto es bajo. Los equipos ágiles lanzan con frecuencia para maximizar el valor para el las empresas, y las emisiones pequeñas tienen un menor riesgo de tener un gran impacto negativo. Es sentido común para trabajar con la empresa para cronometrar las liberaciones por períodos de tiempo

que minimicen las perturbaciones. Automatizar y optimizar los procesos de implementación como tanto como sea posible para mantener pequeñas las ventanas de tiempo de inactividad. Un despliegue rápido El proceso también es útil durante el desarrollo en iteraciones cortas donde podemos desplegar una docena de veces en un día.

Consideraciones internacionales

Markus Gärtner, líder del grupo de pruebas "afectados por la agilidad", explica el enfoque de su equipo para programar sus lanzamientos:

Desarrollamos software de telecomunicaciones para móviles, por lo que normalmente instalamos nuestro software por la noche, cuando es probable que nadie haga llamadas. Esto puede ocurrir durante nuestro horario de oficina, cuando atendemos a un cliente en Australia, pero generalmente es durante la noche.

Mis colegas que realizan la instalación real (hay tres en nuestro equipo) tienen más probabilidades de llegar tarde durante el horario de oficina del día siguiente porque no tenemos un grupo separado para estas tareas.

A medida que las empresas y los equipos de desarrollo se vuelven más globales, el momento del lanzamiento se vuelve más complicado. Afortunadamente, las configuraciones de producción pueden facilitar los lanzamientos. Si su entorno de producción tiene varios servidores de aplicaciones, es posible que pueda desactivarlos uno por uno para su lanzamiento sin interrumpir a los usuarios.

Los nuevos lanzamientos deben ser lo más transparentes posible para el cliente. Cuanto menos lanzamientos de emergencia o parches requeridos después de un lanzamiento, más confianza su cliente tendrá tanto en el producto como en el equipo de desarrollo.

Aprenda de cada lanzamiento y tome medidas para que el próximo funcione mejor suavemente. Involucrar a todos los roles, como administradores de sistemas y bases de datos, en la planificación. Evalúe cada lanzamiento y piense en formas de mejorar el siguiente.

RESUMEN

Este capítulo cubrió los siguientes puntos:

- La entrega exitosa de un producto incluye más que solo la aplicación que está creando.
- Planifique los entregables no relacionados con el software, como documentación, avisos legales y capacitación.
- El final del juego es una oportunidad para darle el toque final a su producto.

Otros grupos pueden ser responsables de los entornos, las herramientas y otros componentes del juego final y del lanzamiento. Coordine con ellos con anticipación.

Asegúrese de probar los scripts de actualización de la base de datos, las conversiones de datos y otras partes de la instalación.

UAT es una oportunidad para que los clientes prueben sus datos y desarrollen su confianza en el producto.

Presuponga tiempo para ciclos adicionales según sea necesario, como ciclos posteriores al desarrollo para coordinar las pruebas con partes externas.

Establezca criterios de aceptación de lanzamiento durante la planificación del lanzamiento para que pueda saber cuándo está listo para lanzarlo.

Los evaluadores a menudo participan en la gestión de lanzamientos y las pruebas del paquete.

Al lanzar el producto, considere el paquete completo: lo que el cliente necesita y espera.

Aprenda de cada versión y adáptese para mejorar sus procesos.

Esta página se dejó en blanco intencionalmente.

Parte VI

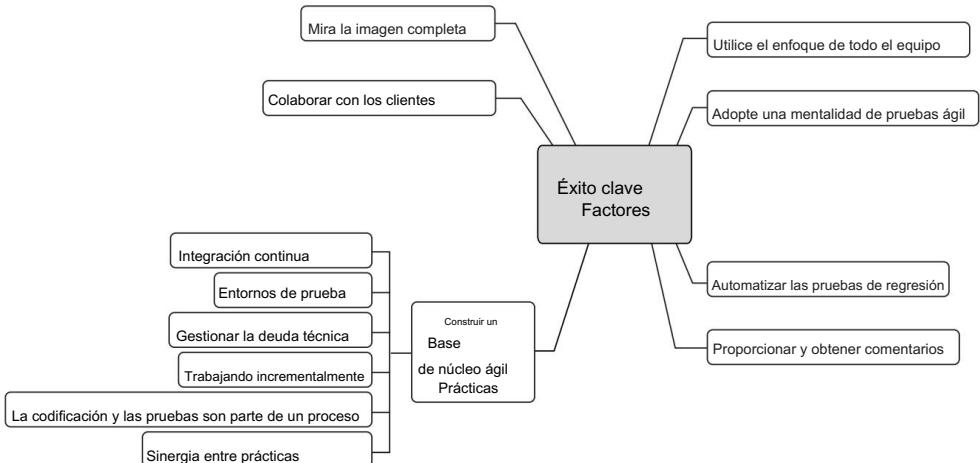
RESUMEN

En el Capítulo 21, “Factores clave de éxito”, reunimos todo y resumimos el enfoque ágil para las pruebas.

Esta página se dejó en blanco intencionalmente.

Capítulo 21

FACTORES CLAVES DEL ÉXITO



Después de haber viajado a través de una iteración y más allá, siguiendo a un evaluador ágil mientras participa en muchas actividades, ahora podemos seleccionar algunos factores clave que ayudan a los evaluadores a tener éxito en equipos ágiles y ayudan a los equipos ágiles a tener éxito en la entrega de un producto de alta calidad. Creemos que los evaluadores ágiles tienen algo especial que ofrecer. Los evaluadores “infectados por Agile” aprenden cómo aplicar prácticas y principios ágiles para ayudar a todo su equipo a producir un mejor producto. Los programadores “infectados por las pruebas” en equipos ágiles aprenden a utilizar las pruebas para producir un mejor trabajo. Las líneas entre roles son borrosas, pero eso es algo bueno. Todo el mundo está centrado en la calidad.

Hemos recopilado algunas pautas de prueba críticas para equipos y evaluadores ágiles a partir de nuestra propia prueba y error, así como de los equipos con los que hemos trabajado.

Estas pautas se basan en la matriz de pruebas ágiles, en nuestra experiencia de aprender a superar obstáculos culturales y organizacionales, nuestras aventuras al desempeñar el rol de probador en equipos ágiles y nuestra experiencia en descubrir cuál es la mejor manera de utilizar la automatización de pruebas. Nos gustan los números de la suerte, por eso en este capítulo presentamos siete factores clave que ayudan a un evaluador ágil a tener éxito.

Le pedimos a un pequeño grupo de personas que estaban revisando algunos de nuestros capítulos que sugirieran el orden en el que presentar estos factores de éxito. Los resultados variaron bastante, aunque muchos (pero no todos) coincidieron en los dos primeros. Elija el factor de éxito que le brindará el mayor retorno de la inversión y comience a trabajar en él hoy.

FACTOR DE ÉXITO 1: UTILIZAR EL ENFOQUE DE EQUIPO COMPLETO

Cuando todo el equipo de desarrollo asume la responsabilidad de las pruebas y la calidad, se dispone de una gran variedad de habilidades y niveles de experiencia para afrontar cualquier problema de prueba que pueda surgir. La automatización de pruebas no es un gran problema para un grupo de programadores expertos. Cuando las pruebas son una prioridad del equipo y cualquiera Puede registrarse para tareas de prueba, el equipo diseña código comprobable.

Hacer que los evaluadores formen realmente parte del equipo de desarrollo significa darles la el apoyo y la formación que necesitan para adaptarse al rápido ritmo del desarrollo ágil. Tienen tiempo para adquirir nuevas habilidades para colaborar estrechamente. con miembros de los equipos de desarrollo y de clientes.

Si gestiona un equipo ágil, utilice las sugerencias de la Parte II, "Organizacional Desafíos", para ayudar a su equipo a adoptar el enfoque de equipo completo. Recordar que la calidad, no la velocidad, es el objetivo del desarrollo ágil. Tu equipo necesita probadores para ayudar a los clientes a aclarar los requisitos y convertirlos en pruebas que guíen desarrollo y proporcionar un punto de vista único que promoverá la entrega de un producto sólido. Asegúrese de que los evaluadores puedan transferir sus habilidades y experiencia a el resto del equipo. Asegúrese de que no estén encasillados en un rol como el de solo haciendo pruebas manuales. Asegúrese de que cuando pidan ayuda (lo que puede requerir mucho coraje de su parte), los miembros de su equipo se la brinden. El lo contrario también es cierto; un evaluador debe dar un paso al frente cada vez que alguien necesite la ayuda que pueda brindar.

Consulte el Capítulo 2, "Diez principios para evaluadores ágiles", para ver un ejemplo de cómo funciona el "Poder de tres".

Si usted es un evaluador en un equipo ágil y hay reuniones de planificación y discusiones de diseño que no lo incluyen a usted ni a los usuarios comerciales, luchando por definir sus historias y requisitos solo, es hora de levantarse y ve a hablar con el resto del equipo. Siéntate con los programadores, invítate a las reuniones y proponer probar el "Poder de Tres" involucrando a un probador, un programador y experto en negocios. Ser útil, dar retroalimentación y ayudar.

Los clientes dan ejemplos. Haz de tus problemas los problemas del equipo, y haz tuyos sus problemas. Pide a tus compañeros de equipo que adopten un equipo completo acercarse.

FACTOR DE ÉXITO 2: ADOPTE UN ÁGIL PRUEBA DE MENTALIDAD

En el Capítulo 2, "Diez principios para los probadores ágiles", advertimos a los probadores ágiles que perder cualquier mentalidad de "Policía de Calidad" que podrían haber traído consigo.

Ahora estás en un equipo ágil, donde los programadores realizan pruebas y los evaluadores hacen todo lo que se les ocurre para ayudar al equipo a ofrecer el mejor producto posible. Como

Como enfatizamos en el Capítulo 2, una actitud de prueba ágil es proactiva, creativa y Abierto a nuevas ideas y dispuesto a asumir cualquier tarea. El probador ágil constantemente perfecciona su oficio, siempre está dispuesta a colaborar, confía en sus instintos y le apasiona ayudar al equipo y al negocio a tener éxito.

No queremos decir que debas ponerte tu capa de Super Tester e ir a protegerte. el mundo de los insectos. No hay lugar para grandes egos en los equipos ágiles. Su Los compañeros de equipo comparten su pasión por la calidad. Concéntrate en los objetivos del equipo y hazlo. lo que pueda para ayudar a todos a hacer su mejor trabajo.

Utilice principios y valores ágiles para guiarse. Pruebe siempre el enfoque más sencillo para satisfacer una necesidad de prueba. Sea valiente al buscar ayuda y experimentar con nuevas ideas. Centrarse en ofrecer valor. Comunicarse tan directamente y con la mayor frecuencia posible. Sea flexible al responder al cambio. Recuerda eso el desarrollo ágil se centra en las personas y que todos deberíamos disfrutar de nuestro trabajo. En caso de duda, recurra a los valores y principios para decidir qué hacer.

Un componente importante de la mentalidad de pruebas ágiles es el impulso de encontrar continuamente mejores formas de trabajar. Un evaluador ágil exitoso pule constantemente su oficio. Lea buenos libros, blogs y artículos para obtener nuevas ideas y habilidades. Asistir a reuniones de grupos de usuarios locales. Participe en discusiones de listas de correo para obtener retroalimentación sobre problemas o nuevas ideas. Si su empresa no le paga por asistir a una buena conferencia, escriba lo que ha aprendido en un informe de experiencia para canjee por una inscripción gratuita a la conferencia. Devolviendo a sus pruebas y Las comunidades de desarrollo ágil también le ayudarán.

Experimente con nuevas prácticas, herramientas y técnicas. Anima a tu equipo para probar nuevos enfoques. Las iteraciones cortas son ideales para la experimentación. Puede que falles, pero será rápido y podrás intentar otra cosa.

Si gestiona evaluadores ágiles o un equipo ágil, déles tiempo para aprender y bríndeles apoyo para la capacitación que necesitan. Quitar obstáculos para que puedan hacer su mejor trabajo.

Cuando se enfrente a problemas que afecten las pruebas, tráigalos al equipo. Pídale al equipo que piense en formas de superar estos obstáculos. Las retrospectivas son un lugar para hablar sobre problemas y cómo resolverlos. Mantenga una acumulación de impedimentos y aborde uno o dos en cada iteración. Usar grandes gráficos visibles, o sus equivalentes virtuales, para garantizar que todos estén consciente de los problemas que surgen y de que todos pueden seguir el progreso de la codificación y las pruebas.

Consulte el Capítulo 2,
"Diez principios para los
evaluadores ágiles", para
obtener más información
sobre la mentalidad
de las pruebas ágiles.

FACTOR DE ÉXITO 3: PRUEBAS DE REGRESIÓN AUTOMATIZADAS

¿Puede un equipo ágil tener éxito sin automatización de pruebas? Tal vez, pero los equipos exitosos que conocemos dependen de pruebas de regresión automatizadas. Como hemos dicho muchas veces en este libro, si dedica todo su tiempo a realizar regresión manual pruebas, nunca tendrás tiempo para las importantes pruebas exploratorias que Descubrir los comportamientos dañinos que se esconden en el código.

El desarrollo ágil utiliza pruebas para guiar el desarrollo. Para escribir código en Para pasar la prueba, necesita una forma rápida y sencilla de ejecutar la prueba. Sin el Ciclo de retroalimentación corto y regresión de red de seguridad que brindan las suites, su equipo pronto quedará atrapado en una deuda técnica, con una creciente cola de defectos y velocidad cada vez más lenta.

—
Consulte la Parte II para obtener más información sobre los cuadrantes de pruebas ágiles.

Automatizar las pruebas de regresión es un esfuerzo de equipo. Todo el equipo debe elegir. Herramientas adecuadas para cada tipo de prueba. Pensar en las pruebas por adelantado le permitirá Los programadores diseñan código para facilitar la automatización de pruebas. Utilice las pruebas ágiles Cuadrantes y pirámide de automatización de pruebas para ayudarle a automatizar diferentes tipos de pruebas de manera efectiva.

—
Consulte el Capítulo 14, "Estrategia de automatización", para obtener más información sobre la pirámide de automatización de pruebas.

Recuerde comenzar de manera simple. Te sorprenderá el valor de algunos productos básicos. Las pruebas de humo automatizadas o las pruebas unitarias automatizadas pueden proporcionar.

La automatización de pruebas es un esfuerzo de equipo. También es difícil, al menos al principio. A menudo hay un gran "joroba de dolor" que superar. Si gestionas un desarrollo o testing equipo, asegúrate de brindar suficiente apoyo en forma de tiempo, capacitación y motivación. Si eres un tester en un equipo sin automatización y el Los programadores están demasiado frenéticos al intentar escribir código de producción como para detenerse y Piensa en probar, tienes un gran desafío por delante. Experimentar con diferentes formas de obtener apoyo de la gerencia y de los miembros del equipo para iniciar un pequeño esfuerzo de automatización.

—
Consulte la bibliografía para obtener recursos sobre cómo promover el cambio.

FACTOR DE ÉXITO 4: PROPORCIONAR Y OBTENER RETROALIMENTACIÓN

La retroalimentación es un valor ágil central. Las iteraciones cortas de Agile están diseñadas para proporcionar retroalimentación constante para mantener al equipo encaminado. Los probadores están en una posición única para ayudar a proporcionar retroalimentación en forma de resultados de pruebas automatizadas, descubrimientos realizados durante las pruebas exploratorias y observaciones de situaciones reales. usuarios del sistema.

Ágil se trata de retroalimentación

Bret Pettichord, CTO de WatirCraft y coautor de compartió estas ideas [Lecciones aprendidas en Suave-mercancía Pruebas](#), sobre la importancia de la retroalimentación para el desarrollo ágil.

Los métodos ágiles le permiten a su equipo obtener comentarios sobre el software que está creando. Ese es el punto. La retroalimentación funciona en varios niveles.

La programación en pares brinda a los desarrolladores comentarios instantáneos sobre su código.

Las historias representan unidades de trabajo donde los evaluadores y analistas pueden brindar retroalimentación a los desarrolladores. Los lanzamientos de iteraciones facilitan la retroalimentación desde fuera del equipo. La mayoría de las prácticas ágiles son valiosas porque crean circuitos de retroalimentación que permiten a los equipos adaptarse.

Muchos equipos adoptan Agile con un enfoque instantáneo sin darse cuenta del objetivo de las prácticas. Programan en pareja sin discusión ni cambio de controladores. Envían código al control de calidad que los evaluadores no pueden probar porque los límites de la historia son arbitrarios; no pueden decir si encontraron un error o simplemente el final de la historia. Las iteraciones se convierten en hitos del cronograma en lugar de oportunidades reales para mejorar la alineación y ajustar los objetivos.

La razón por la que los equipos ágiles pueden funcionar con menos planificación es porque la retroalimentación le permite asegurarse de que está en el camino correcto. Si no tienes retroalimentación significativa, entonces no eres ágil. Simplemente estás en una nueva forma de caos.

En mi último proyecto, definimos nuestras historias para que tuvieran sentido para todos los miembros del equipo. Nuestros analistas, evaluadores y desarrolladores pudieron comprender y revisar historias individuales. Pero descubrimos que teníamos que crear un grupo más grande, al que llamamos funciones, para facilitar una revisión significativa desde fuera de nuestro equipo. Nos aseguramos de que todas las historias de una función estuvieran completas antes de solicitar comentarios de fuera del equipo.

Ser capaz de dar y recibir comentarios significativos suele ser un desafío para las personas. Sin embargo, es crucial para el éxito con Agile.

Los equipos ágiles se meten en aprietos terribles cuando los ejecutivos o clientes les entregan una lista de requisitos al principio y les dicen que utilicen Agile (porque es más rápido), y luego no quiere participar en el proceso de retroalimentación.

Agile no es más rápido por sí solo. La agilidad es solo un beneficio en un mundo que reconoce el valor de la adaptación. Y esa adaptabilidad debe llegar hasta quienquiera que financie el proyecto. No basta con que el equipo sea ágil. Los patrocinadores también deben ser ágiles. ¿Son realmente necesarios todos los requisitos? ¿Sabemos exactamente cómo debe verse el software desde el principio?

Agile es más rápido porque la retroalimentación le permite encontrar y concentrarse en las funciones más valiosas. Si está seguro de saber lo que se debe crear, no utilice Agile. Si no tiene tiempo para recopilar y actuar en función de los comentarios de los clientes, no utilice Agile. Si está seguro de que todos entienden exactamente lo que se debe hacer desde el principio, entonces no utilizar ágil.

Las prácticas ágiles crean una infraestructura técnica y organizativa para facilitar la obtención de comentarios y la actuación en función de ellos. Si no va a adaptarse a la retroalimentación, entonces esta infraestructura es un desperdicio que sólo lo ralentizará.

Para nosotros, el valor del desarrollo ágil no es que sea más rápido, sino que ofrece suficiente valor con la suficiente rapidez para ayudar al negocio a crecer y tener éxito. Los evaluadores desempeñan un papel clave a la hora de proporcionar la retroalimentación que permite que eso suceda.

Los evaluadores también necesitan comentarios. ¿Cómo sabe que tiene los ejemplos correctos del comportamiento deseado por parte de los clientes? ¿Cómo saber si la prueba ¿Los casos que escribiste reflejan estos ejemplos correctamente? ¿Pueden los programadores comprender qué codificar observando los ejemplos que ha capturado y el ¿Pruebas que has creado?

Una de las habilidades más valiosas que puedes aprender es cómo pedir comentarios sobre tu propio trabajo. Pregunte a los programadores si obtienen suficiente información para comprender los requisitos y si esa información guía su codificación. Pregunte a los clientes si creen que se están cumpliendo sus criterios de calidad. Tómese el tiempo en Tanto las reuniones de planificación de iteraciones como las retrospectivas para hablar sobre estos problemas y sugerir formas de mejorar.

FACTOR DE ÉXITO 5: CONSTRUIR UNA CIMENTO DE PRÁCTICAS BÁSICAS

Un viejo dicho en el negocio de las pruebas es: "No se puede probar la calidad del producto". Por supuesto, esto también se aplica al desarrollo ágil. Sentimos que no puedes entregar software de alta calidad sin seguir algunas prácticas fundamentales. Si bien pensamos que estas son prácticas ágiles, han existido por más tiempo. el término "desarrollo ágil", y son simplemente prácticas centrales de éxito desarrollo de software.

Integración continua

Todo equipo de desarrollo necesita gestión del código fuente e integración continua para tener éxito. No puedes realizar pruebas efectivas si no sabes exactamente

lo que estás probando y no puedes probar nada si no tienes un código que puedas implementar.

Todos los miembros del equipo deben revisar su trabajo al menos una vez al día. Cada

La integración debe ser verificada mediante una compilación automatizada que incluya pruebas para proporcionar información rápida sobre el estado del software.



Consulte la
bibliografía para
obtener más información.
sobre la integración
continua.

Implementar un proceso de integración continua debería ser uno de los primeros prioridades de cualquier equipo de desarrollo de software. Si tu equipo no tiene en Al menos una compilación verificada diariamente, deja lo que estás haciendo y comienza una. Es que importante. No es necesario que sea perfecto para empezar. Si tienes un enorme integrar el sistema, definitivamente es más desafiante. Sin embargo, en general es No es tan difícil. Hay una gran cantidad de herramientas excepcionales, tanto de código abierto y comerciales, disponibles para este fin.

Entornos de prueba

No puede realizar pruebas de manera productiva sin un entorno de prueba que usted controle. Tú Necesita saber qué compilación se implementa, qué esquema de base de datos se utiliza, si alguien más está actualizando ese esquema y qué otros procesos están funcionando en la máquina.

El hardware es cada vez más barato y cada vez hay más software de código abierto. Está disponible y se puede utilizar para entornos de prueba. Su equipo debe hacer el inversión para que pueda realizar eficazmente pruebas exploratorias manuales y automatizadas de forma rápida y eficiente. Si hay un problema con el entorno de prueba, dígalo y deje que el equipo lo resuelva creativamente.

Gestionar la deuda técnica

Incluso los buenos equipos de desarrollo de software, al sentir presión de tiempo, descuidan la refactorización o recurren a soluciones rápidas y trucos para resolver un problema rápidamente. como el El código se vuelve más confuso y difícil de mantener, aparecen más errores, y no pasa mucho tiempo antes de que la velocidad del equipo se vea consumida por las correcciones de errores. e intentar darle sentido al código para agregar nuevas funciones. Su El equipo debe evaluar constantemente la cantidad de deuda técnica que lo arrastra. reducirlo y trabajar para reducirlo y prevenirlo.

La gente suele decir: "Nuestra dirección no nos dará tiempo para hacer las cosas bien, "No tenemos tiempo para refactorizar y tenemos plazos ajustados". Sin embargo, es No es difícil presentar un caso de negocio claro que muestre lo que significa la creciente deuda técnica. le está costando a la empresa. Hay muchas formas de medir el código y los defectos. tasas que pueden traducir la deuda técnica en su impacto en el resultado final. Simplemente señalar su velocidad decreciente puede ser suficiente. Las empresas necesitan

sus equipos de desarrollo de software sigan siendo consistentemente productivos. Ellos Es posible que tenga que reducir el alcance de las características deseadas para permitir tiempo suficiente para un buen diseño de código guiado por pruebas y buenas prácticas como pequeña refactorización continua.

Una buena cobertura de las pruebas de regresión automatizadas es clave para minimizar los problemas técnicos. deuda. Si faltan, reserve tiempo en cada iteración para desarrollar las pruebas automatizadas, planifique una "iteración de refactorización" para actualizar o agregar las herramientas necesarias, y escribir pruebas y realizar importantes esfuerzos de refactorización. En cada iteración, tome el Es hora de guiar el código con pruebas, refactorizar el código que estás tocando según sea necesario y agregue pruebas automatizadas donde faltan. Incrementa tus estimaciones a cuenta para este trabajo. A la larga, el equipo podrá ir mucho más rápido.

Trabajando incrementalmente

Una de las razones por las que los equipos ágiles pueden crear un producto de calidad es que trabajan a pequeña escala. Las historias representan unos días de trabajo, y cada historia puede ser roto en varias láminas finas o hilos de acero y construido paso a paso. Esto permite probar cada pieza pequeña y luego probar incrementalmente a medida que las piezas se van juntar.

Lea más sobre trozos pequeños y porciones finas en el Capítulo 8, "Pruebas empresariales que respaldan al equipo".

Si los miembros de su equipo se sienten tentados a asumir una gran parte de la funcionalidad inmediatamente, anímelos a considerar un enfoque gradual. Hacer preguntas: "¿Cuál es el valor empresarial central de esta historia? ¿Cuál es el camino más básico? a través de este fragmento de código? ¿Qué vendría después? Sugerir tarea de escritura, tarjetas para codificar y probar las piezas pequeñas, obtener una prueba de concepto para su diseño, y confirme su estrategia de pruebas y automatización de pruebas.

La codificación y las pruebas son parte de un proceso

Las personas que son nuevas en Agile a menudo preguntan a los evaluadores ágiles: "¿Qué haces hasta que todos ¿Las historias están terminadas y puedes probarlas? Los practicantes ágiles experimentados dicen: "Los evaluadores deben participar durante toda la iteración, todo el proceso de desarrollo. De lo contrario, no funciona".

Los evaluadores escriben pruebas, basadas en ejemplos proporcionados por los clientes, para ayudar a los programadores a comprender la historia y comenzar. Las pruebas y los ejemplos proporcionan un lenguaje común que entienden todos los involucrados en la producción del software. Los evaluadores y programadores colaboran estrechamente a medida que avanza la codificación, y ambos también colaborar estrechamente con los clientes. Los programadores muestran a los probadores funcionalidad que han escrito, y los evaluadores muestran a los programadores lo inesperado comportamientos que han encontrado. Los evaluadores escriben más pruebas a medida que avanza la codificación,

Los participantes los hacen pasar y los evaluadores realizan más pruebas exploratorias para saber si Se ha entregado el valor correcto. Cada iteración ágil consta de docenas de iteraciones constantes, rápidas e incrementales de prueba-código-prueba-código-prueba.

Cuando este ciclo de colaboración y retroalimentación se altera y las pruebas se separan del desarrollo, suceden cosas malas. Si se prueba una historia en la iteración después de la cual fue codificada y se encuentran errores, el programador tiene que dejar de trabajar en la nueva historia, recuerda cómo funcionó el código durante la última historia de la iteración, corríjala y espere a que alguien pruebe la solución. Hay pocos hechos en el desarrollo de software, pero sabemos con certeza que los errores son más baratos de corregir cuanto antes se encuentren.

Lea más sobre codificación y pruebas en el Capítulo 18, "Codificación y pruebas".

Cuando la codificación se guía constantemente por pruebas, y las pruebas ocurren junto con codificación, es mucho más probable que logremos el comportamiento y proporcionemos el valor que el cliente quería. Las pruebas son una responsabilidad del equipo. Si tu equipo no comparte este punto de vista, pida a todos que piensen en su enfoque en la calidad, su deseo de ofrecer el mejor producto posible y qué pasos pueden tomar para asegurar que el equipo alcance sus objetivos.

Sinergia entre prácticas

Una única práctica de desarrollo ágil, como la integración continua, puede marcar la diferencia. diferencia, pero la combinación de múltiples prácticas ágiles es mayor que la suma de las partes. El diseño basado en pruebas, la propiedad colectiva del código y la integración continua brindan retroalimentación rápida y mejoran continuamente el código. diseño y la capacidad de ofrecer valor empresarial rápidamente. Automatizar pruebas es Bueno, pero usar pruebas automatizadas para impulsar el desarrollo, seguidas de pruebas exploratorias para detectar brechas o debilidades, es mucho mejor en muchos niveles.

Algunas prácticas no funcionan bien de forma aislada. La refactorización es imposible sin pruebas automatizadas. Es posible realizar pequeños lanzamientos en forma de minicascada y evitar todos los beneficios del desarrollo ágil. Si su cliente en el sitio no es empoderada para tomar decisiones, su valor para el equipo es limitado.

Las prácticas ágiles fueron diseñadas para complementarse entre sí. Tómese el tiempo para comprender el propósito de cada uno, considere lo que se necesita para aprovecharlo al máximo. cada práctica y tome decisiones reflexivas sobre lo que funciona para su equipo.

FACTOR DE ÉXITO 6: COLABORA CON LOS CLIENTES

Uno de los mayores valores que los evaluadores aportan a los equipos ágiles es ayudar Los clientes aclaran y priorizan los requisitos, ilustrando los requisitos.

con ejemplos concretos de comportamiento deseado y escenarios de usuario, y convirtiendo esos ejemplos en pruebas ejecutables. Los evaluadores hablan el idioma del dominio de el lenguaje empresarial y técnico del equipo de desarrollo. Hacemos Buenos facilitadores y traductores.

Nunca interfieras en la comunicación directa entre programadores y clientes. Fomente la mayor comunicación directa posible. Utilizar el "Poder de tres". Cuando los requisitos no se cumplen o se malinterpretan, el cliente, el programador y el evaluador deben trabajar juntos para obtener respuestas a las preguntas. Haga que los clientes hablen frente a una pizarra o su virtual equivalente tantas veces como sea necesario. Si los clientes se encuentran dispersos por el campus, el país o el mundo, utilice todas las herramientas que pueda encontrar para mejorar la comunicación y la colaboración. Teleconferencias, mensajes instantáneos y wikis No son un sustituto ideal de la conversación cara a cara, pero son mejores que enviar correos electrónicos o no hablar en absoluto.

FACTOR DE ÉXITO 7: MIRAR EL PANORAMA GRANDE

Esta es una generalización, por supuesto, pero hemos descubierto que los evaluadores tienden a mirar el panorama general y generalmente desde el punto de vista del cliente. Programadores normalmente tienen que concentrarse en entregar la historia en la que están trabajando ahora, y Si bien pueden utilizar pruebas para guiarse, deben centrarse en la implementación técnica de los requisitos.

Este punto de vista general es una gran contribución para el equipo. Basado en pruebas El desarrollo, bien hecho, ofrece un código sólido que, de forma aislada, puede estar libre de defectos. ¿Qué pasa si esa nueva característica causa que alguna parte aparentemente no relacionada de la aplicación para romper? Alguien tiene que considerar el impacto a mayor escala. sistema y comunicarlo al equipo. ¿Qué pasa si hemos pasado por alto algunos? ¿Pequeño detalle que irritará a los clientes? La nueva interfaz de usuario puede ser perfecta codificado, pero si el color de fondo hace que el texto sea difícil de leer, eso es lo que el usuario final lo notará.

Utilice los cuadrantes de pruebas ágiles como guía para ayudarle a planificar las pruebas que cubrir todos los ángulos. Utilice la idea de la pirámide de pruebas para garantizar un buen retorno de la inversión de su automatización de pruebas. Guiar el desarrollo con pruebas ayuda a garantizar que no Se pierde algo grande, pero no es perfecto. Utilice pruebas exploratorias para aprender Más información sobre cómo debería funcionar la aplicación y qué dirección deben tomar sus pruebas. Haga que sus entornos de prueba sean lo más similares posible a los de producción, utilizando datos que reflejen el mundo real. Sea diligente al recrear una situación de estilo de producción para actividades como pruebas de carga.

La Parte III explica cómo utilizar los cuadrantes de pruebas ágiles.

Es fácil para todos los miembros del equipo concentrarse únicamente en la tarea o historia en cuestión. Ésa es una desventaja de trabajar en pequeñas porciones de funcionalidad a la vez. Ayude a su equipo a dar un paso atrás de vez en cuando para evaluar cómo sus historias actuales encajan en el gran esquema del negocio. Sigan preguntándose cómo pueden hacer un mejor trabajo para ofrecer valor real.

RESUMEN

Las pruebas y la calidad son responsabilidad de todo el equipo, pero los evaluadores aportan un punto de vista especial y habilidades únicas. Como evaluador, su pasión por ofrecer un producto que deleite a sus clientes lo ayudará a superar las frustraciones que usted y su equipo puedan encontrar. No tengas miedo de ser un agente de mejora continua. Deje que los principios y valores ágiles lo guíen mientras trabaja con el cliente y los equipos de desarrollo, agregando valor en cada iteración.

En este capítulo final, analizamos siete factores clave para el éxito de las pruebas ágiles:

1. Utilice el enfoque 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 básicas.
6. Colaborar con los clientes.
7. Mire el panorama general.

Esta página se dejó en blanco intencionalmente.

GLOSARIO

Este glosario contiene las definiciones de los autores de los términos utilizados en este libro.

Prueba de aceptación Las pruebas de aceptación son pruebas que definen el valor comercial que debe ofrecer cada historia. Pueden verificar requisitos funcionales o requisitos no funcionales, como el rendimiento o la confiabilidad. Aunque se utilizan para ayudar a guiar el desarrollo, se encuentran en un nivel más alto que las pruebas de nivel unitario utilizadas para el diseño de código en el desarrollo basado en pruebas. La prueba de aceptación es un término amplio que puede incluir pruebas tanto comerciales como tecnológicas.

Las API de interfaz de programación de aplicaciones (API) permiten que otro software invoque alguna funcionalidad. La API puede consistir en funciones, procedimientos o clases que admitan solicitudes realizadas por otros programas.

Compilación Una compilación es el proceso de convertir el código fuente en un artefacto implementable que se puede instalar para ejecutar la aplicación. El término "compilación" también se refiere al artefacto desplegable.

Componente Un componente es una parte más grande del sistema general que puede implementarse por separado. Por ejemplo, en la plataforma Windows, las bibliotecas vinculadas dinámicas (DLL) se utilizan como componentes, los archivos Java (archivos JAR) son componentes en la plataforma Java y una arquitectura orientada a servicios (SOA) utiliza servicios web como componentes.

Prueba de componentes Una prueba de componentes verifica el comportamiento de un componente. Las pruebas de componentes ayudan con el diseño de componentes al probar las interacciones entre objetos.

Condiciones de satisfacción Las condiciones de satisfacción, también llamadas condiciones de satisfacción o condiciones de satisfacción empresarial, son suposiciones y decisiones clave que toma el equipo del cliente para definir el comportamiento deseado de

el código entregado para una historia determinada. Las condiciones de satisfacción son criterios mediante los cuales se puede medir el resultado de una historia. Evolucionan durante las conversaciones con el cliente sobre criterios de aceptación de alto nivel para cada historia. Discutir las condiciones de satisfacción ayuda a identificar suposiciones riesgosas y aumenta la confianza del equipo a la hora de escribir y estimar correctamente todas las tareas para completar la historia.

Pruebas basadas en el contexto Las pruebas basadas en el contexto siguen siete principios, el primero de los cuales es que el valor de cualquier práctica depende de su contexto. Cada nuevo proyecto y cada nueva aplicación pueden requerir diferentes maneras de abordar un proyecto. Las siete prácticas se pueden encontrar en el sitio web www.context-driven-testing.com/.

Equipo de atención al cliente El equipo de atención al cliente identifica y prioriza las funciones que necesita la empresa. En Scrum, estas características se convierten en epopeyas o temas, que se dividen en historias y comprenden el trabajo pendiente del producto. Los equipos de clientes incluyen a todas las partes interesadas fuera del equipo de desarrollo, como expertos comerciales, expertos en la materia y usuarios finales. Los evaluadores y desarrolladores trabajan en estrecha colaboración con el equipo del cliente para especificar ejemplos de comportamiento deseado para cada historia y convertir esos ejemplos en pruebas para guiar el desarrollo.

Prueba de cliente Una prueba de cliente verifica el comportamiento de una porción o parte de una funcionalidad que es visible para el cliente y se relaciona directamente con una historia o característica. Los términos "prueba de cara al negocio" y "prueba de cara al cliente" se refieren al mismo tipo de prueba que la prueba del cliente.

Equipo de desarrollo El equipo de desarrollo es el equipo técnico que produce el software solicitado por el equipo del cliente. Todos los involucrados en la entrega de software son desarrolladores, incluidos programadores, evaluadores, expertos en bases de datos, administradores de sistemas, redactores técnicos, arquitectos, expertos en usabilidad y analistas. Este equipo de desarrollo trabaja en conjunto para producir el software y brindar valor al negocio, ya sea un equipo ubicado en el mismo lugar o un equipo virtual.

Épica Una épica es una funcionalidad o característica descrita por el cliente y es un elemento del trabajo pendiente del producto. Una epopeya se divide en historias relacionadas que luego se dimensionan y estiman. Algunos equipos utilizan el término "tema" en lugar de épico.

Pruebas exploratorias Las pruebas exploratorias son pruebas interactivas que combinan el diseño de pruebas con la ejecución de pruebas y se centran en aprender sobre la aplicación.

Consulte el Capítulo 10, "Pruebas comerciales que critican el producto", para obtener una definición detallada de pruebas exploratorias.

Objeto falso Un objeto falso reemplaza la funcionalidad del componente del que se depende con una implementación más simple. Emula el comportamiento del componente real del que depende, pero es más fácil de usar con fines de prueba.

Característica Una característica es una parte de la funcionalidad descrita por el cliente y es un elemento en el trabajo pendiente del producto. Una característica se divide en historias relacionadas que luego se dimensionan y estiman. En el desarrollo ágil, los términos "épico" o "tema" se utilizan a menudo en lugar de "característica".

Prueba funcional Las pruebas funcionales verifican el comportamiento esperado del sistema dado un conjunto de entradas y/o acciones.

Greenfield Los proyectos Greenfield son nuevos proyectos de desarrollo de aplicaciones que comienzan desde cero sin una base de código existente. No hay restricciones, por lo que los equipos de desarrollo tienen muchas opciones abiertas.

Entorno de desarrollo integrado (IDE) Un entorno de desarrollo integrado, o IDE, es un conjunto de herramientas que admiten la programación y las pruebas.

Por lo general, incluye un editor, un compilador o un depurador de intérpretes, capacidades de refactorización y herramientas de automatización de compilación. Los IDE generalmente permiten la integración con un sistema de control de código fuente y brindan soporte específico del idioma para ayudar con el diseño del código.

Iteración Una iteración es un ciclo de desarrollo corto, generalmente de una a cuatro semanas, al final del cual se puede entregar potencialmente el código listo para producción. Es posible que se necesiten varias iteraciones, cada una de la misma duración, para ofrecer un tema o una epopeya completa. Algunos equipos en realidad lanzan el código para producción en cada iteración, pero incluso si el código no se publica, está listo para publicarse.

Java Messaging Service (JMS) La API Java Messaging Service (JMS) es un estándar de mensajería que permite que los componentes de aplicaciones basados en Java 2 Platform, Enterprise Edition (J2EE) creen, envíen, reciban y lean mensajes.

Sistema heredado Un sistema heredado es aquel que no tiene ninguna (o pocas) pruebas de regresión automatizadas. Introducir cambios en el código heredado o refactorizarlo puede resultar arrriesgado porque no existen pruebas para detectar cambios no deseados en el comportamiento del sistema.

Extensiones multipropósito de correo de Internet (MIME) Las extensiones multipropósito de correo de Internet, o MIME, extienden el formato del correo de Internet para permitir mensajes no textuales, cuerpos de mensajes de varias partes y mensajes de texto y encabezados que no sean US-ASCII.

Objeto simulado Un objeto simulado simula las respuestas de un objeto existente. Ayuda a diseñar y probar interacciones entre objetos, reemplazando un componente real para que una prueba pueda verificar sus resultados indirectos.

Product Backlog Product Backlog es un término de Scrum para la lista maestra priorizada de todas las funcionalidades deseadas en el producto. Este trabajo pendiente crece con el tiempo a medida que la organización piensa en nuevas funciones que pueden necesitar.

Product Owner Product Owner es un término de Scrum para la persona responsable de priorizar el backlog del producto o las historias. Por lo general, es alguien que desempeña una función de marketing o un experto empresarial clave involucrado en el desarrollo.

Aseguramiento de la calidad (QA) El aseguramiento de la calidad del equipo, o QA, se puede definir como acciones tomadas para garantizar el cumplimiento de un estándar de calidad. En el desarrollo de software, el término "equipo de control de calidad" se utiliza a menudo para referirse al equipo que realiza las pruebas de software. Los equipos de prueba (ver Equipo de prueba) brindan a las partes interesadas información relacionada con la calidad del producto de software. Realizan actividades para aprender cómo debe comportarse el sistema bajo prueba y verificar que se comporta como se esperaba. En el desarrollo ágil, estas actividades están completamente integradas con las actividades de desarrollo. Los evaluadores suelen formar parte del equipo de desarrollo junto con todos los demás involucrados en el desarrollo del software.

Código de producción El código de producción es el código del sistema que se utiliza o se utilizará en producción, a diferencia del código que se escribe para probarlo. El código de prueba invoca u opera sobre el código de producción para verificar su comportamiento.

Refactorización Refactorización es cambiar el código, sin cambiar su funcionalidad, para hacerlo más fácil de mantener, más fácil de leer, más fácil de probar o más fácil de extender.

Prueba de regresión Una prueba de regresión verifica que el comportamiento del sistema bajo prueba no haya 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 se pasan las pruebas, pasan a formar parte de un conjunto de pruebas de regresión, 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 del producto que potencialmente puede lanzarse a producción. El candidato a la liberación puede someterse a pruebas adicionales o complementarse con documentación u otros materiales.

Retorno de la inversión (ROI) El retorno de la inversión, o ROI, es un término tomado del mundo de las inversiones financieras y es una medida de la eficiencia de una inversión. El ROI se puede calcular de diferentes maneras, pero es básicamente 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 una prueba, comparado con el costo de producir y mantener esa prueba o actividad.

SOAP SOAP es un protocolo para intercambiar mensajes basados en XML a través de redes, normalmente utilizando HTTP/HTTPS. Constituye la capa básica de la pila de protocolos de servicios web y proporciona un marco de mensajería básico sobre el que se pueden construir capas abstractas. Un patrón de mensajería SOAP común es el patrón de llamada a procedimiento remoto (RPC), en el que el nodo de la red del cliente envía un mensaje de solicitud al nodo del servidor y el servidor envía inmediatamente una respuesta al cliente.

Historia Una historia de usuario es una breve descripción de una funcionalidad contada desde la perspectiva del usuario que es valiosa para el usuario o para el equipo del cliente. Las historias se escriben tradicionalmente en fichas. La tarjeta normalmente contiene una descripción de una línea de la característica. Por ejemplo, "Como comprador, puedo poner artículos en mi carrito de compras para poder pagarlos más tarde" es una historia. Las tarjetas solo se pueden utilizar en combinación con conversaciones posteriores entre el equipo del cliente y el equipo de desarrollo y alguna verificación de que la historia se ha implementado mediante la escritura y la ejecución de pruebas.

Prueba de historia Una prueba de historia define el comportamiento esperado del código que entregará la historia. Las pruebas de historia pueden estar orientadas al negocio, especificando los requisitos funcionales, o orientadas a la tecnología, como pruebas de seguridad o rendimiento. Estas pruebas se utilizan para guiar el desarrollo y verificar el código entregado. La mayoría de los profesionales ágiles utilizan el término "prueba de historia" como sinónimo de "prueba de aceptación", aunque el término "prueba de aceptación" podría usarse para pruebas que verifican el comportamiento en un nivel superior al de una historia.

Story Board El story board, también llamado tablero de tareas, se utiliza para realizar un seguimiento del trabajo que realiza el equipo durante una iteración. Para cada historia se escriben tarjetas de tareas, que pueden tener colores coordinados según el tipo de tarea. Estas tarjetas, junto con algún tipo de señal visual, proporcionan un mecanismo sencillo para ver el estado actual del progreso de una iteración. Puede utilizar columnas o

Etiquetas adhesivas Etiquetas adhesivas de diferentes colores en tarjetas para diferentes estados, como "Por hacer", "Trabajo en progreso", "Verificar" y "Listo". El guión gráfico puede ser un tablero físico en una pared o un tablero virtual en línea.

Tarea Las tareas son piezas de trabajo necesarias para terminar una historia. Una tarea puede ser una acción necesaria para implementar una pequeña parte de una historia, o puede ser para construir una parte de infraestructura o realizar pruebas que abarquen más de una historia. Generalmente debería representar un día o menos de trabajo.

Deuda Técnica Ward Cunningham fue el primero en introducir esta metáfora. Cuando un equipo produce software sin utilizar buenas prácticas como TDD, integración continua y refactorización, puede incurrir en deuda técnica. Al igual que la deuda financiera, la deuda técnica acumula intereses que le costarán más al equipo en el futuro. A veces esta deuda puede valer la pena, como por ejemplo para aprovechar una oportunidad de negocio repentina. Sin embargo, normalmente la deuda técnica se agrava y reduce la velocidad del equipo. Se puede producir cada vez menos valor empresarial en cada iteración porque el código carece de una red de seguridad de pruebas de regresión automatizadas o se ha vuelto difícil de entender y mantener.

Prueba doble Una prueba doble es cualquier objeto o componente que se instala en lugar del componente real con el propósito expreso de ejecutar una prueba. Los dobles de prueba incluyen objetos ficticios, objetos simulados, resguardos de prueba y objetos falsos.

Desarrollo basado en pruebas (TDD) En el desarrollo basado en pruebas, el programador escribe y automatiza una pequeña prueba unitaria antes de escribir el pequeño fragmento de código que hará que la prueba pase. El código de producción está diseñado para funcionar una prueba a la vez.

Desarrollo de prueba primero En el desarrollo de prueba primero, las pruebas se escriben antes del código de producción correspondiente, pero el código no necesariamente está diseñado para funcionar una prueba a la vez. Las pruebas de clientes o de historias se pueden utilizar en el desarrollo de pruebas primero, así como en pruebas unitarias.

Trozo de prueba Un trozo de prueba es un objeto que reemplaza un componente real que necesita el sistema bajo prueba con un objeto específico de la prueba que alimenta las entradas indirectas deseadas al sistema bajo prueba. Esto permite que la prueba verifique la lógica independientemente de los otros componentes.

Equipo de prueba Un equipo de prueba realiza actividades que ayudan a definir y posteriormente verificar el comportamiento deseado del sistema bajo prueba. El equipo de prueba proporciona información a las partes interesadas sobre la calidad externa del sistema, los riesgos que pueden estar presentes y las posibles estrategias de mitigación de riesgos. En ágil de-

desarrollo, estas actividades están completamente integradas con las actividades de desarrollo. Los evaluadores suelen formar parte del equipo de desarrollo junto con todos los demás involucrados en el desarrollo del software.

Probador Un probador proporciona información a las partes interesadas sobre el software que se está desarrollando. Un evaluador ayuda a los clientes a definir requisitos funcionales y no funcionales y criterios de calidad, y ayuda a convertirlos en pruebas que guían el desarrollo y verifican el comportamiento deseado. Los evaluadores realizan una amplia variedad de actividades relacionadas con la entrega de software de alta calidad, como la automatización de pruebas y las pruebas exploratorias. En el desarrollo ágil, todos los miembros del equipo de desarrollo realizan actividades de prueba. Los miembros del equipo que se identifican como evaluadores trabajan en estrecha colaboración con otros miembros de los equipos de desarrolladores y clientes.

Tema Un tema es lo mismo que una epopeya o una característica. Es una pieza de funcionalidad descrita por el cliente y colocada en la cartera de pedidos del producto para dividirla en historias dimensionadas y estimadas.

Prueba unitaria Una prueba unitaria verifica el comportamiento de una pequeña parte del sistema general. Puede ser tan pequeño como un único objeto o método que sea consecuencia de una o más decisiones de diseño.

Velocidad La velocidad de un equipo de desarrollo es la cantidad de valor que ofrece en cada iteración, medida en puntos de la historia, días ideales u horas. Generalmente, sólo se incluyen en la velocidad las historias completadas. Velocity es útil para la empresa a la hora de planificar funciones y lanzamientos futuros. Los equipos ágiles utilizan su velocidad en la iteración anterior para ayudar a determinar la cantidad de trabajo que pueden realizar en la siguiente iteración.

Lenguaje de descripción de servicios web (WSDL) El lenguaje de descripción de servicios web (WSDL) es un formato XML para describir servicios de red como un conjunto de puntos finales que operan en mensajes que contienen información orientada a documentos o a procedimientos.

Esta página se dejó en blanco intencionalmente.

BIBLIOGRAFÍA

LIBROS, ARTÍCULOS, ARTÍCULOS Y PUBLICACIONES EN BLOG

Alianza ágil. "Principios detrás del Manifiesto Ágil", www.agilemanifesto.org/principles.html, 2001.

Alles, Micah, David Crosby, Carl Erickson, Brian Harleton, Michael Marsiglia, Greg Pattison y Curt Stienstra. "Presenter First: Organizing Complex GUI Applications for Test-Driven Development", Agile 2006, Minneapolis, MN, julio de 2006.

Ambler, Scott. Técnicas ágiles de bases de datos: estrategias efectivas para el desarrollador de software ágil, Wiley, 2003.

Astels, David. Desarrollo basado en pruebas: una guía práctica, Prentice Hall, 2003.

Bach, James. "Explicación de las pruebas exploratorias", www.satisfice.com/articles/et-articulo.pdf, 2003.

Bach, Jonatán. "Gestión de pruebas basada en sesiones", Revista de ingeniería de calidad y pruebas de software, noviembre de 2000, www.satisfice.com/articles/sbtm.pdf.

Beck, Kent. Explicación de la programación extrema: acepte el cambio, Addison-Wesley, 2000.

Beck, Kent y Andrés, Cynthia. Explicación de la programación extrema: acepte el cambio. Segunda edición, Addison-Wesley, 2004.

Berczuk, Stephen y Brad Appleton. Patrones de gestión de la configuración de software: trabajo en equipo eficaz, integración práctica, Addison-Wesley, 2003.

- Boltón, Michael. "Pruebas sin mapa", Better Software, enero de 2005, www.developsense.com/articles/Testing%20Without%20A%20Map.pdf.
- Bos, Erik y Christ Vriens. "An Agile CMM", en Extreme Programming and Agile Methods–XP/Agile Universe 2004, 4th Conference on Extreme Programming and Agile Methods, Calgary, Canadá, 15 al 18 de agosto de 2004, Proceedings, ed. Carmen Zannier, Hakan Erdoganmus, Lowell Lindstrom, págs. 129–138, Springer, 2004.
- Boutelle, Jonathan. "Pruebas de usabilidad para el desarrollo ágil", www.jonathanboutelle.com/mt/archives/2005/08/usability_testi_1.html, 2005.
- Marrón, Tito. "La (falta de) pruebas en espiral de la muerte", <http://ivory.idyll.org/blog/mar-08/software-quality-death-spiral.html>, 2008.
- Buwalda, Hans. "Telenovela Testing", revista Better Software, febrero de 2004, www.logigear.com/resources/articles_lg/soap_operate_testing.asp.
- Clark, Mike. Automatización de proyectos pragmáticos: cómo crear, implementar y monitorear aplicaciones Java, The Pragmatic Programmers, 2004.
- Cohn, Mike. Historias de usuarios aplicadas al desarrollo de software ágil, Addison-Wesley, 2004.
- Cohn, Mike. Estimación y planificación ágiles, Prentice Hall, 2005.
- Crispin, Lisa y Tip House. Prueba de programación extrema, Addison-Wesley 2002.
- Crispín, Lisa. Artículos "Contratación de un probador ágil", "Una estrategia de selección de herramientas ágiles para herramientas de prueba web", "Impulso de la calidad del software: cómo el desarrollo basado en pruebas impacta la calidad del software", <http://lisa.crispin.home.att.neto>.
- DeMarco, Tom y Timothy Lister. Gestión de riesgos en proyectos de software, Dor-set House, 2003.
- Derby, Esther y Larsen, Diana. Retrospectivas ágiles: cómo hacer grandes los buenos equipos, estantería pragmática, 2006.
- Derby, Esther y Rothman, Johanna. Detrás de puertas cerradas: secretos de una gran gestión, Pragmatic Bookshelf, 2006.

De Souza, Ken. Blog "Un probador disfrazado de desarrollador", <http://kendesouza.blogspot.com>.

Dustin, Elfriede, Chris Wysopal, Lucas Nelson y Dino Dia Zovi. El arte de las pruebas de seguridad del software: identificación de fallas de seguridad del software, Symantec Press, 2006.

Dustin, Elfriede. "Teamwork Tackles the Quality Goal", Software Test & Performance, volumen 2, número 200, marzo de 2005.

Duvall, Paul, Steve Matyas y Andrew Glover. Integración continua: mejora de la calidad del software y reducción del riesgo, Addison-Wesley, 2007.

Eckstein, Jutta. Desarrollo ágil de software a gran escala: sumergirse en lo profundo, Casa Dorset, 2004.

Evans, Eric. Diseño basado en dominios: abordar la complejidad en el corazón del software, Addison-Wesley, 2003.

Plumas, Michael. Trabajar eficazmente con código heredado, Prentice Hall, 2004.

Freeman, Steve y Nat Pryce. "Objetos simulados", www.mockobjects.com.

Fowler, Martín. "Integración continua", <http://martinfowler.com/articles/IntegracionContinua.html>, 2006.

Fowler, Martín. "Aplicación estranguladora", [www.martinfowler.com/bliki/AplicaciónEstranguladora.html](http://martinfowler.com/bliki/AplicacionEstranguladora.html), 2004.

Fowler, Martin, "TechnicalDebt", <http://martinfowler.com/bliki/TechnicalDebt.html>, 2003.

Gärtner, Markus, Blog, <http://blog.shino.de>.

Galeno, Roberto. Finales del software: eliminación de defectos, control de cambios y cuenta atrás para la entrega a tiempo, Dorset House, 2005.

Ghiorghiu, Grig. "Rendimiento versus carga versus pruebas de estrés", <http://agiletesting.blogspot.com/2005/02/rendimiento-vs-load-vs-stress-testing.html>, 2005.

Ghirghiu, Grig. Blog "Pruebas ágiles", <http://agiletesting.blogspot.com>.

Agar, Jon. Documentos de prueba de software, www.swtesting.com/hagar_papers_index.html.

Hendrickson, Elisabeth. "Desarrolladores de pruebas, probadores de desarrolladores", <http://testobsessed.com/2007/01/17/tester-developers-developer-testers/>, 2007.

Hendrickson, Elisabeth. "Hoja de referencia de heurística de prueba", <http://testobsessed.com/wordpress/wp-content/uploads/2007/02/testheuristicscheatsheetv1.pdf>, 2007.

Hendrickson, Elisabeth. "Frame-works/herramientas de automatización de pruebas ágiles", <http://testobsessed.com/2008/04/29/agile-friendly-test-automation-toolsframeworks>, 2008.

Highsmith, Jim. Gestión ágil de proyectos: creación de productos innovadores, Addison-Wesley, 2004.

Hunt, Andrew y David Thomas. El programador pragmático: de viajero a maestro, Addison-Wesley, 1999.

Kaner, Cem, James Bach y Bret Pettichord. Lecciones aprendidas en pruebas de software, Wiley, 2001.

Kerth, normando. Retrospectivas de proyectos: manual para revisiones de equipos, Dorset House, 2001.

Kniberg, Henrik. "Cómo ponerse al día con la automatización de pruebas", <http://blog.crisp.se/henrikkniberg/2008/01/03/1199386980000.html>, 2008.

Kniberg, Henrik. Scrum y XP desde las trincheras, Lulu.com, 2007.

Koenig, Dierk, Andrew Glover, Paul King, Guillaume Laforge y Jon Skeet. Groovy en acción, Publicaciones Manning, 2007.

Kohl, Jonathan. "Man and Machine", revista Better Software , diciembre de 2007.

Kohl, Jonathan. Blog y artículos, www.kohl.ca/.

Louvion, Christophe. Blog, www.runningagile.com.

Manns, Mary Lynn y Linda Rising. Cambio sin miedo: patrones para introducir nuevas ideas, Addison-Wesley, 2004.

Marick, Brian. Secuencias de comandos diarias con Ruby: para equipos, evaluadores y para usted, Estantería pragmática, 2007.

Marick, Brian, "Mi proyecto de pruebas ágiles", www.exampler.com/old-blog/2003/21/08/, 2003.

Marick, Brian. "Una alternativa al TDD orientado a las empresas", www.exampler.com/blog/categoría/aa-ftt, 2008.

Marick, Brian. Blog y artículos sobre pruebas ágiles, <http://exampler.com>.

Marcano, Antony. Blog, www.testingreflections.com.

Meszaros, Gerard. Patrones de prueba de XUnit: código de prueba de refactorización, Addison-Wesley, 2007.

Meszaros, Gerard y Janice Aston. "Agregar pruebas de usabilidad a un proyecto ágil", Agile 2006, Minneapolis, MN, 2006, <http://papers.gerardmeszaros.com/AgileUsabilityPaper.pdf>.

Meszaros, Gerard, Ralph Bohnet y Jennitta Andrea. "Pruebas de regresión ágil mediante grabación y reproducción", XP/Agile Universe 2003, Nueva Orleans, LA, 2003, <http://agileregressiontestpaper.gerardmeszaros.com>.

Meszaros, Gerard. "Uso de Storyotypes para dividir historias de XP infladas", <http://storyotypespaper.gerardmeszaros.com>.

Mugridge, Rick y Ward Cunningham. Apto para el desarrollo de software: marco para pruebas integradas, Prentice Hall, 2005.

Newkirk, James y Alexei Vorontsov. Desarrollo basado en pruebas en Microsoft .NET, Microsoft Professional, 2004.

Nielsen, Jacob. "Presupuestos de tiempo para sesiones de usabilidad", www.useit.com/alertabox/usability_sessions.html, 2005.

Norte, Dan. "Presentación de BDD", <http://dannorth.net/introduciendo-bdd>, 2006.

- Patterson, Kerry, Joseph Gernny, Ben McMillan, Al Switzler y Stephen R. Grupo. *Conversaciones cruciales: herramientas para hablar cuando hay mucho en juego* McGraw-Hill, 2002.
- Patton, Jeff. "Pruebe el software antes de codificar", StickyMinds.com, agosto de 2006, www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&ObjectId=11104.
- Patton, Jeff. "Diseño y desarrollo de productos ágiles y holísticos", www.agileproductdesign.com/blog/agile_product_development.html, 2006.
- Polonia, Andy. "El cliente perfecto", www.pols.co.uk/archives/category/testing, 2008.
- Enagua, Bret. "Automatización de pruebas caseras", www.io.com/~wazmo/articulos/homebrew_test_automation_200409.pdf, 2004.
- Enagua, Bret. "Siete pasos para probar el éxito de la automatización", www.io.com/~wazmo/papers/seven_steps.html, 2001.
- Poppendieck, María y Tom Poppendieck. *Implementación del desarrollo de software ajustado: del concepto al efectivo*, Addison-Wesley, 2006.
- Poppendieck, María y Tom Poppendieck. *Desarrollo de software ajustado: un conjunto de herramientas ágil*, Addison-Wesley, 2003.
- Rainsberger, JB Recetas JUnit: métodos prácticos para pruebas de programadores, Publicaciones Manning, 2004.
- Rasmusson, Jonathan. "Introducción de XP en proyectos totalmente nuevos: lecciones aprendidas", IEEE Software, 2003, <http://rasmusson.files.wordpress.com/2008/01/s3021.pdf>.
- Robbins, Stephen y Tim Judge. *Fundamentos del comportamiento organizacional*, novena edición, Prentice Hall, 2007.
- Schwaber, Ken. *Gestión ágil de proyectos con Scrum*, Microsoft Press, 2004.
- Costa, James y Shane Warden. *El arte del desarrollo ágil*, O'Reilly Media, 2007.

Soni, Mukesh. "Prevención de defectos: reducción de costos y mejora de la calidad", iSixSigma, <http://software.isixsigma.com/library/content/c060719b.asp>.

Resumen, Megan. "Caja de zapatos 'Shout-Out': aumentar la moral del equipo", <http://megansumrell.wordpress.com/2007/08/27/shout-out-shoebox-boosting-team-morale>, 2007.

Sutherland, Jeff, Carsten Ruseng Jakobsen y Kent Johnson. "Scrum y CMMI Nivel 5: La poción mágica para los guerreros del código", Agile 2007, Washington, DC, 2007, <http://jeffsutherland.com/scrum/>
Sutherland-ScrumCMMI6pages.pdf.

Tabaka, Jean. Explicación de la colaboración: habilidades de facilitación para líderes de proyectos de software, Addison-Wesley, 2006.

Tomás, Mike. "Strangling Legacy Code", revista Better Software , octubre de 2005, http://samoht.com/wiki_downloads/StranglingLegacyCodeArticle.pdf.

Tholfsen, Mike. "El ascenso de los campeones del cliente", STAREAST, 7 al 9 de mayo de 2008.

Voris, Juan. Pantallas ADEPT AS400 para pruebas y prototipos externos, www.AdeptTesting.org.

Despierta, Bill. "Gráfico de radar XP", <http://xp123.com/xplor/xp0012b/index.shtml>, 2001.

Vriens, Cristo. "Certifying for CMM Level 2 and ISO9001 with XP@Scrum", en ADC 2003: Actas de la Conferencia de Desarrollo Ágil, 25–28 de junio de 2003, Salt Lake City, UT, EE. UU., 120–124, IEEE, 2003.

REFERENCIAS DE HERRAMIENTAS

Marco de prueba de la GUI de Java de Abbot, <http://abbot.sourceforge.net/doc/descripción general.shtml>.

Adzik, Gojko. DbFit: desarrollo de bases de datos basadas en pruebas, <http://gojko.net/fitness/dbfit/>.

Peleé, Danny. "Lista de herramientas de prueba", <http://testingfaqs.org>, 2008.

Canoo WebTest, herramienta de código abierto para pruebas automatizadas de aplicaciones web, <http://webtest.canoo.com>.

easyb, Marco de desarrollo basado en el comportamiento para la plataforma Java, www.easyb.org/.

Fit, Marco para pruebas integradas, <http://fit.c2.com>.

JUnit, Recursos para el desarrollo basado en pruebas, www.junit.org.

JUnitPerf, decoradores de pruebas JUnit para pruebas de rendimiento y escalabilidad, <http://clarkware.com/software/JUnitPerf.html>.

FitNesse, Wiki independiente totalmente integrado y marco de pruebas de aceptación, www.fitnesse.org.

Hower, Rick, información sobre herramientas de prueba y control de calidad del software, www.softwareqatest.com/qatls1.html.

NUnit, marco de pruebas unitarias para lenguajes .NET, <http://nunit.org/index.php>.

Herramientas de prueba de software de código abierto, noticias y debates, www.opensourcetesting.org/.

RpgUnit, Marco de pruebas de regresión de RPG, www.RPGUnit.org.

Selenium, sistema de prueba de aplicaciones web, <http://selenium.openqa.org>.

SOAPUI, herramienta de prueba de servicios web, www.soapui.org.

Gestión de configuración de origen, <http://better-scm.berlios.de>.

Subversion, sistema de control de versiones de código abierto, <http://subversion.tigris.org>.

Marcos de pruebas unitarias. http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks.

Watir, Pruebas de aplicaciones web en Ruby, <http://wtr.rubyforge.org>, <http://watircraft.com>.

ÍNDICE

- Una herramienta de prueba Abbot
- GUI, 127 pruebas de aceptación. Ver también Pruebas orientadas a empresas definición, 501
- Ejemplo de sistema de monitoreo remoto de datos, 245
- UAT (prueba de aceptación del usuario) en comparación con 130
- Pruebas ad hoc, 198
- Adaptabilidad, habilidades y, 39–40
- ADEPT (Pantallas AS400 para exteriores Creación de prototipos y pruebas), 117–118
- Clientes con claridad avanzada hablando con una sola voz, 373–374 determinar el tamaño de la historia, 375–376 recopilar todos los puntos de vista sobre los requisitos, 374–375 descripción general de, 140–142, 373 Desventajas de la preparación anticipada, 373 cuánto se necesita, 372–373
- Desarrollo ágil Manifiesto ágil y, 3–4 barreras para. Consulte Barreras para adoptar la orientación del equipo de desarrollo ágil de, 6
- Estimación y planificación ágiles (Cohn), 331, 332 Manifiesto ágil enfoque en las personas, 30 declaración de, 4 declaraciones de valor en, 21
- Principios ágiles. Ver Principios, para ágil probadores
- Probadores ágiles. Ver también Probadores mentalidad de prueba ágil, 482–483 definición, 4 dar a todos los miembros del equipo el mismo peso, 31 contratación, 67–69 qué son, 19–20
- Definición de prueba ágil, 6 como mentalidad, 20–21 qué queremos decir, 4–7 valores ágiles, 3–4 FIT IssueTrack de Alcea, 84 pruebas Alpha, 466–467 ant, 284 como herramienta de compilación, 126 compilaciones continuas y 175, 291 AnthillPro, 126 ANTS Profiler Pro, 234 Apache JMeter. Consulte las herramientas de prueba funcional de la capa API de JMeter, 168–170 Fit y FitNesse, 168–170 descripción general de, 168 pruebas de servicios web, 170 automatización de pruebas de API, 282 descripción general de, 205–206 API (interfaces de programación de aplicaciones), 501
- Appleton, Brad, 124
- Solicitud bajo prueba (AUT), 246

- Pruebas de
integración de aplicaciones con aplicaciones externas,
459
- Ejemplo de sistema de monitoreo remoto de datos,
242–243
- Enfoque
incremental de arquitectura para las pruebas,
114 capas, 116
pruebas del cuadrante 1 y 99
escalabilidad y 104, 221
comprobables, 30, 115, 182, 184, 267
pantallas AS400 para pruebas y prototipos externos
(ADEPT), 117–118 supuestos,
ocultos Respuesta de los
evaluadores ágiles, 25 fallas en la
detección, 32 preguntas
que descubren, 136 peores
escenarios y, 334 Actitud, mentalidad
de prueba
ágil, 482–483 barreras para adoptar el
desarrollo ágil, 48 versus habilidades, 20 Auditorías,
cumplimiento
de la auditoría. requisitos,
89–90
- AUT (aplicación bajo prueba), 143, 225, 246,
317
- Autorización, pruebas de seguridad y, 224
- Factores clave de éxito de las pruebas
de regresión automatizadas,
484 candidatos de liberación y 458
como red de seguridad, 261–262
- Listas de pruebas automatizadas, alternativas de planes de pruebas,
353–354
- Flujo de código
de automatización y,
269 de implementación,
232 impulsando el desarrollo con, 262–263 de
pruebas exploratorias, 201 miedo
a, 269–270
retroalimentación de, 262
liberando a las personas para otros trabajos, 259–
261 de estructura de prueba funcional, 245–247
hogar -prueba elaborada, 175
inversión requerida, 267–268
- curva de aprendizaje, 266–267
código heredado y, 269
mantenibilidad y, 227–228 pruebas
manuales versus, 258–259
obstáculos para, 264–265
viejos hábitos y, 270
descripción general
de, 255 actitud de los programadores con respecto a,
265–266
razones para, 257–258
responder al cambio y, 29
ROI y 264 tarjetas
de tareas y 394–395 capacidad
de prueba y 149–150 pruebas
como documentación, 263–264
- Estrategia de automatización
prácticas de codificación ágil y, 303–304
aplicación de una herramienta a la vez, 312–313
herramientas de generación de datos,
304–305 acceso a bases de datos y,
306–310 diseño y mantenimiento y, 292–294
desarrollo, 288–289
identificación de requisitos de herramientas , 311–
312 implementación, 316–319
enfoque iterativo, 299–300
manténgalo simple, 298–299
aprender haciendo, 303
gestión de pruebas automatizadas, 319
enfoque de múltiples capas, 290–292
organización de resultados de pruebas,
322–324 organización de
pruebas , 319–322
descripción general
de, 273 principios, 298 herramientas de grabación/
reproducción y, 294, 296–297 comenzando con el área de mayor dolor,
289–290
tomarse el tiempo para hacerlo bien, 301–303
pirámide de automatización de pruebas, 276–
279 categorías de pruebas,
274–276 selección de herramientas, 294–
298, 313–316 comprender el propósito de las pruebas y,
310–311 qué se puede automatizar, 279–
285 lo que podría ser difícil de automatizar,
287–288

- qué no debe automatizarse, 285–287 enfoque de equipo completo, 300–301
- Herramientas de automatización, 164–177
- Herramientas de prueba funcional de capa API, 168–170
 - compilaciones y 126
 - Herramientas de prueba de GUI, 170–176 descripción general de, 164–165
 - herramientas de prueba a nivel de unidad, 165–168 herramienta de prueba de servicios web, 170
- B
- Bach, James, 195, 200, 212
- Bach, Jonatán, 201
- Pruebas de back-end
- detrás de la GUI, 282 pruebas que no son de UI, 204–205
- Bambú, 126
- Barreras para adoptar el desarrollo ágil, 44–49 roles múltiples o conflictivos, 45 diferencias culturales entre roles, 48–49 falta de capacitación, 45 falta de comprensión de conceptos ágiles, 45–48 pérdida de identidad, 44–45 descripción general de, 44 experiencias y actitudes pasadas, 48
- Técnica de referencia de prueba de ruptura de líneas de base, 363 rendimiento, 235–237
- Archivos
- por lotes, 251 procesamiento, 345 proceso de programación, 182
- BDD (desarrollo impulsado por el comportamiento)
- herramienta easyb, 166–168
 - herramientas para pruebas del cuadrante 1, 127
- Beck, Kent, 26, 99
- Benander, Marcos, 51
- Evaluación comparativa, 237
- Berczuk, Stephen, 124
- Pruebas beta, 466–467
- Panorama
- general en el que se centran los evaluadores ágiles, 23 pruebas y ejemplos de alto nivel, 397–402
- factores clave de éxito, 490–491 peligro de olvido, 148 pruebas de regresión y, 434
- Bolton, Michael, 195
- Jefe, Erik, 114
- Condiciones de borde
- Pruebas de API y, 205
 - automatización y, 11
 - herramientas de generación de datos y, 304 identificación de variaciones de prueba, 410 escritura de casos de prueba para, 137
- Boyer, Erika, 140, 163, 372, 432
- Automatización de
- lluvia de ideas que brinda a los evaluadores un mejor trabajo, 260
 - antes de la iteración, 370, 381
 - cuadrantes como marco para, 253 tomarse el tiempo para, 301
 - evaluadores, 121 técnica de referencia de prueba de ruptura, 363 navegadores, pruebas de compatibilidad y 230 límites de presupuesto, 55 seguimiento de errores. Consulte Errores de seguimiento de
 - defectos. Consulte Defectos
- Automatización de compilación, 280–282 compilaciones candidatas a lanzamiento desafiantes, 473 definición, 501 incremental, 178–179 aceleración, 118–119 Herramientas de automatización de compilación, 126, 282 Patrón de compilación/operación/ comprobación, 180
- Herramientas de compilación, 126 BuildBeat, 126 Analistas de negocios, 374 rol acuerdo sobre requisitos, 428, 430
- lenguaje común y, 134, 291, 414 en el equipo del cliente, demostración de 6 a 7 iteraciones y, 443 lenguaje de, 291
- Poder de Tres y, 482 herramientas orientadas a, 134

- Pruebas de cara al negocio
Pruebas ágiles como,
6 cuadrantes 2 y 3, 97–98
pruebas de cara a la tecnología en comparación con, 120
pruebas de cara al negocio, criticando el producto
(Cuadrante 3), 189–215
pruebas de aceptación, 245
pruebas de API, 205–206
demostraciones, 191–192
herramientas de emulador, 213–
214 pruebas de un extremo a otro,
249–250 pruebas exploratorias, 195–202, 248–249
generación de datos de prueba,
212 pruebas de GUI,
204 herramientas de monitoreo,
212–213 descripción general
de, 189–191 informes,
208–210 pruebas de escenarios,
192–195 pruebas basadas en sesiones,
200–201 pruebas de configuración,
211–212 herramientas de
simulación, 213 herramientas para pruebas
exploratorias, 210–211 pruebas
de usabilidad, 202–204 pruebas de
aceptación del usuario, 250
documentación de usuario, 207–
208 pruebas de servicios web, 207 pruebas
orientadas al negocio, equipo
de soporte (Cuadrante 2), 129–
151 claridad avanzada, 140–142 pruebas
funcionales de automatización, 245–247
lenguaje común y 134–135 condiciones de satisfacción
y, 142–143 finalización,
146–147 impulsar el desarrollo con, 129–132
generar requisitos, 135–140 pruebas
integradas, 248 enfoque
incremental, 144–146 requisitos dilema y,
132–134 efectos en cadena, 143–144 mitigación
de riesgos y, 147–149
capacidad de prueba y automatización,
149–150 kit de herramientas para. Consulte
Pruebas de servicios web del kit de herramientas
(cuadrante 2), 247–248 Impacto
empresarial, 475–476
- Valor empresarial
que agrega valor, 31–33
como objetivo del desarrollo ágil, 5–8, 69, 454 métricas
y, 75 ciclos de
lanzamiento y, 3 rol,
función, patrón de valor empresarial, 155 enfoque de
equipo y, 16
Busse, Mike, 106, 235, 284, 313
Buwalda, Hans, 193
- C
- Datos canónicos, automatización de bases de datos y,
308–309
Prueba web de Canoo
automatizar pruebas de GUI, 184, 186
Conjunto de pruebas de regresión GUI, 291
Pruebas de humo GUI, 300
Herramientas de prueba de
GUI, 174–175 organización de
pruebas y 320 scripts y 320
Editor XML para, 125
- Integración del modelo de madurez de capacidad
(CMMI), 90–91
- Herramienta de captura y reproducción, 267
- Celebrar los éxitos,
implementar cambios y, 50–52 iteraciones,
concluir y, 449–451
- Chandra, Apurva, 377
- Chang, Tae, 53–54
- Cambio
celebrando los éxitos, 50–52 dando
propiedad al equipo, 50 presentando,
49 no es fácil, 56–57
capacidad de respuesta, 28–29
hablando de miedos, 49–50
- Listas de
verificación de preparación
para la publicación, 474 herramientas para obtener ejemplos y requisitos,
156
- CI. Ver Integración continua (CI)
- CI Factory, 126
- CMMI (Integración del modelo de madurez de
capacidad), 90–91

- Coubicación, logística del equipo y, 65–66
- Entrenadores que se adaptan a la cultura ágil y, 40 curva de aprendizaje y, 266 que brindan estímulo, 69 desarrollo de habilidades y, 122 capacitación y, 45–46
- Cockburn, Alistair, 115
- Automatización de código y flujo de código, 269 automatización y código heredado, 269 estrategia de automatización y, 303–304 documentación de, 251 estándares, 227 escritura comprobable, 115
- Cobertura de código, métricas de lanzamiento, 360–364
- Codificación y pruebas, 405–441 agregar complejidad, 407 alternativas para lidar con errores, 424–428 elegir cuándo corregir errores, 421–423 colaborar con programadores, 413–414 lidar con errores, 416–419 decidir qué errores registrar, 420–421 impulsar el desarrollo y, 406 facilitar la comunicación, 429–432 centrarse en una historia, 411–412 identificar variaciones, 410 métricas de iteración, 435–440 medios para registrar errores, 423–424 descripción general de, 405
- Poder de tres para resolver diferencias de punto de vista, 411 pruebas de regresión y, 432–434 recursos, 434–435 evaluación de riesgos, 407–409 como proceso simultáneo, 409–410, 488–489 empezar de forma sencilla, 406, 428–429 hablar con los clientes, 414–415 pruebas que critican el producto, 412–413
- Cohn, Mike, 50, 155, 276, 296, 331, 332
- Colaboración con los clientes, 396–397 factores clave de éxito, 489–490
- con programadores, 413–414 enfoque de equipo completo, 15–16
- Collino, Alejandro, 103, 363
- Comunicación lenguaje común y, 134–135 con el cliente, 140, 396–397
- DTS (Sistema de seguimiento de defectos) y, 83 facilitar, 23–25, 429–432 entrega de productos y, 462–463 tamaño como desafío, 42–43 entre equipos, 69–70 resultados de pruebas, 357–358
- Comparaciones, automatización, 283.
- Pruebas de compatibilidad, 229–230
- Automatización de pruebas de componentes, definición 282, función de soporte 501, 5
- Condiciones de satisfacción de las pruebas de cara al negocio y, definición 142–143, 501–502
- Definición de pruebas basadas en el contexto, 502 cuadrantes y 106–107
- Proceso de construcción continuo notificación de fallo y, 112 comentarios y, 119
- Pruebas de FitNesse y, 357 implementación, 114 integración de herramientas con, 175, 311 control de código fuente y, 124 qué pueden hacer los evaluadores, 121
- Principio de retroalimentación continua, 22
- Principio de mejora continua, 27–28
- Automatización de integración continua (CI), 280–282 como práctica principal, 486–487 instalabilidad y, 231–232 Ejemplo de sistema de monitoreo remoto de datos, 244 pruebas en ejecución y, 111–112
- Conversión, migración de datos y, 460–461
- Prácticas básicas de codificación y pruebas como un solo proceso, 488–489 integración continua, 486–487

- Prácticas básicas, continuación
enfoque incremental, 488 descripción general de, 486
sinergia entre prácticas, 489 gestión de deuda técnica, 487–488 entornos de prueba, 487
Coraje, principios, 25–26, 71
- Credibilidad, construcción, 57 Criticar el negocio del producto frente a las pruebas. Consulte Pruebas orientadas al negocio, críticas del producto (Cuadrante 3)
Pruebas orientadas a la tecnología. Ver Pruebas de cara a la tecnología, criticando el producto (Cuadrante 4)
- CrossCheck, pruebas de servicios web, 170
- CruiseControl, 126, 244, 291 Cambio cultural, 37. Véase también Organizaciones Cunningham, Ward, 106, 168, 506 Expectativas del cliente, impacto en el negocio y 475–476 soporte de producción, 475
- Prueba de cara al cliente. Consulte Pruebas empresariales Atención al cliente, DTS (Sistema de seguimiento de defectos) y 82
equipo de clientes definición, 502
interacción entre los equipos de clientes y desarrolladores, 8 descripción general de, 7
- Pruebas de clientes Pruebas Alfa/Beta, 466–467 definición, 502 descripción general de, 464
UAT (pruebas de aceptación del usuario), 464–466
- Clientses colaborando con, 396–397, 489–490 considerando todos los puntos de vista durante la planificación de la iteración, 388–389
entregando valor a, 22–23
importancia de comunicarse con, 140, 414–415, 444 demostración de iteración, 191–192, 443–444 participación en la planificación de iteraciones, 384–385
- relación con, 41–42 revisando pruebas de alto nivel con, 400 hablando con una sola voz, 373–374
CVS, control de código fuente y, 124
- D
- Datos automatización de la creación o configuración, 284–285 limpieza, 461 conversión, 459–461
planificación de lanzamiento y, 348 tarjetas de tareas de escritura y, 392
Pruebas basadas en datos, 182–183
Fuentes de datos, pruebas, 249
Herramientas de generación de datos, 304–305
Migración de datos, automatización, 310, 460
Bases de datos que evitan el acceso al ejecutar pruebas, 306–310
datos canónicos y automatización, 308–309
mantenibilidad y 228 entrega y actualizaciones de productos, 459–461 datos y automatización similares a los de producción, 309–310
configurar/derribar datos para cada prueba automatizada, 307–308
prueba de migración de datos, 310
De Souza, Ken, 223
Plazos, alcance y 340–341
Descripción general de las métricas de defectos, 437–440 métricas de versión, 364–366
Seguimiento de defectos, 79–86
DTS (Sistema de seguimiento de defectos), 79–83
mantener el enfoque y, 85–86
descripción general de, 79 razones para, 79 herramientas para, 83–85 Sistema de seguimiento de defectos. Ver DTS (Sistema de seguimiento de defectos)
Alternativas de defectos para tratar errores, 424–428
elegir cuándo corregir errores, 421–423

- lidar con errores, 416–419 decidir qué errores registrar, 420–421 medios para registrar errores, 423–424 métricas y, 79
- TDD (desarrollo basado en pruebas) y 490 tarjetas de tareas de escritura y 391–392 tolerancia cero a errores, 79, 418–419
- Entregables
 - entregables de “ajuste y acabado”, 454 no software, 470 descripción general de, 468–470
- Entregando producto
 - Pruebas alfa/beta, 466–467 impacto empresarial y, 475–476 comunicación y, 462–463 expectativas del cliente, 475 pruebas del cliente, 464 conversión de datos y actualizaciones de bases de datos, 459–461 entregables, 468–470 final del juego, 456–457 instalación pruebas, 461–462 integración con aplicaciones externas, 459 pruebas no funcionales y, 458–459 descripción general de, 453 empaquetado, 474–475 tiempo de planificación para las pruebas, 455–456 ciclos de pruebas posteriores al desarrollo, 467–468 soporte de producción, 475 aceptación de lanzamiento criterios, 470–473 gestión de versiones, 470, 474 lanzamiento de producto, 470 entorno de preparación y, 458 pruebas de lanzamiento candidatos, 458
- UAT (pruebas de aceptación del usuario), 464–466
 - ¿Qué pasa si no está listo?, 463–464
 - ¿Qué constituye un producto?, 453–455.
- Demostraciones/demostraciones
 - de una iteración, 443–444 valor para los clientes, 191–192
- Implementación, automatización, 280–282
- Diseñar
 - una estrategia de automatización y, 292–294
 - diseñar teniendo en cuenta las pruebas, 115–118
- Casos de prueba
 - detallados del arte y la ciencia de la escritura, 178 enfoque general y, 148–149
 - diseño con, 401
- Interacción del equipo
 - de desarrolladores entre los equipos de clientes y desarrolladores, 8
 - descripción general de, 7–8
- Desarrollo
 - desarrollo ágil, 3–4, 6 pruebas de conducción automatizadas, 262–263 pruebas de conducción orientadas al negocio, 129–132
 - conducción de codificación, 406 ciclos de prueba posteriores al desarrollo, 467–468
- Picos de desarrollo, 381
- Equipo de desarrollo, herramienta de diferencias 502, 283
- Equipos distribuidos, 431–432
 - sistemas de rastreo de defectos y 82
 - logística física, 66 pruebas de alto nivel en línea para, 399 guiones gráficos en línea para, 357 respuesta al cambio, 29 herramientas basadas en software para obtener ejemplos y requisitos, y 163–164
- Documentación de
 - pruebas automatizadas como fuente de, 263–264
 - problemas y correcciones, 417
 - informes, 208–210 de código de prueba, 251 pruebas como, 402 documentación de usuario, 207–208
- Listo
 - saber cuándo termina una historia, 104–105 de varios niveles, 471–472
- Impulsar el desarrollo con pruebas. Ver TDD (desarrollo basado en pruebas)
- DTS (Sistema de seguimiento de defectos), 80–83
 - beneficios de, 80–82
 - elegir medios para registrar errores, 424 documentar problemas y soluciones, 417

- DTS (Sistema de seguimiento de defectos), continuación
 errores de registro y, 420
 motivos para no usarlo, 82–83
- Dymond, Robin, xxx
- Análisis dinámico, herramientas de prueba de seguridad, 225.
- E
- herramienta de desarrollo basada en el comportamiento easyb, 165–168
- Fácil Mock, 127
- Eclipse, 125, 316
- Casos
 extremos que identifican variaciones, 410 no tienen tiempo para, 112
 comienzan de manera simple y luego agregan complejidad, 406–407
 casos de prueba
 para, 137 Sistema integrado, ejemplo de monitoreo
 remoto de
 datos, 248 Empoderamiento de
 equipos, 44 Herramientas de
 emulador,
 213–214 Pruebas
 ágiles de final
 de juego, 91 iteración, 14 entrega de
 producto y, 456–
 457 lanzamiento y, 327 Extremo
 a finalizar las pruebas, 249–250
Disfrute, principio de, 31 Entorno, entorno de
prueba, 347–348 Épico.
 Consulte también
 Definición de temas, 502
 características que se
 convierten, 502
 iteraciones, 76, 329 planificación,
 252 ePlan Services, Inc., xli, 267
 Errores, pruebas manuales y, 259
 Estimación
 del tamaño de la historia, 332–338 eValid, 234 Patrones
 basados en eventos, patrones de diseño de pruebas, 181
 Scripting diario con Ruby para
 equipos, probadores y usted (Marick), 297, 303
Desarrollo
 basado en ejemplos, 378–380 Ejemplos
 para obtener requisitos, 136–137 herramientas para obtener ejemplos, 152–153
 155–156
- Pruebas ejecutables, 406
- Actividades, características y
 habilidades de las pruebas exploratorias (ET) (Hagar), 198–200
- atributos del probador exploratorio, 201–202
- automatización de, 201
- definición, 502–503 final
- del juego y, 457 explicado
(Bolton), 195–198 pruebas manuales y,
 280 herramientas de monitoreo,
 212 descripción general de,
 26, 195
- Ejemplo de sistema de monitoreo remoto de datos, 248–249
- pruebas basadas en sesiones y, 200–201
- configuración, 211–
 212 simuladores y emuladores, 212–213 pruebas
 que critican el producto, 412–413 herramientas para,
 210–212 herramientas
 para generar datos de prueba, 212 qué no
 debe automatizarse, 286 Calidad externa,
 definición de pruebas de cara al negocio, 99, 131 equipos
 externos,
 43, 457 Programación Extrema.
Ver XP (Programación Extrema)
- Explicación de la programación extrema (Beck), 26
- F
- Comunicación cara a cara, 23–25
- Pruebas de conmutación por error, 232
- Fracaso, coraje para aprender, 25
- Objetos falsos, 115, 118, 306, 502–503
- Tolerancia a fallos, entrega de productos y, 459
- Miedo
 barreras a la automatización, 269–270 cambio
 y, 49–50
- Cambio intrépido (Manns y Rising), 121
- Plumas, Michael, 117, 288
- Características
 defectos vs., 417–418
 definición, 502–503
 centrándose en el valor, 341

- Pruebas
- automatizadas de retroalimentación que proporcionan, 262 principio de retroalimentación continua, 22 enfoque iterativo y, 299–300 factores clave de éxito, 484–486 pruebas de gestión para, 323–324 Pruebas del cuadrante 1 y, 118–119 Entregables de “ajuste y acabado”, 454 Fit (Marco para Prueba Integrada), 134–135
 - Herramientas de prueba funcionales de capa API, 168–169 pirámide de pruebas de automatización y 278FIT IssueTrack, Alcea, 83–84 Ventajas
 - de FitNesse, 163
 - Herramientas de prueba funcional de capa API, 169–170 automatización de pruebas funcionales con, 30, 145 pruebas de cara al negocio con, 154, 178 colaboración y, 164 compilaciones continuas y, 119, 357 verificación de datos con, 287 finalización y, 472 uso alertador de, 122 ejemplos y, 136, 169 comentarios y, 323–324 reglas de análisis de archivos ilustradas con, 205 pruebas funcionales detrás de la GUI, 291, 300 scripts locales y, 305 JUnit en comparación con, 299 palabras clave o palabras de acción para automatizar pruebas, 182–183 pruebas manuales versus automatizadas, 210 demandas de memoria de, 306 pruebas de organización y, 319–320 descripción general de, 168–170 pruebas remotas y, 432 lista “iniciar, detener, continuar”, 446 soporte para herramientas de control de código fuente, 320 pirámide de automatización de pruebas y, 278 tarjetas de prueba y, 389–390 casos de prueba como documentación, 402 diseño y mantenimiento de pruebas, 292
 - capa de base de datos de prueba con 284 historias de prueba, 395 requisitos de trazabilidad y 88 pruebas de aceptación de usuario, 295 wikis y 186
- Carne, Patricio, 377, 440
- Pruebas de
- escenarios de diagramas de flujo y, 194–195 herramientas para obtener ejemplos y requisitos, 160–163
- Fowler, Martin, 117 Marco para pruebas integradas. Ver Fit (Marco para Prueba Integrada)
- Marcos, 90–93 ftptt, 234 analistas
- funcionales, 386 problemas de compatibilidad de pruebas funcionales y, 230 definición, 502–503 pruebas de un extremo a otro, 249–250 capas, 246 pruebas no funcionales en comparación con, 225 Ejemplo de sistema de monitoreo remoto de datos, 245–247
- GRAMO
- Galen, Bob, 455–456, 471
- Gartner, Markus, 395, 476
- Equipos geográficamente dispersos que se enfrentan, 376–378 facilitan la comunicación y, 431–432
- Gheorghiu, Grig, 225–226, 234
- Glover, Andrés, 166
- Pruebas de código de proyectos Greenfield y, definición 116, 502–503
- Estrategia de automatización GUI (interfaz gráfica de usuario) y, 293 flujo de código y, 269 estándares, 227
- Pruebas de humo GUI
- Canoo WebTest y 300 compilaciones continuas y 119 métricas de defectos, 437

Herramientas de prueba de GUI, 170–176

Canoo Web Test, 174–175

herramientas de automatización de pruebas "caseras", 175

herramientas de prueba de código abierto,

172 descripción general de, 170–

171 herramientas de grabación/reproducción, 171–172

Rubí con Watir, 172–174

Selenio, 174

Pruebas de interfaz gráfica de usuario

Pruebas de API, 205–206

automatización, 282–283, 295–296

pirámide de pruebas de automatización y, 278

Pruebas de humo GUI, 119, 300, 437

descripción general de, 204

Pruebas de servicios web, 207

h

Agar, Juan, 198

Compatibilidad

de hardware y, 229 costo de los entornos de prueba, 487 pruebas funcionales y, 230 inversión en automatización y, 267 entorno de producción y, 310 escalabilidad y, 233 infraestructura de prueba, 319 instalación de productos de prueba, 462

Hendrickson, Elisabeth, 203, 315–316

Casos de prueba de alto nivel, 397–402

maquetas, 398–399

descripción general de, 397–

398 revisión con clientes, 400 revisión

con programadores,

400–401

casos de prueba como documentación, 402

Contratación de un evaluador, 67–69

Holzer, Jason, 220, 448

Herramienta de prueba casera

herramientas de automatización, 314

Herramientas de prueba GUI,

175 resultados de

prueba, 323 httperf, 234

Hudson, 126

I

IBM Rational ClearCase, 124

Definición de IDE (entornos de desarrollo integrados), 502–503

herramientas de análisis

de registros, 212 herramientas

para pruebas del cuadrante 1, 124–126 pruebas

de compatibilidad

de pruebas de "ilidad", 229–230 pruebas

de instalación, 231–232 pruebas de

interoperabilidad, 228–229 pruebas de

mantenibilidad, 227–228 pruebas de

confiabilidad, 230–231, 250–251 pruebas de

seguridad, 223–227

Impacto, en todo el sistema, 342

Implementación del desarrollo de software ajustado: desde

Concepto de efectivo (Poppendieck), 74, 416

Enfoque de mejora

para la mejora de procesos, 448–449 principio de mejora continua, 27–28 ideas de mejora a partir de retrospectivas,

447–449

Desarrollo incremental creando

pruebas de forma incremental, 178–179 como

práctica principal, 488

pruebas de "ilidades" y 232

cortes finos, fragmentos pequeños, 144–146

pruebas tradicionales versus ágiles, 12–13

Fichas, registro de errores, 423

Infraestructura

Pruebas del cuadrante 1, 111–112

infraestructura de prueba, 319

planes de prueba y 346–347

pruebas de instalabilidad, 231–232

pruebas de instalación, 461–462

entornos de desarrollo integrados. Ver IDE (Entornos de

Desarrollo Integrados)

Integración, pruebas de

interoperabilidad y, 229 productos

y aplicaciones externas, 459 IntelliJ IDEA, 125

Calidad interna que

mide la calidad

interna del código, 99 cumple con los estándares

del equipo, 366

- Pruebas del cuadrante 1 y, 111
velocidad y, 112
- Pruebas de interoperabilidad, 228–229
- Inversión, automatización que requiere, 267–268
- Estrategia
- de automatización de iteración y definición 299–300, demostración 502–503, vida útil de un probador 443–444 y 327 actividades previas a la iteración. Consulte Actividades previas a la iteración que priorizan historias y, 338 revisión, 415, 435–437 pruebas tradicionales versus ágiles, 12–13 Inicio de la iteración, 383–403 colaboración con los clientes, 396–397 considerando todos los puntos de vista, 385–389 control de la carga de trabajo, 393 pruebas y ejemplos de alto nivel, 397–402 planificación de iteraciones, 383–384 detalles del proyecto de aprendizaje, 384–385 descripción general de, 383 historias comprobables, 393–396 tarjetas de tareas de escritura, 389–392
- Métricas de iteración, 435–440
métricas de defectos, 437–440
medir el progreso con, 435–437 descripción general de, 435
utilidad de, 439–440
- Planificación de iteraciones
considerando todos los puntos de vista, 385–389
control de la carga de trabajo, 393
detalles del proyecto de aprendizaje, 384–385
descripción general de, 383–384 escritura de tarjetas de tareas, 389–392
- Reunión de revisión de iteración, 415
- Resumen de la iteración, 443–451
celebrando éxitos, 449–451 demostración de iteración, 443–444 ideas de mejora, 447–449 retrospectivas, 444–445 ejercicio de “iniciar, detener, continuar” para retrospectivas, 445–447
- ITIL (Infraestructura de Tecnología de la Información) Biblioteca), 90–91
- j
- Compórtate, 165
- JConsola, 234
- Pruebas
- de referencia de rendimiento de JMeter, 235 pruebas de rendimiento, 223, 234, 313
- Definición de JMS (Java Messaging Service), integración 502–503 con aplicaciones externas y, 243 pruebas de fuentes de datos y, 249 JProfiler, 234
- 234 JUnit FitNesse como alternativa para TDD, 299 pruebas funcionales, 176 herramientas de prueba de carga, 234–235 herramientas de prueba unitaria, 126, 165, 291 JUnitPerf, 234 Desarrollo justo a tiempo, 369.
- Véase también Pre -actividades de iteración
- k
- Factores clave de éxito
mentalidad de prueba ágil, 482–483
automatización de pruebas de regresión, 484
enfoque general, 490–491 codificación y pruebas como un solo proceso, 488–489 colaboración con clientes, 489–490 integración continua (CI), 486–487 retroalimentación, 484–486 fundamento de las prácticas básicas, 486 enfoque incremental (porciones finas, trozos pequeños), 488 descripción general de, 481 sinergia entre prácticas, 489 gestión de deuda técnica, 487–488 entornos de prueba, 487 enfoque de equipo completo, 482
- Pruebas basadas en palabras clave, 182–183
- Rey, José, 176
- Base de conocimientos, EDE, 80–81

-
- Kohl, Jonathan, 201, 204, 211
 Rey, Dierk, 320
- I
- Lenguaje, necesidad de común, 134–135
 Arquitectura en capas, 116
 Medidas magras, métricas, 74–75
Estrategia
 de automatización del aprendizaje
 y, 303 principio de mejora continua, 27
Curva de aprendizaje, automatización y, 266–267, 303
Código heredado, 269
Rescate de código heredado (Plumas), 117
CCDE de sistemas
 heredados,
 definición 269, 502–503
 errores de registro y
 pruebas 421, 117
Lecciones aprendidas en pruebas de software (Pettichord),
 485
Sesiones de lecciones aprendidas, 383. Véase
 también
Retrospectivas Procesos ligeros, 73–
 74 **Planes de pruebas ligeros**,
 350 **Pruebas de carga**. Consulte **Pruebas de carga y**
rendimiento
LoadRunner, 234
LoadTest, 234 **Logística física**,
 65–66 **Herramienta**
LogWatch, 212 **Pérdida de identidad**, temor de los
equipos de control de calidad, 44–45 **Louvion, Christophe**, 63
- METRO
- Pruebas de mantenibilidad**, 227–228
Gestión, 52–55 claridad
 avanzada y, 373–374 cambio
 cultural y, 52–54 descripción
 general de, 52
 proporcionar métricas a, 440
Cambios
 culturales de los gerentes, 52–
 54 cómo influir en las pruebas, 122–123
 hablar el idioma del gerente, 55
Manns, María Lynn, 121–122
- Automatización**
 de pruebas manuales versus,
 258–259
 peligro de, 289 **Marcano, Antony**, 83, 426 **Marick, Brian**, 5, 24, 97, 134, 170, 203, 303 **Martin, Micah**, 169 **Martin, Robert C.**, 169
Matrices
 pruebas de alto nivel y, 398–399
 matrices de texto, 350–353
Maven, 126
McMahon, Chris, 260
Tiempo medio entre fallas, pruebas de confiabilidad,
 230
Tiempo medio hasta el fallo, pruebas de fiabilidad, 230
Medios, para registrar errores, 423–424
- Demostraciones de reuniones**,
 71, 192 dispersas geográficamente,
 376 inicio de iteración,
 372 planificación de iteración, 23–24, 244, 331, 384,
 389 revisión de iteración, 71,
 415 planificación previa,
 370–372 planificación de
 lanzamiento, 338, 345
 retrospectiva, 447
 programación, 70 proceso de
 dimensionamiento y, 336–
 337 stand-up, 177, 429, 462
 participación en equipo y, 32 planificación de pruebas, 263
Pérdidas de memoria, 237–238
Pruebas de gestión de la memoria, 237–238
Meszaros, Gerald, 99, 111, 113, 138, 146, 182, 204,
 291, 296, 430
- Métricas**, 74–79
 cobertura de código, 360–
 364 comunicación de, 77–78
 métricas de defectos, 364–366, 437–
 440 métricas de iteración, 435–
 440 que justifican la inversión en automatización,
 268 mediciones ajustadas, 74–
 75 descripción
 general de, 74 aprobación
 pruebas, 358–360 razones para rastrear defectos, 52, 75–77, 82

métricas de lanzamiento, 358	Nessus, escáner de vulnerabilidades, 226 .NET
ROI y, 78–79 qué no	Memory Profiler, 234 NetBeans, 125
hacer con, 77	NetScout, 235
Cartas de radar XP, 47–48	Pruebas no
Hitos, celebración de éxitos, 449–450	funcionales. Consulte también Pruebas
MIME (correo de Internet multipropósito)	orientadas a la tecnología, crítica del producto
Extensiones)	(Cuadrante 4) entrega del
definición, 504	producto y, 458–459 pruebas funcionales en
fuentes de datos de prueba y 249	comparación con, 225 requisitos, 218–219 cuándo
Mapas mentales, 156–158	realizar, 222 North, Dan, 165
Pruebas	NSpec, 165 NUnit, 126, 165
ágiles con mentalidad como,	oh
20–21 factores clave de éxito, 482–483	Oleszkiewicz, Jakub, 418
proactivos, 369–370	Pruebas únicas, 286–287
Fenómeno de la “minicascada”, 46–47	Herramientas de código abierto
Definición de	herramientas de prueba ágiles de código abierto,
objetos simulados,	172–175 automatización y, 314–315
504 alivio de riesgos y 459	Herramientas de prueba GUI, 172
herramientas para implementación, 127	IDE, 124–125
pruebas unitarias y 114	OpenWebLoad, 234
Maquetas que	Sistemas operativos (SO), pruebas de compatibilidad y, 230
facilitan la comunicación y 430 pruebas de alto nivel	Organizaciones, 37–44
y 398–399 historias y 380 herramientas	desafíos del desarrollo ágil, 35 culturas en
para obtener ejemplos	conflicto, 43 relaciones con los
y requisitos.	clientes y, 41–42 descripción general de, 37–38
160	filosofía de calidad, 38–40
Desarrollo impulsado por modelos, 398	tamaño y, 42–43 ritmo sostenible de
Modelos	pruebas y, 40–41
modelos de calidad, 90–93	equipo empoderamiento, 44
Ejemplo de modelado de UI, 399	SO (sistemas operativos), pruebas de compatibilidad y, 230
Herramientas de seguimiento, 212–213, 235	Propiedad, dando propiedad al equipo, 50
Enfoque multicapa, estrategia de automatización,	PÁG
290–292	Embalaje, entrega de productos y 474–475
Extensiones de correo de Internet multipropósito	Revisión de código de
(MIME),	programación en pares y 227
504 fuentes de datos	desarrolladores capacitados en 61
de prueba y 249	
Convenciones de nomenclatura, 227	
Nantes, 126	
Navegación, pruebas de usabilidad y, 204.	
Compórtate, 165	
Neocarga, 234	

- Programación en pares, IDE continuos
y, 125 enfoque de
equipo y, 244 Pruebas en pares,
413 Aprobación de
pruebas, métricas de lanzamiento, 358–360
PerfMon, 235
Perforce, 124
Automatización de pruebas de
rendimiento y carga,
283 líneas de base, 235–
237 pruebas de administración de memoria, 237–
238 descripción
general de, 234 entrega del
producto y, 458 pruebas de
escalabilidad, 233–234
entorno de prueba, 237
herramientas para, 234–235
cuándo realizar, 223 quién realiza la
prueba, 220–221 Rendimiento, recompensas
y, 70–
71 Peligros de olvidar el panorama
general, 148 mentalidad policial de
calidad, 39 la crisis de las
pruebas, 416 esperando la compilación
del martes, 280 realmente no eres parte del
equipo, 32 Perkins, Steve, 156, 159,
373
herramientas de generación de
datos PerlClip, 305 herramientas para
generar datos de prueba, 212
Pruebas de persona, 202–204
Pettichord, Bret, 175, 264, 485 Desarrollo por fases y
cerrado, 73–74, 129 Logística
física, 65–
66 Planificación
anticipada, 43 iteración. Consulte
Planificación de iteraciones, lanzamiento/planificación de
temas. Consulte Pruebas de
planificación de lanzamiento. Consulte Planificación
de pruebas PMO
(Oficina de gestión de proyectos), 440 Pols, Andy, 134
Patrón de puertos y adaptadores (Cockburn),
115 Pruebas posteriores al
desarrollo, 467–468 Errores posteriores a la iteración, 421 Pounder, 234
poder de tres
experto en negocios y, 482
encontrar un lenguaje común, 430 buena
comunicación y, 33, 490 resolución de
problemas y, 24 resolver
diferencias de punto de vista, 401, 411 enfoque de todo
el equipo y, 482
Automatización pragmática de proyectos, 260
Actividades previas a la iteración, 369–
382 avanzan en la
claridad, 373 beneficios de trabajar en historias con anticipación,
370–372
clientes que hablan con una sola voz, 373–374 determinan
el tamaño de la historia, 375–376 evalúan
la cantidad de preparación previa necesaria, 372–373
ejemplos, 378–380
reúnen todos los puntos
de vista sobre los requisitos, 374–375 equipo
geográficamente disperso y,
376–378
descripción general
de, 369 priorización de defectos,
381 mentalidad proactiva, 369–370
recursos, 381
estrategias de prueba y, 380–381
Reunión de planificación previa, 370–372
Principios, prácticas de
automatización de codificación ágil, 303–
304 enfoque iterativo, 299–300
manténgalo simple, 298–299
aprender haciendo, 303
descripción general
de, 298 tomarse el tiempo para hacerlo bien,
301–303 enfoque de equipo completo, 300–301
Principios, para evaluadores ágiles
retroalimentación continua, 22
mejora continua, 27–28 coraje, 25–26
entrega de valor al
cliente, 22–23 disfrute, 31 comunicación cara
a cara, 23–25
mantenerlo simple, 26–27 descripción general
de, 21–22

- enfoque en las personas, 30 respuesta al cambio, 28–29 autoorganización, 29–30
- Priorización de defectos, 381
- Priorización de historias, 338–340
- Mentalidad proactiva, 369–370 Valor comercial del producto, 31–33 entrega.
- Consulte Realización de pruebas de productos que critiquen (P3 y P4), 101–104, qué constituye un producto, 453–455 Propietario del producto.
- considerando todos los puntos de vista durante la planificación de iteraciones, 386–389 definición, 504 planificación de iteraciones y, 384 Roles Scrum, 141, 373 herramientas orientadas a, 134
- Producción
- registro de errores y, 421
 - soporte, 475
- Pirámide de pruebas
- de automatización de código de producción y, 277–278 definición, 504
 - entrega de valor, 70
 - programadores escribiendo, 48
 - control de código fuente y, 434
 - sincronización con pruebas, 322 desarrollo de prueba primero y 113 pruebas de soporte, 303–304
- Datos similares a producción, automatización de bases de datos y, 309–310
- Desarrollo profesional, 57
- Herramientas de perfilado, 234
- Actitud de los programadores con respecto a la automatización, 265–266 pruebas generales, 397
- colaboración con, 413–414 considerar todos los puntos de vista durante la planificación de la iteración, 387–389
- facilitar la comunicación y 429–430 revisar pruebas de alto nivel con, 400–401 proporción probador-desarrollador, 66–67 evaluadores en comparación con, 4, 5
- capacitación, 61 tarjetas de tareas de escritura y, 391 Oficina de Gestión de Proyectos (PMO), 440 Proyectos, ejemplo de PAS, 176–177
- Prototipos accesibles como lenguaje común, 134 maquetas y, 160 en papel, 22, 138–139, 380, 400, 414 papel versus tipo Mago de Oz, 275 UI (interfaz de usuario), 107 Pulse, 126 Herramienta de prueba unitaria PyUnit para Python, 126
- Q**
- Definición de control de calidad (garantía de calidad), 504 en títulos de trabajo, 31 equipo de control de calidad independiente, 60 intercambiable con "prueba", 59 enfoque de equipo completo, 39 trabajo en equipos tradicionales, 9 Cuadrante 1. Consulte Pruebas orientadas a la tecnología, equipo de soporte (Cuadrante 1)
- Cuadrante 2. Ver Pruebas orientadas al negocio, equipo de soporte (Cuadrante 2)
- Cuadrante 3. Ver Pruebas de cara al negocio, criticando el producto (Cuadrante 3)
- Cuadrante 4. Ver Pruebas de cara a la tecnología, criticando el producto (Cuadrante 4)
- Categorías de pruebas de automatización de cuadrantes, 274–276 de cara al negocio (Q2 y Q3), 97–98 pruebas basadas en el contexto y 106–108 críticas del producto (Q3 y Q4), 104 gestión de la deuda técnica, 106 descripción general de, 97–98 como guía de planificación, 490 propósito de las pruebas y, 97 Resumen del cuadrante 1, 99 Resumen del cuadrante 2, 99–100 Resumen del cuadrante 3, 101–102 Resumen del cuadrante 4, 102–104 responsabilidad compartida y, 105–106 finalización de la historia y, 104–105

- Cuadrantes, apoyo continuo al equipo (Q1 y Q2), 100–101 frente a la tecnología (Q1 y Q4), 97–98
- Papel del cliente de calidad en el establecimiento de estándares de calidad, 26 modelos, 90–93
- filosofía organizacional con respecto a, 38–40
- Seguro de calidad. Ver QA (garantía de calidad)
- Mentalidad policial de calidad, 57
- preguntas, para obtener requisitos, 135–136
- R
- Cartas de radar, XP, 47–48
- Rasmusson, Jonathan, 11 años
- Estrategia de automatización de herramientas de grabación/reproducción y, 294, 296–297
- Herramientas de prueba de GUI, 171–172
- Pruebas de recuperación, 459
- Pruebas de redundancia, 232
- Caña, David, 171, 377
- Definición de refactorización, 504
- IDE compatibles, 124–126
- Suite de regresión, 434
- Pruebas de regresión, 432–434
- pruebas de regresión automatizadas como red de seguridad, 261–262
- automatización como factor de éxito, 484
- comprobar el panorama general, 434 definición, 504
- pruebas exploratorias y, 212 mantener la compilación "verde", 433 mantener la compilación rápida, 433–434 registrar errores y, 420 suite de regresión y, 434
- versiones candidatas y, 458
- Criterios
- de aceptación de lanzamiento, 470–473
- final del juego, 327, 456–457
- gestión, 474 entrega del producto, 470 ¿qué pasa si no está listo, 463–464?
- Liberar candidatos
- compilaciones candidatas a lanzamiento desafiantes, 473
- definición, 505
- pruebas, 458
- Cobertura del código
- de métricas de lanzamiento, 360–364 métricas de defectos, 364–366
- descripción general de, 358 pruebas aprobadas, 358–360
- Notas de versión, 474
- Planificación de lanzamiento, 329–367
- descripción general de, 329 priorización y, 338–340 propósito de, 330–331 alcance, 340–344 dimensionamiento y, 332–337 alternativas de plan de prueba, 350–354 planificación de prueba, 345–350 visibilidad y, 354 –366
- Descripción general de las pruebas de confiabilidad, 230–231
- Ejemplo de sistema de monitoreo remoto de datos, 250–251
- Ejemplo de sistema de monitoreo remoto de datos pruebas de aceptación, 245 aplicación, 242–243
- aplicación de cuadrantes de prueba, 252–253 estructura de prueba funcional automatizada, 245–247
- documentar el código de prueba, 251
- pruebas integradas, 248 pruebas de extremo a extremo, 249–250
- pruebas exploratorias, 248–249
- descripción general de, 242 pruebas de confiabilidad, 250–251 informar resultados de pruebas, 251 equipo y proceso, 243–244 fuentes de datos de prueba , 249
- pruebas unitarias, 244–245 pruebas de aceptación de usuario, 250 servicios web, 247–248
- Miembro del equipo remoto. Ver equipos dispersos geográficamente

- Tareas repetitivas, automatización, 284.
- Documentación de informes y, 208–210
- Ejemplo de sistema de monitoreo remoto de datos, 251
- Repositorio, 124
- Requisitos que
- abordan las pruebas de cara al negocio, 130
 - documentación de, 402 que
 - reúnen todos los puntos de vista sobre los requisitos, 374–375 cómo obtener, 135–140 no
 - funcionales, 218–219 dilema, 132–134 herramientas para obtener ejemplos y requisitos, 155–156
- Recursos
- completar historias y, 381 contratar probadores ágiles, 67–69
 - descripción general de, 66 proporción probador-desarrollador, 66–67 pruebas y, 434–435
- Tiempo de respuesta
- API, 411
 - pruebas de carga y, 234–235
 - objetivos medibles y, 76 servicios web y, 207
- Retrospectivas de
- mejora continua y 28 ideas de mejora, 447–449 planificación de iteraciones y 383
 - descripción general de, 444–445
 - mejora de procesos y 90
 - ejercicios de “iniciar, detener y continuar”. 445–447
- Retorno de la inversión. Ver ROI (retorno de la inversión)
- Recompensas, rendimiento y, 70–71
- Herramientas de pruebas unitarias de cliente rico, 127 Rising, Linda, 121–122
- Análisis de riesgos, 198, 286, 290, 345–346
- evaluación de riesgos, 407–409
- mitigación de pruebas, 147–149
- Rogers, Pablo, 242, 310, 388, 398
- Automatización del ROI (retorno de la inversión) y, 264 definición, 505 medición lean y, 75 métricas y, 78–79 hablando el lenguaje del gerente, 55
- Rol, función, patrón de valor empresarial, 155
- Roles
- en conflicto o roles múltiples, 45 diferencias culturales entre, 48–49 equipo de cliente, 7 equipo de desarrollador, 7–8 interacción de, 8
- Unidad RPG, 118
- RSpec, 165, 318
- Prueba de Rubí::Unidad, 170
- Ruby con pruebas
- funcionales de Watir, 247
 - Pruebas de GUI, 285
 - identificación de defectos con, 212
 - palabras clave o palabras de acción para automatizar pruebas, 182 descripción general de, 172–174 automatización de pruebas con, 186
- RubyMock, 127
- Reglas, gestión de errores y, 425
- S
- Pruebas de seguridad, 232 Santos, Rafael, 448
- Condiciones de satisfacción. Consulte Condiciones de satisfacción
- Pruebas de escalabilidad, 233–234
- Pruebas de escenarios, 192–193
- diagramas de flujo y, 194–195
 - descripción general de, 192–195 pruebas de telenovela, 193
- Alcance, 340–344
- definición de pruebas orientadas al negocio, 134 plazos y cronogramas y, 340–341 enfoque en el valor, 341–342 descripción general de, 340 impacto en todo el sistema, 342

- Alcance, planes de
 prueba continuos y, 345
 participación de terceros y, 342–344
- Deslizamiento del alcance, 385, 412
- Scripts
 que automatizan comparaciones, 283
 como herramientas de
 automatización, 297 scripts de
 conversión, 461 herramientas de
 generación de datos, 305 pruebas exploratorias y, 211–212
- Melé
 rol de propietario del producto, 141, 373
- Ejemplo de sistema de monitoreo remoto de datos,
 244
- revisiones de sprint, 444
- ScrumMaster
 enfoque para la mejora de procesos, 448–449 dimensionar
 historias y, 336–337 escribir tarjetas
 de tareas y, 391
- SDD (desarrollo basado en pruebas de historias)
 que identifica variaciones, 410
 descripción general de, 262–
 263 desarrollo de prueba primero y, 263
 servicios web de prueba y, 170
- Pruebas de seguridad
 enfoque externo hacia adentro de los atacantes, 225
 descripción general de, 223–
 227 conocimientos especializados necesarios para, 220
- Selenio
 Herramientas de prueba de GUI,
 174–175 implementación de automatización, 316–
 318 herramientas de código
 abierto, 163 automatización de pruebas con, 186, 316
- Autoorganización
 principios, 29–30
 equipos autoorganizados, 69
- Pruebas basadas en sesiones, 200–201
- Automatización de configuración,
 284–285 pruebas exploratorias, 211–212
- Recursos compartidos
 acceso a, 43
 especialistas como, 301
 tareas de redacción y, 390
- Responsabilidad compartida, 105–106
- Caja de zapatos de agradecimiento, 450
- “Muéstrame”, colaboración con programadores,
 413–414
- Simplicidad,
 automatización y, 298–299
 codificación,
 406 errores de registro y, 428–429
 principio de “mantenerlo simple”, 26–27 Herramientas
 del simulador, pruebas
 integradas y, 248 descripción general
 de, 213 Tamaño,
 organizacional, 42–43 Historias de
 dimensionamiento, 332 –337
 ejemplo de, 334–337 cómo
 hacerlo, 332–333
 descripción general
 de, 332 el papel del evaluador
 en,
 333–334 Adaptabilidad de
 habilidades y, 39–
 40 versus actitud, 20 principio de mejora continua,
 27
 quién realiza las pruebas y, 220–221
- Pequeños trozos, desarrollo incremental,
 144–146
- JABÓN
 definición, 505
 pruebas de rendimiento y, 223, 234
- Pruebas de telenovela, 193
 definición
 de SoapUI, 505
 pruebas de rendimiento y 223, 234 pruebas
 de servicios web, 170–171
- Prueba SOAT, 234
- Herramientas basadas en software, 163
- Patrones de gestión de configuración de software:
 Trabajo en equipo efectivo, integraciones prácticas
 (Berczuk y Appleton), 124
- Software de finales (Galen), 471
- Beneficios del control del
 código fuente de, 255
 descripción general de, 123–
 124 herramientas para, 124, 320

- Cumplimiento de SOX, 469
- Hable con una sola voz, clientes, 373–374
- Especialización, 220–221
- La velocidad como objetivo, 112
- Picos, desarrollo y prueba, 381
- Hojas de cálculo,
- hojas de cálculo de prueba,
 - 353 herramientas para obtener ejemplos y requisitos.
- 159
- Revisões de Sprint, 444. Ver también Demos/
demonstraciones
- SQL*Loader, 460
- Pruebas de estabilidad,
- 28 Entorno de prueba, 458
- Reuniones de pie, 177, 429, 462
- Mantenibilidad
- de estándares y, 227 modelos de
 - calidad y, 90–93 “Iniciar, detener,
continuar”, retrospectivas,
- 445–447
- Análisis estático, herramientas de pruebas de
seguridad, 225 Steel thread, desarrollo incremental, 144,
338, 345
- Stories. Ver también Pruebas orientadas a empresas
- beneficios de trabajar antes de las iteraciones,
 - 370–372 brevedad de,
 - 129–130 pruebas orientadas
 - al negocio como, 130 determinar el
 - tamaño de la historia, 375–376 centrarse
 - en una historia al codificar,
- 411–412
- identificar variaciones, 410 saber
 - cuándo se termina una historia,
- 104–105
- registro de errores y, 420–421
 - maquetas y, 380
 - priorización, 338–340
 - recursos y, 381 alcance
 - y 340
- dimensionamiento. Consulte
- Dimensionamiento de historias
- comenzando de manera
- simple, 133, 406 pruebas de historia
- definidas, 505 impacto en todo el sistema, 342 planes de prueba y, 345489 Sistema, impacto de la historia en todo el sistema, 342
- estrategias de prueba y, 380–381
- comprobables, 393–
- 396 tratar errores como,
- 425 gráficos de
- evolución de guiones gráficos,
 - 429 definición, 505–506
- ejemplos, 356–357 en
- línea, 357, 384
- físicos, 356
- pegatinas y, 355
- tareas, 222, 355 , 436
- virtuales, 357, 384, 393
- trabajos en progreso, 390
- auditorías de
- tarjetas de historias
 - y 89 que se ocupan de errores y, 424–425
- planificación de iteraciones y 244
- narrativas de historias, 409
- desarrollo basado en pruebas de historias. Ver SDD
(desarrollo basado en pruebas de historias)
- Aplicación estranguladora (Fowler), 116–117
- Automatización de estrategias. Consulte Planificación
de pruebas de estrategia de automatización
- frente a estrategia de prueba,
- 86–87 estrategias de prueba, 380–
- 381 Estrategia, para escribir pruebas, construir
- pruebas de forma incremental, 178–
- 179 planificación de iteraciones y,
- 372 mantener las pruebas
- aprobadas, 179 descripción general
 - de, 177–178 diseño de pruebas.
- patrones, 179–183 comprobabilidad y
- 183–185 pruebas de estrés. Consulte
- Prueba de carga de Subversion (SVN), 124, 320
- Factores de éxito. Consulte
- Factores clave de éxito Éxitos, celebración de
 - la implementación del cambio y conclusión
- de la iteración 50–52 y, 449–451
- Sumrell, Megan, 365, 450 Ritmo sostenible de las
- pruebas, 40–41, 303 SVN (Subversion),
 - 124, 320 GUI de SWTBot herramienta
- de prueba, 127 Sinergia, entre prácticas,

- t
tail-f, 212
- Tartaglia, Coni, 439, 454, 470, 473 Tableros de tareas. Consulte Tableros gráficos Tarjetas de tareas que automatizan pruebas y, 394–395 planificación de iteraciones y, 389–392 entrega de productos y, 462–463 Tareas que completan tareas de prueba, 415–416 definición, 505–506 TDD (desarrollo basado en pruebas) pruebas de conducción automatizadas, 262–263 defectos y, 490 definición, 506 descripción general de, 5 Desarrollo de prueba primero en comparación con, 113–114 pruebas unitarias y, 111, 244–245 Ciudad del equipo, 126 Estructura del equipo, 59–65 equipos de proyecto ágiles, 64–65 equipo de control de calidad independiente, 60 integración de evaluadores en un proyecto ágil, 61–63 descripción general de, 59 estructura funcional tradicional frente a estructura ágil, 64 equipos automatización como esfuerzo de equipo, 484 construcción, 69–71 celebrando el éxito, 50–52 ubicado en el mismo lugar, 65–66 controlando la carga de trabajo y, 393 cliente, 7 desarrollador, 7–8 empoderamiento de, 44 facilitando la comunicación y, 429–432 dispersos geográficamente, 376–378, 431–432 otorgar el mismo peso a todos los miembros del equipo, 31 otorgar propiedad, 50 contratar probadores ágiles para, 67–69 interacción entre los equipos de clientes y desarrolladores, 8 planificación de iteraciones y, 384–385 logística, 59 resolución de problemas y, 123 Ejemplo de sistema de monitoreo remoto de datos, 243–244 responsabilidad compartida y, 105–106 tradicional, 9–10 uso de pruebas para respaldar los cuadrantes 1 y 2, 100–101 enfoque de todo el equipo. Consulte Enfoque de equipo completo trabajando en equipos ágiles, 10–12 Desmontaje, para pruebas, 307–308 Deuda técnica defectos como, 418 definición, 506 gestión, 106, 487–488 Descripción general de las pruebas orientadas a la tecnología, 5 Cuadrantes 1 y 4, 97–98 Pruebas de cara a la tecnología, criticando el producto. (Cuadrante 4), 217–239 líneas de base, 235–237 codificación y pruebas y, 412–413 pruebas de compatibilidad, 229–230 pruebas de instalación, 231–232 pruebas de interoperabilidad, 228–229 pruebas de mantenibilidad, 227–228 pruebas de gestión de memoria, 237–238 descripción general de, 217–219 pruebas de carga y rendimiento, 234 herramientas de prueba de carga y rendimiento, 234–235 pruebas de confiabilidad, 230–231, 250–251 pruebas de escalabilidad, 233–234 pruebas de seguridad, 223–227 entorno de prueba y, 237 cuándo usar, 222–223 quién realiza la prueba, 220–222 Pruebas orientadas a la tecnología, equipo de apoyo. (Cuadrante 1) crear herramientas, 126 diseñar teniendo en cuenta las pruebas, 115–118 facilidad para realizar tareas, 114–115 IDE para, 124–126 soporte de infraestructura, 111–112

- descripción general de, 109–
110 propósito de, 110–111
control del código fuente, 123–124
velocidad como beneficio de, 112–114
retroalimentación oportuna, 118–
119 kit de
herramientas para, 123
herramientas de prueba
unitaria, 126–127 pruebas unitarias, 244–245 qué hacer
si el equipo no
realiza estas pruebas, 121–123 dónde/cuándo detenerse, 119–121
- Pirámide de automatización de
pruebas: enfoque de múltiples capas para la automatización y,
290–291
descripción general de, 276–
279 metáfora de los tres cerditos, 278
- Prueba detrás de la interfaz de usuario, 282
- Casos de prueba
agregar complejidad, 407 como
documentación, 402 desarrollo
basado en ejemplos, 379 identificar variaciones,
410 comenzar de manera simple,
406
- Cobertura de prueba (y/o cobertura de código),
360–364
- Patrones de diseño de pruebas, 179–183
Patrón de construcción/operación/comprobación,
180 pruebas basadas en datos y palabras clave,
182–183
descripción general
de, 179 patrones de génesis de pruebas
(Veragen), 179 patrones de eventos, actividades y basados
en el tiempo, 181
definición de dobles
de pruebas, 506 arquitecturas en capas
y 116 desarrollo basado en pruebas. Ver TDD (desarrollo basado
en pruebas)
Entornos de prueba, 237, 487 Definición
de desarrollo de prueba primero,
506 TDD (desarrollo
basado en pruebas) en comparación con, 113–114 Gestión
de pruebas, 186 Kit
de herramientas de gestión de
pruebas (cuadrante 2), 186
- Alternativas al plan de prueba, 350–354
Planificación de pruebas, 345–350
listas de pruebas automatizadas, alternativas de planes de pruebas,
353–354
infraestructura y, 346–347 descripción
general de, 86, 345 razones
para escribir, 345–346 entorno de
prueba y, 347–348 alternativas de plan de
prueba, 350–354 planes de prueba,
muestra de plan de prueba 350
liviano, 351 estrategia de
prueba vs., 86–88 trazabilidad
y, 88 tipos de pruebas y,
346 por dónde empezar, 345
Comunicación de los
resultados de
las pruebas, 357–358 organización,
322–324 planificación de
lanzamiento y, 349–350 Habilidades de
prueba. Consulte Picos de
pruebas de
habilidades, 381 hojas de
cálculo de prueba,
353 iteraciones de estrategias de prueba, actividades previas a la iteración y,
380–381
plan de prueba vs., 86–88
Definición de
resguardos de
prueba, 506 integración con aplicaciones externas y,
459
pruebas unitarias y, 127
- Equipos de prueba, 506–507. Consulte también Herramientas de
prueba de Teams. Ver también Kits de herramientas
Funcional de la capa API, 168–170
pruebas exploratorias, 210–211
generación de datos de prueba con, 212
Pruebas de GUI, 170–176
elaboradas en casa, 175
de cosecha propia, 314
IDE, 124–126
pruebas de rendimiento, 234–235 pruebas
de seguridad, 225 pruebas
a nivel de unidad, 126–127, 165–168 pruebas
de servicios web, 170

- Tipos de
pruebas alfa/beta, 466–467
exploratorias. Ver Pruebas exploratorias (ET) funcionales.
- Consulte GUI de pruebas funcionales. Consulte
Integración de pruebas de
GUI, carga 229, 459. Consulte
Rendimiento de las pruebas
de carga. Consulte Rendimiento y confiabilidad de las
pruebas
de carga, 230–231, 250–251 seguridad,
220, 223–227 estrés. Ver
Unidad de prueba de carga.
Consulte Usabilidad de las
pruebas unitarias. Consulte Pruebas de
usabilidad y pruebas de aceptación del usuario. Ver
UAT (prueba de aceptación de usuario)
- Estrategia de redacción de pruebas. Ver Estrategia, para escribir.
pruebas
- Capacidad de prueba, 183–
185 pruebas automatizadas versus manuales del cuadrante 2,
185 automatización y, 149–150
diseño de código y diseño de pruebas y, 184–185
descripción general de,
183 de historias, 393–396
- Probadores
agregar valor, 12
probadores ágiles, 4, 19–20
mentalidad de prueba ágil, 20–21
automatización que permite centrarse en trabajos
más importantes, 260
colaboración con los clientes, 396–397 considerar todos
los puntos de vista durante la planificación de la iteración,
386–389 controlar la carga
de trabajo y, 393 definición, 507 facilitar la
comunicación, 429–
430 retroalimentación y 486 contratar probadores
ágiles, 67–69 cómo influir
en las pruebas, 121–122 integración
de probadores en un proyecto ágil,
61–63
iteraciones y, 327 facilitar
el trabajo, 114–115 dimensionar pisos,
333–334
- Proporción evaluador-desarrollador, 66–
67 escribir tarjetas de tareas y 391
- Declaración de derechos del evaluador,
49–50
Probar codificación y pruebas simultáneamente, 409–410
completar tareas de prueba, 415–416 identificar
variaciones, 410 administrar, 320–
322 organizar resultados de
pruebas , 322–324 pruebas de organización,
319–322 tiempo de planificación,
455–456 ciclos posteriores al desarrollo,
467–468 cuadrantes. Consulte Cuadrantes de
lanzamiento de candidatos, 458
evaluación de riesgos y, 407–
409 ritmo sostenible de, 40–41 tradicional
frente a ágil, 12–15 transparencia de
las pruebas, 321–322 Pruebas en
contexto
- pruebas basadas en contexto y, 106–108 definición,
502 herramienta de
prueba GUI TestNG, 127 pruebas
que nunca fallan, 286 matrices de
texto, 350–353 The Grinder, 234
temas. Consulte también
Definición de planificación de lanzamiento, 507
historias de
priorización y, 339 tarjetas de tareas de
escritura y, 392 Cortes finos, desarrollo
incremental y, 338 Pruebas de compatibilidad de terceros y, 230
planificación de
lanzamiento y, 342–344 software, 163
Tholfsen, Mike, 203 Thomas, Mike, 116,
194 Metáfora de
los tres cerditos, 278 Líneas
de tiempo, alcance y, 340–341
Herramientas de construcción del kit de
herramientas (Cuadrante 1), 126 IDE, 124–
126 descripción general de,
123 control de código
fuente, 123–124
herramientas de prueba
unitaria, 126– 127

- Kit de herramientas (Cuadrante 2)
- Herramientas de prueba funcional de la capa API, 168–170
 - herramientas de automatización, 164–165
 - pruebas de creación incrementales, 178–179 listas de verificación, 156
 - diagramas de flujo, 160–163
 - Herramientas de prueba GUI, 170–176
 - mantener las pruebas aprobadas, 179
 - mapas mentales, 156–158
 - maquetas, 160
 - herramientas basadas en software, 163
 - hojas de cálculo, 159
 - estrategias para escribir pruebas, 177–178 patrones de diseño de pruebas, 179–183 gestión de pruebas , 186 capacidad de prueba y, 183–185
 - estrategia de herramientas, 153–155
 - herramientas para obtener ejemplos y requisitos, 155–156
 - herramientas de prueba a nivel unitario, 165–168
 - Herramienta de prueba de servicios web, 170
- Herramientas de emulador del kit
- de herramientas (Cuadrante 3), 213–214
 - herramientas de monitoreo, 212–213
 - herramientas de simulador, 213
 - pruebas de aceptación del usuario, 250
- Líneas base del kit de herramientas (Cuadrante 4), 235–237
- Herramientas de prueba de carga y rendimiento, 234–235
- Herramientas**
- Herramientas de prueba funcional de la capa API, 168–170
 - automatización, 164–165 generación de datos, 304–305 seguimiento de defectos, 83–85 obtención de ejemplos y requisitos, 155–156, 159–163 herramientas de emulador, 213–214 pruebas exploratorias, 210–211 generando datos de prueba, 212
- Herramientas de prueba de GUI, 170–176 elaboradas en casa, 175
- de cosecha propia, 314
 - IDE, 124–126 pruebas de carga, 234–235
- monitoreo, 212–213 código abierto, 172, 314–315 pruebas de rendimiento, 234–235 para propietarios de productos y expertos comerciales, 134 pruebas de seguridad, 225 simuladores, 213 basados en software, 163 pruebas a nivel de unidad, 126–127, 165–168 proveedor/comercial, 315–316 herramienta de prueba de servicios web, 170
- Herramientas, automatización**
- ágil, 316 aplicación de una herramienta a la vez, 312–313 elaboración propia, 175 creación propia, 314 identificación de requisitos de herramientas, 311–312 código abierto, 314–315 selección, 294–298 proveedores, 315–316 Trazabilidad DTS y, 82 matrices, 86 planificación de pruebas y, 88 Seguimiento, estado y tareas de prueba, 354–357 Procesos tradicionales, transición. Ver
- Transición de procesos tradicionales a ágiles Equipos tradicionales, 9–10 Pruebas tradicionales versus ágiles, 12–15 Capacitación como entregable, 469 falta de, 45 Transición de procesos tradicionales a ágiles, 73–93 seguimiento de defectos. Consulte Seguimiento de defectos del proceso existente y, 88–92 mediciones ajustadas, 74–75 procesos livianos y, 73–74 métricas y, 74–79 descripción general de, 73 planificación de pruebas. Ver planificación de pruebas
- EN**
- Ciclos de pruebas posteriores al desarrollo UAT (pruebas de aceptación del usuario), entrega de productos 467–468 y, 464–466

- UAT (pruebas de aceptación del usuario), continúa en el Cuadrante 3, 102
planificación de lanzamiento para, 331, 346
Ejemplo de sistema de monitoreo remoto de datos, 250
en el plan de prueba, 351 probando nuevas funciones y 102
escribiendo en la reunión de inicio de la iteración, 372
UI (interfaz de usuario). Consulte también estrategia de automatización
de GUI (interfaz gráfica de usuario) y 293 modelado y 399
herramientas de prueba unitaria, 165–168. Ver también por herramientas unitarias individuales
herramientas de desarrollo impulsadas por el comportamiento, 166–168 lista de, 126–127 descripción
general de, 165
Automatización de pruebas unitarias, 282 BDD (desarrollo impulsado por el comportamiento), 165–168 definición, 507 métricas y 76 funciones de soporte de, 5 TDD (prueba -desarrollo impulsado) y, 111 pruebas orientadas a la tecnología, 120 herramientas para las pruebas del Cuadrante 1, 126–127 Pruebas de usabilidad, 202–204 verificación de aplicaciones de la competencia, 204 navegación y, 204 descripción general de, 202 necesidades de los usuarios y pruebas de personalidad, 202 –204 qué no debe automatizarse, 285–286 Casos de uso, 398 Pruebas de aceptación del usuario. Ver UAT (prueba de aceptación de usuario)
Documentación de usuario, 207–208
Interfaz de usuario (UI). Consulte también estrategia de automatización
de GUI (interfaz gráfica de usuario) y 293 modelado y 399
Historia de usuario. Ver Historia Tarjeta de historia de usuario.
Consulte la tarjeta de historias Historias de usuarios aplicadas al desarrollo de software ágil (Cohn), 155.
- V
Vaage, Carol, 330 Valor agregado, 31–33 entrega al cliente, 22–23 enfoque en, 341–342 evaluadores agregando, 12 valores, ágil, 3–4. Ver también Principios, para ágil probadores Variaciones, codificación y pruebas y, 410 Automatización de velocidad y, 255, 484 tasa de agotamiento y, 79 impacto en la base de datos, 228 defectos y, 487 definición, 507 maximización, 370 ritmo sostenible de pruebas y, 41 tomarse el tiempo para hacerlo bien, 301 deuda técnica y, 106, 313, 418, 506
Herramientas de automatización de proveedores, 315–316 herramienta de captura y reproducción, 267 IDE, 125 planificación y, 342–344 herramientas de control de código fuente, 124 trabajar con, 142, 349 Veragen, Pierre, 76, 163, 179, 295, 363, 372, 444 Control de versiones, 123–124, 186. Consulte también Puntos de vista de control del código fuente. Véase también Panorama general considerando todos los puntos de vista durante la planificación de la iteración, 385–389 reuniendo todos los puntos de vista relacionados con los requisitos, 374–375 Poder de tres y 411 usando múltiples puntos de vista para generar requisitos, 137–138
Visibilidad, 354–366 cobertura de código, 360–364 comunicar los resultados de las pruebas, 357–358 métricas de defectos, 364–366 número de pruebas aprobadas, 358–360

- descripción general de,
354 métricas de lanzamiento,
358 seguimiento de tareas y estado de prueba, 354–357
- Estudio visual, 125
- Voris, Juan, 117
- EN
- Enfoque en cascada, hacia el desarrollo.
desarrollo ágil en comparación con 12–13 fenómeno de
“mini-cascada”, 46–47 éxitos de, 112 planes de
prueba y 346 Watir
(Pruebas de aplicaciones
web en Ruby), 163, 172–174, 320. Véase también Descripción
de servicios web de Ruby con Watir Idioma (WSDL),
507
- Automatización de pruebas
de servicios web, 282
descripción general de, 207
- Ejemplo de sistema de monitoreo remoto de datos,
247–248
- herramientas para, 170–171
- Carga web, 234
- Whelan, Declan, 321
- Pizarras blancas
de desarrollo basado en ejemplos, 379
facilitando la comunicación, 430 modelando,
399 diagrama de
planificación, 371 revisando
pruebas de alto nivel con programadores,
400–401
alternativas del plan de prueba, 353–354
- Enfoque de equipo completo, 325
- ventajas de, 26
desarrollo ágil frente a tradicional, 15–16 estrategia de
automatización y, 300–301 límites
presupuestarios y, 55 disfrutar
del trabajo y, 31 factores clave de éxito, 482,
491 emparejamiento de evaluadores
con programadores, 279
- responsabilidad compartida y, 105–106
formación de equipos y, 69
estructura de equipo y, 59–62 para
probar la automatización, 270
gestión de pruebas y, 322 equipo
multifuncional tradicional en comparación con, 64 valor de
los
miembros del equipo y, 70
- Wiki
como herramienta de comunicación,
164 documentación gráfica de ejemplos,
398–399
maquetas, 160, 380
requisitos, 402 listas de
verificación de historias y, 156
casos de prueba,
372 trazabilidad y, 88
- Wilson-Welsh, Patrick, 278 Pruebas
del Mago de Oz, 138–139 Diagramas de
flujo de trabajo, 398 Trabajar
eficazmente con código heredado (Feathers), 117 , 288 Carga de
trabajo,
393 Peores
escenarios, 136, 334 Pruebas de
redacción, estrategia para. Consulte Estrategia, para
escribir pruebas
- WSDL (Lenguaje de descripción de servicios web),
507
- X
- XP (Programación extrema)
equipo ágil que adopta, 10–11 coraje
como valor fundamental en, 25 xUnit,
126–127
- Y
- Yakich, Joe, 316
- CON
Tolerancia cero a errores, 79, 418–419