LIBRO BLANCO | MAYO DE 2016

Empleo de pruebas basadas en modelos para impulsar un desarrollo guiado por el comportamiento

Huw Price
CA Technologies

ca technologies

Índice

Sección 1

Claridad y colaboración como principios de fomento

Debido a que las iniciativas del negocio y del departamento de TI están cada vez más alineadas, se hace imprescindible poder comunicar los requisitos empresariales de manera que se transfieran directamente a los proyectos técnicos. Las organizaciones modernas confían cada vez más en software que pueda aportar valor a sus clientes y es por ello por lo que los equipos de TI tienen que entregar software probado en su totalidad que satisfaga las necesidades empresariales cambiantes más rápido y por menos.

En cambio, con unos requisitos de software ambiguos es menos probable que los equipos de TI entiendan a la perfección el software tal y como lo conciben los usuarios finales y los analistas de negocios, como también es menos posible que el software aporte valor a la organización.

Esta frustración a la hora de desarrollar software es la que responde plenamente a una posible interpretación de los requisitos, solo para descubrir que no se trata de lo que se buscaba en un primer momento. Esta realidad motivó que Dan North defendiera el "desarrollo guiado por el comportamiento". Los constantes "malentendidos y confusiones" que experimentaba le llevaron a comparar el desarrollo con estar perdido en una serie de "callejones sin salida", donde, a menudo, pensaba: "¡Ojalá alguien me lo hubiera dicho!".[1]

Evidentemente, si se tienen en cuenta los desafíos a los que se enfrentan los desarrolladores y evaluadores durante los proyectos de TI típicos, esta no es la única frustración que experimentan. De media, solo el 4 % de los proyectos se inician con unos requisitos claros[2], lo que, a su vez, explica el 56 % de los defectos[3]. En general, dichos defectos se descubren tarde, durante la fase de producción, donde se tarda en solucionarlos 50 veces más que lo que se tardaría si se detectaran en la fase elaboración de requisitos[4]. En consecuencia, la repetición del trabajo debida a la ambigüedad de los requisitos es un motivo por el que el 60 % de los proyectos de software fallan o conllevan un sobrecoste. De hecho, unos requisitos mal definidos son la causa del 82 % del esfuerzo total empleado en solucionar errores[5] y del derroche de la enorme cantidad de 250 000 millones de dólares al año[6]. Por lo tanto, North tenía razones para pensar que se tenía que poder aplicar el desarrollo guiado por pruebas (TDD, por sus siglas en inglés) de manera que se obtuviera un buen resultado a la primera y se evitaran los obstáculos de los malentendidos.

Inspirado por el concepto de "lenguaje ubicuo", tomado del desarrollo basado en el dominio, el desarrollo guiado por el comportamiento (BDD) busca fomentar la colaboración entre el negocio y el departamento de TI. Dada la necesidad de que los equipos de TI respondan de forma rápida a los requisitos empresariales en constante cambio, se trata de un concepto aceptable: un lenguaje común que entiendan tanto los principales implicados como los secundarios (es decir, los que se centran en los objetivos empresariales y el comportamiento de la aplicación y los que aplican estos resultados y comportamientos) y que, debido a su semiformalidad, se traduzca con facilidad a la lógica del sistema que se va a desarrollar o probar.

Por consiguiente, el desarrollo guiado por el comportamiento consiste en formular correctamente los requisitos guiados por el comportamiento para que se puedan implementar con éxito en el desarrollo. En este documento únicamente se considerará este aspecto: se analizará cuál es la mejor manera de implementar estos principios esenciales de fomento de la colaboración y la claridad entre el negocio y el departamento de TI durante el ciclo de desarrollo.

Si se tiene en cuenta el proceso de recopilación de los requisitos que normalmente se proponen, se explica que la introducción del modelado formal pueda constituir, con las herramientas adecuadas, una técnica sinérgica para el desarrollo guiado por el comportamiento. Se cuestionará el hecho de suponer que la documentación completa, como oposición a la "documentación mínima", necesariamente se traduce en la incapacidad de responder rápidamente ante un cambio. En su lugar, se sostendrá que el modelado formal confiere integridad a los requisitos, que, junto con la precisión, es necesaria para que aumente la probabilidad de entregar software que se comporte como los principales implicados desean.

Sección 2

Falta de integridad

Según los principios de fomento del desarrollo guiado por el comportamiento que se han tratado, es evidente que deben ser comprensibles tanto para el negocio como para el departamento de TI y se deben implementar como requisitos técnicos que los evaluadores y desarrolladores puedan usar. Sobre el papel, recopilar requisitos mediante la interacción de los principales implicados que realmente obtendrán valor directo del software, lo que aporta ejemplos concretos o reales de cómo quieren que se comporte una aplicación, parece ser una forma segura de garantizar que, verdaderamente, los requisitos reflejan el sistema concebido por el negocio.

Sin embargo, a pesar de que este desarrollo "hacia dentro" lleva a erradicar la ambigüedad, la naturaleza de los requisitos y su determinación plantean cuestiones sobre en qué medida se pueden implementar para probar y desarrollar sistemas.

Tal y como afirma Llyr Wyn Jones en su artículo "A Critique of Testing", se puede considerar la implementación de los requisitos como la transformación de la información. Se toma la información de un modo y se manifiesta de otro; por ejemplo, un desarrollador, al codificar, transforma los requisitos en un componente de software. Todo el ciclo de desarrollo de software (SDLC, por sus siglas en inglés) se puede plantear como un flujo de información: el usuario trabaja con un analista de negocios para diseñar los requisitos y los casos prácticos, el desarrollador escribe el código y, después, los evaluadores escriben manualmente escenarios de pruebas y scripts de pruebas. En cada una de estas fases, la información se transfiere de una forma y se "traduce" de otra distinta.

Concibiendo los requisitos como información activa, Llyr Wyn Jones define la ambigüedad en referencia al concepto de "incertidumbre". Según describe, esto conduce a resultados diferentes, ya que la incertidumbre en relación con el significado de las instrucciones lleva a más de una manera de obedecer el requisito. Asimismo, esta falta de integridad surge cuando la información necesaria de un sistema se presta a conjeturas[7]. En consecuencia, ambas situaciones conducen hacia la incertidumbre y la probabilidad de que se produzcan malentendidos entre los implicados principales y los secundarios.

En resumen, la falta de integridad, así como la ambigüedad, hace menos probable que el software se comporte como los implicados pretenden que lo haga. La falta de integridad aumenta la probabilidad de que se produzcan, durante el desarrollo, el tipo de malentendidos que Dan North trató de resolver, lo que difiere totalmente de los principios que llevaron a la formulación del desarrollo guiado por el comportamiento.

Sección 3

Falta de integridad en los requisitos guiados por el comportamiento

En el contexto del desarrollo guiado por el comportamiento, el flujo de información descrito por Jones se produce en un "bucle de retroalimentación": se formulan los requisitos, se desarrolla y se prueba el software, y se transfieren los informes a los principales implicados para su evaluación. En este caso, el riesgo de incertidumbre es que el "bucle" se repita indefinidamente, ya que el software, normalmente, no se entrega como se pretende y los requisitos están en constante cambio.

Al elaborar los requisitos y los escenarios de pruebas en función de los objetivos empresariales y el comportamiento deseado, las pruebas enfatizan lo que debería suceder. En otras palabras: principalmente, se centran en el resultado ideal. Este hecho se puede observar en los lenguajes naturales propuestos para diseñar situaciones.

Vista desde la perspectiva de los operadores booleanos, la lógica del sistema se puede reducir totalmente a tres funciones: NOT, OR y AND, además de XOR (una combinación de las tres). A su vez, se puede plantear en términos de declaraciones IF...THEN.

El lenguaje Gherkin, por ejemplo, hace más hincapié en las declaraciones "IF", "AND" y "THEN". Describe las condiciones iniciales y las cláusulas establecidas para una situación, así como los "desencadenantes" que existen realmente (es decir, "IF") y el resultado que se espera de la situación ("THEN"). Las características AND de cada uno de esos pasos pueden tener múltiples desencadenantes o hechos. En apariencia, OR está ausente cuando se recopilan escenarios de pruebas en este formato. Se puede observar de forma más clara en el lenguaje natural alternativo de Rspec, que puede conformar situaciones como una serie de declaraciones "When...Then"[8]. En otras palabras: este tipo de lenguajes naturales no consideran qué sucede cuando no existe un desencadenante.

Sin embargo, las decisiones que este "OR" refleja son fundamentales para la lógica de un sistema y cada una representa una posible ruta diferente que el sistema deberá destacar como parte de las pruebas. Esta cuestión es especialmente importante en el caso de las rutas negativas, que deben constituir en torno al 80 % de la tarea total de la fase de pruebas. Este tipo de resultados inesperados tienen lugar precisamente cuando no existen los desencadenantes y, por ello, no se tiene en cuenta la condición "IF", cuando son estos resultados inesperados y valores atípicos los que tienen más probabilidades de provocar un error en el sistema.

Hay quien argumenta que el desarrollo guiado por el comportamiento es una forma adecuada de identificar las rutas que no se han considerado al diseñar el programa, ya que se tienen en cuenta los resultados inesperados si se producen y cuando se producen. Después de todo, el desarrollo guiado por el comportamiento es un proceso repetitivo y, por lo tanto, reducir el "bucle de retroalimentación" permite que estas rutas negativas se atiendan antes y que las pruebas se adelanten dentro del ciclo. Algunos van más allá y sugieren que esas organizaciones deberían "abogar por la incertidumbre"[9].

No obstante, como se ha mencionado, dejar la detección de defectos para las pruebas es costoso y tedioso. Además, en la realidad, las repeticiones cortas se convierten, con frecuencia, en pequeñas cascadas que hacen que las pruebas se pospongan o no se lleguen a ejecutar. Asimismo, existe un problema más grave, el de la observación de los defectos. Se pretende que el resultado esperado ("Then") de una situación sea una condición verificable y se supone que las pruebas sirven para determinar que se produce el resultado esperado y que lo causó el desencadenante adecuado, además de que surgió de la clausula establecida correcta. No obstante, cuando se realizan pruebas *ad hoc*, no hay forma de saber con seguridad si los resultados se obtuvieron por la razón que se supone que los provoca y no porque dos o más defectos se anulen.

Sección 4

Pruebas basadas en modelos y desarrollo guiado por el comportamiento

Con el fin de dirigirse hacia la integridad y aumentar las probabilidades de que se entregue software de la manera deseada, se debe introducir el concepto de "OR" junto con "AND" en el desarrollo guiado por el comportamiento. Se deben conectar estos escenarios de desarrollo guiado por el comportamiento presentados de forma discreta y los lenguajes como el Gherkin para reflejar la lógica de un sistema. De este modo, las pruebas en el desarrollo guiado por el comportamiento también considerarán qué sucede si el desencadenante no funciona como debería, lo que lleva al sistema a una ruta negativa.

Normalmente, al escribir y ejecutar pruebas, los evaluadores generan modelos de formularios, si bien de manera implícita. Por ejemplo, durante una prueba negativa es posible que deban considerar qué sucede si el desencadenante no se produce, pero las cláusulas establecidas sí. De nuevo, en el desarrollo guiado por el comportamiento, si dos escenarios de pruebas comparten un paso, están conectados de forma implícita por un "OR". Por ejemplo, es posible que compartan un desencadenante, pero no otro, lo que constituye tomar una decisión en la lógica del sistema.

Sin embargo, cuando se ha llevado a cabo el modelado *ad hoc*, las pruebas solo tendrán posibilidad de abarcar las situaciones que suceden en la mente de los desarrolladores y evaluadores. Este es el caso del desarrollo guiado por el comportamiento, cuando las historias de usuarios se presentan como unidades lineales y discretas que no reflejan cómo se relacionan entre sí en la lógica del sistema. Por consiguiente, la cobertura funcional de las pruebas se mantendrá baja, normalmente entre el 10 y el 20 %, ya que incluso un sistema sencillo tiene probabilidades de disponer de miles de combinaciones de entradas y salidas posibles, más de lo que cualquier persona puede retener en la mente y vincular de forma correcta.

Si el desarrollo guiado por el comportamiento requiere que, de todas formas, los evaluadores modelen, no hay razón por la que esto no se pueda llevar a cabo de forma sistemática, proporcionando una documentación de requisitos sin ambigüedades y, fundamentalmente, completa. Este método sistemático es necesario cuando las pruebas no abarcan todas las rutas posibles de un sistema, incluidas las rutas negativas y los resultados inesperados.

Sección 5

Modelado del diagrama de flujo como parte del desarrollo guiado por el comportamiento

A continuación, se explica una propuesta sobre cómo se puede incorporar el modelado en el desarrollo, de manera sinérgica, a los principios de desarrollo guiado por el comportamiento tratados anteriormente.

1. Especificación mediante ejemplos. Cabe señalar que los requisitos pueden seguir basándose en el comportamiento y se pueden elaborar a partir de la interacción del implicado sobre cómo querría que apareciese el sistema. Los requisitos formulados por el negocio se establecen mediante ingeniería inversa, como un diagrama de flujo, o los propios implicados principales crean un flujo. Este último es posible porque el diagrama de flujo sirve como lenguaje ubicuo. Philip Howard, de Bloor Research, describió el modelo de diagrama de flujo como algo que puede estar compuesto por toda la lógica funcional sobre un sistema que los evaluadores y desarrolladores necesitan, sin dejar al negocio "boquiabierto"[10].

   Si los desarrolladores y los evaluadores están usando requisitos de un desarrollo guiado por el comportamiento existente, el proceso consiste en conectar los pasos superpuestos de las situaciones, que, en ese caso, constituyen decisiones (OR) en la lógica del sistema. Este proceso obliga a los modeladores a reflexionar sobre la lógica del sistema, aplicando la integridad, mientras que CA Agile Requirements Designer (anteriormente, Agile Designer de Grid Tools) identificará cualquier ruta que falle y sugerirá rutas para las posibles situaciones que no se hayan considerado. En consecuencia, las rutas que no se han considerado o las negativas se identifican antes de que se complete el bucle de retroalimentación, lo que reduce el número de repeticiones necesarias.

2. Escenarios de modelado. En realidad, las historias de usuario tienen un nivel demasiado elevado para el desarrollo y la realización de pruebas. En su lugar, las situaciones que hay que desarrollar y probar se modelan en el diagrama de flujo, en el que los pasos de los escenarios forman el proceso y los bloques de decisiones de un diagrama de flujo, así como las situaciones se convierten en rutas a través de la lógica del sistema. Asimismo, si se modelan todas las situaciones, se abarca cada historia de usuario, ya que las situaciones son derivaciones de las historias de usuario.

   Todos los aspectos de una narrativa se pueden transferir fácilmente a un diagrama de flujo. Por ejemplo, es posible que un bloque del proceso especifique qué papel desempeña el implicado, describiendo cómo un "cliente intenta retirar dinero de un cajero automático". En consecuencia, es posible que las rutas a través de un diagrama de flujo reflejen una prueba de unidad en el desarrollo guiado por el comportamiento, para el que se especifica una función, una característica deseada y una ventaja o resultado esperado.

   Debido a los métodos combinatorios, no existirá una ruta única ni distinta desde el principio hasta el final de cada situación. En su lugar, los componentes de la lógica funcional de múltiples situaciones o características se combinarán en

determinadas partes del diagrama de flujo, lo que refleja una importante diferencia entre las pruebas elaboradas a partir de los requisitos del desarrollo guiado por el comportamiento recopiladas mediante lenguajes naturales como Gherkin y el modelado de la principal lógica funcional del sistema: al modelar un solo escenario de pruebas puede abarcar múltiples situaciones, ya que se ha consolidado en el sistema como un todo, más que presentarse como unidades discretas. Esta consolidación de las historias de usuario como sistema completo es el paso clave para reducir la incertidumbre.

Por ejemplo, es posible que varios escenarios se consoliden en un determinado "Given" una vez se haya modelado el sistema como un solo diagrama de flujo. Este "Given" podría ser un subflujo o una parte de las múltiples rutas, que, en consecuencia, describe un bloque de decisiones.

De modo similar, varios "When" pueden funcionar en cualquier ruta, bajo la forma de bloques de decisiones o bloques de procesos. Como resultado, se consolidan múltiples situaciones porque si dos o más situaciones comparten los desencadenantes, pueden pasar por el mismo bloque. Cada decisión puede llevar a una nueva ruta o conjunto de rutas, de forma continua, hasta que se modele un sistema en su totalidad, incluidas todas las rutas negativas. Por ejemplo, si dos situaciones comparten todo excepto un desencadenante, podrían seguir la misma ruta hasta el último bloque de decisiones, donde divergen.

3. El bucle de retroalimentación: verificación del modelo. Si son los usuarios y los analistas de negocio los que han modelado un sistema, no se requiere verificación y se puede pasar a la siguiente fase del desarrollo guiado por el comportamiento. En un caso así, en efecto, la demora en la respuesta será muy corta, como se describirá más adelante.

   Si son los evaluadores y los desarrolladores los que han modelado el sistema, verificar el modelo es igualmente rápido y sencillo. En CA Agile Requirements Designer, es posible realizarlo con casos prácticos. Cada ruta del diagrama de flujo representará un caso práctico diferente, que equivale a un escenario de prueba o situación una vez se ha modelado un sistema. La verificación consiste en pasar por un conjunto de casos prácticos (presentados tanto con el vocabulario común del texto sin formato como en un diagrama de flujo) que confirmen que se ha diseñado el sistema tal y como se concibió.

4. El bucle de retroalimentación: validación del modelo. Los escenarios de pruebas se pueden elaborar automáticamente del diagrama de flujo, ya que se compone de la lógica funcional de un sistema. Como se ha mencionado, cada ruta posible de un sistema constituye un escenario de pruebas. CA Agile Requirements Designer es capaz de identificar de forma automática las rutas posibles, así como calcular las estadísticas precisas para la cobertura funcional que proporcionan.

   Es más, se pueden eliminar las pruebas redundantes y no válidas, además de los escenarios de pruebas posibles deduplicados, lo que proporciona el menor conjunto de escenarios de pruebas necesario para proporcionar la cobertura funcional máxima. Como se ha indicado, se puede consolidar la lógica de múltiples escenarios y, por lo tanto, es posible probar un conjunto de situaciones mediante un conjunto de casos de prueba más pequeño; por ejemplo, se pueden probar 15 situaciones posibles mediante tres casos de prueba. La optimización de las pruebas ofrece varios algoritmos para identificar qué rutas proporcionan la cobertura funcional máxima. Por ejemplo, en un caso, CA Agile Requirements Designer redujo el número de pruebas posibles de 326 a solo 17, con una cobertura completa.

   Una vez almacenadas estas rutas, se pueden exportar y ejecutar. Es posible realizarlo a mano, mediante la incorporación de escenarios de pruebas (vinculados a datos de pruebas y resultados esperados) a una herramienta de gestión de pruebas como HP ALM/QC. De forma alternativa, se pueden exportar las rutas como scripts de pruebas automatizadas, que se ejecutan en numerosos motores de automatización. En ambos casos, la herramienta de gestión de pruebas usada produce la "documentación activa" y los informes.

Este proceso automatizado puede parecer similar al del Cucumber, a excepción de que es probable que la demora en la respuesta sea mucho más reducida, a la vez que se requieren menos bucles de retroalimentación. En primer lugar, la generación automática de los casos de prueba elimina la necesidad de definirlos manualmente (en el caso de Cucumber, suprime la necesidad de convertir de manera manual el lenguaje Gherkin en definiciones de paso en Ruby). Por ejemplo, en un caso, CA Agile Requirements Designer tardó 90 minutos en crear 108 casos de prueba, lo que proporcionó una cobertura funcional completa. En la práctica, el tiempo ahorrado será mayor, ya que el diagrama de flujo se puede modificar y reutilizar fácilmente, como se analizará más adelante.

Además, debido a que se han proporcionado los requisitos completos, verificados y sin ambigüedades a los desarrolladores, es más probable que entreguen el software que se concibió en un principio. Como se ha mencionado, la ambigüedad provoca el 56 % de los defectos y, a este respecto, CA Agile Requirements Designer ha reducido la creación de defectos hasta en un 95 %. Del mismo modo, es probable que se elija en primer lugar cualquier defecto más allá del desarrollo para realizar pruebas, gracias a la cobertura funcional completa que proporcionan los escenarios de pruebas. Además del tiempo que se ahorra en el trabajo duplicado, es probable que se reduzca el tiempo de ejecución. Normalmente, duplicar las pruebas ha reducido su realización en un 30 %, mientras que la optimización de pruebas puede reducir drásticamente el número total de las pruebas necesarias para abarcar cualquier situación.

Sección 6

Respuesta rápida y sencilla ante el cambio

Es posible que exista una objeción a este modelado del diagrama de flujo: el mayor tiempo empleado en los entornos de desarrollo guiados por el comportamiento. Este tiempo se podría haber empleado mejor en ejecutar y volver a ejecutar las repeticiones de pruebas.

En primer lugar, se debe indicar que estas críticas realmente solo se mantienen si los propios evaluadores y desarrolladores tienen que estructurar los diagramas de flujo; no existiría si los analistas de negocio y los usuarios finales hubiesen favorecido los diagramas de flujo en lugar de usar lenguajes como Gherkin. No obstante, incluso si los equipos técnicos tienen que recurrir a la ingeniería inversa para los requisitos del desarrollo guiado por el comportamiento, la cantidad de tiempo empleado en realizar los diagramas de flujo es, en general, menor. Una vez generado el diagrama de flujo, se puede reutilizar con facilidad, incluso cuando los requisitos cambian; lo que va en contra de la suposición de que la documentación completa necesariamente implica la incapacidad de responder de forma rápida ante el cambio.

En CA Agile Requirements Designer, realizar un cambio en un diagrama de flujo es un proceso rápido y sencillo, ya que basta con que los usuarios agreguen un nuevo componente de la lógica funcional al diagrama de flujo. En consecuencia, los escenarios de pruebas se identifican y solucionan automáticamente, mientras que las pruebas redundantes y duplicadas se eliminan. CA Agile Requirements Designer generará, entonces, cualquier caso de prueba nuevo que se requiera para proporcionar la cobertura funcional máxima.

Esto aumenta el valor del trabajo inicial realizado (generar un diagrama de flujo) y elimina el tiempo empleado en comprobar manualmente todos los casos de prueba cuando se realiza un cambio. En una organización, se empleó dos minutos en modificar un diagrama de flujo existente tras realizar una solicitud de cambio. CA Agile Requirements Designer identificó y solucionó automáticamente los tres casos de prueba que se vieron afectados, lo que dejó 64 sin modificar. Como se muestra en este ejemplo, el modelado del diagrama de flujo no solo implica que se probarán de nuevo los aspectos de un sistema que se ha visto afectado por un cambio, lo que ayuda a mejorar la eficiencia del bucle de retroalimentación.

En resumen, es más probable que el tiempo que se ahorra en la duplicación del trabajo y la depuración (provocadas por la falta de integridad y la ambigüedad), la generación de escenarios de pruebas, su ejecución y la implementación de solicitudes de cambios supere al empleado en generar un diagrama de flujo. En consecuencia, la documentación completa contradice los principios del desarrollo guiado por el comportamiento, pero puede mejorar su implementación.

Sección 7

Resumen

El modelado del diagrama de flujo proporciona una técnica para introducir la documentación completa en el desarrollo guiado por el comportamiento sin comprometer sus principios esenciales. El diagrama de flujo se puede elaborar directamente a partir de las interacciones de los implicados y es lógico tanto para el negocio como para el departamento de TI, pues proporciona un lenguaje ubicuo que se puede complementar con todos los lenguajes funcionales sobre un sistema necesarios para los equipos de pruebas y desarrollo. Asimismo, un diagrama de este tipo puede reducir la demora en la respuesta, ya que se reducen los ciclos y las tareas manuales de las pruebas.

El modelado formal aumenta la probabilidad de que se entregue software más preciso respecto a los requisitos a la primera, ya que conlleva un conjunto de requisitos guiados por el comportamiento completos y carentes de ambigüedad. Así, se reduce el número total de bucles de retroalimentación necesarios para que el software cumpla con los requisitos empresariales en un momento dado, por lo que las organizaciones se pueden centrar en la innovación, más que en las repeticiones y la frustrante repetición del trabajo. Esta capacidad de desarrollar soluciones de forma continua con el propósito de maximizar el valor que reporta a los clientes viene dada por la posibilidad de responder de forma rápida y fácil al cambio.

Sección 8

Acerca del autor

Con una carrera profesional que alcanza casi los 30 años, Huw Price ha sido el arquitecto técnico principal de numerosas empresas de software de EE. UU. y Europa, y ha proporcionado asistencia en el diseño arquitectónico de alto nivel a bancos multinacionales y a los principales proveedores de servicios y de servicios sanitarios. Recibió el premio “IT Director of the Year 2010” (Director de TI de 2010) de mano de QA Guild. Huw se ha especializado a lo largo de los años en las herramientas de automatización de pruebas y ha lanzado numerosos productos innovadores que han reestructurado el modelo de pruebas que se usa en el sector del software. En la actualidad, participa como ponente en eventos reconocidos internacionalmente y su trabajo se ha publicado en varias revistas como Professional Tester, CIO Magazine y otras publicaciones técnicas.

CA Technologies adquirió la última empresa de Huw, Grid-Tools, en junio de 2015. Durante casi una década, ya se había dedicado a redefinir la manera en que las grandes organizaciones enfocan su estrategia para realizar pruebas. Con el liderazgo y el enfoque visionario de Huw, la compañía ha introducido una sólida metodología de pruebas centrada en los datos, lo que ha derivado en nuevos conceptos que Huw denomina “objetos de datos”, “transmisión de datos” y “almacén central de datos de pruebas”.

Comuníquese con CA Technologies en ca.com/es

CA Technologies (NASDAQ: CA) crea software que impulsa la transformación de las empresas y les permite aprovechar las oportunidades que brinda la economía de las aplicaciones. El software se encuentra en el corazón de cada empresa, sea cual sea su sector. Desde la planificación hasta la gestión y la seguridad, pasando por el desarrollo, CA trabaja con empresas de todo el mundo para cambiar la forma en que vivimos, realizamos transacciones y nos comunicamos, ya sea a través de la nube pública, la nube privada, plataformas móviles, entornos de mainframe o entornos distribuidos. Para obtener más información, visite **ca.com/es**.

1 Dan North, Introducing BDD (2006), consultado el 6 de marzo de 2015 en http://dannorth.net/introducing-bdd/.

2 Chaos Manifesto 2013 (Standish Group: 2013), consultado el 7 de marzo de 2015 en http://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf.

3 Requirements Based Testing Process Overview (Bender RBT: 2009), consultado el 5 de marzo de 2015 en http://benderrbt.com/Bender-Requirements%20Based%20Testing%20Process%20Overview.pdf.

4 Why testing should start early in software development life cycle? (Software Testing Class: 2012), consultado el 6 de marzo de 2015 en http://www.softwaretestingclass.com/why-testing-should-start-early-in-software-development-life-cycle/.

5 Bender RBT, Requirements Based Testing Process Overview.

6 Kathleen Barret, Business Analysis: The Evolution of a Profession (IIBA: 2013), consultado el 5 de marzo de 2015 en http://www.iiba.org/Careers/Careers/Business-Analysis-The-Evolution-of-a-Profession.aspx.

7 Consulte Erik Kamsties, "Understanding Ambiguity in Requirements", citado en Engineering and Managing Software Requirements (Springer: 2005), P. 250.

8 Para ver un ejemplo de ello, consulte https://es.wikipedia.org/wiki/Desarrollo_guiado_por_comportamiento.

9 Dan North, Embracing Uncertainty (Goto Con: 2013), consultado el 6 de marzo de 2015 en http://gotocon.com/dl/goto-chicago-2013/slides/DanNorth_EmbracingUncertainty.pdf.

10 Test Case Generation, consultado el 6 de marzo de 2015 en https://www.ca.com/es/collateral/industry-analyst-report/bloor-research-market-report-test-case-generation.register.html.

CS200-196736_0516