Asegurar la calidad de lo que vende la empresa al cliente resulta fundamental para poder sobrevivir en el mercado. Hoy en día hay muchas aplicaciones *software* que intentan dar respuesta a nuevas necesidades, y hay que ser ágiles y garantizar la calidad de las entregas.

Cómo asegurar la calidad del *software*"

Raúl Sánchez Vítores,
Gestor de proyectos QA en Fractalia

QUÉ ES 'QUALITY ASSURANCE'

Q*uality Assurance*' (QA) o 'Aseguramiento de la Calidad' es básicamente un conjunto de actividades de ingeniería que se encargan de garantizar la calidad de un producto o servicio.

En este caso, el producto a tratar va a ser el *software* de la empresa. Hay que ser conscientes de que la situación actual es que estamos en un mundo muy cambiante y competitivo, y hay que garantizar calidad para no quedarse fuera del mercado, y ser al mismo tiempo lo más ágiles posibles en nuestra entrada a ese nicho de negocio.

Además, el coste de la falta de calidad, con los consiguientes defectos en producción, es mucho más elevado que si

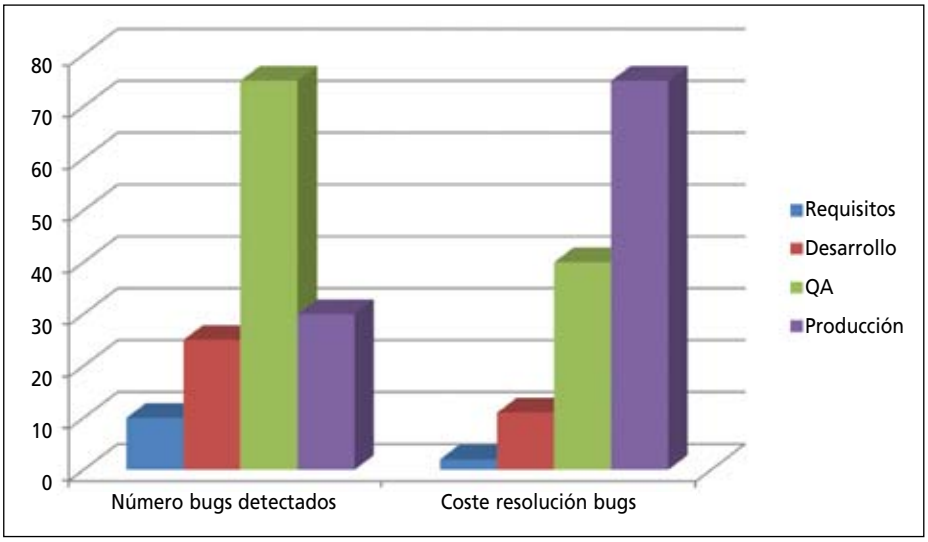

Figura 1: La necesidad de las pruebas.

se descubren antes de realizar la entrega. Es más, en muchos casos estos costes en cliente pueden conllevar quedarse fuera del proyecto, mientras que si los errores (bugs) se descubren en la fase de especificación, simplemente es cambiar una línea en un documento. Por tanto, hay que intentar detectarlos lo antes posible. (Ver *Figura 1*).

NIVELES DE PRUEBAS A REALIZAR

Una vez que está clara la importancia de la calidad en el desarrollo *software* realizado, hay que ver ahora cómo asegurar que esto se cumple.

Para ello, hay diversos niveles de pruebas a ejecutar:

– El primer nivel serían las pruebas unitarias, a desarrollar por el departamento de Desarrollo, ya que son conocedores del código fuente implementado. Son pruebas destinadas a garantizar que las distintas partes del código funcionan correctamente por separado. Estas pruebas no se deben confundir con las pruebas que debe realizar el desarrollador mientras está programando su funcionalidad. Además, estas pruebas unitarias sirven para poderlas lanzar automáticamente cada vez que se saque una 'build' (compilación) del producto, y ver así, si no se ha roto nada durante ese tiempo de desarrollo. Una vez terminado todo el proceso de desarrollo se puede pasar al siguiente nivel de pruebas.

– Luego estaría el nivel de pruebas de integración, que es una tarea conjunta entre el departamento de Desarrollo y de QA. Sirve para comprobar que todos los módulos funcionan correctamente en conjunto, de forma básica, y son el preludio del siguiente nivel.

– De forma básica, se podría terminar con el nivel de pruebas de sistema, que son responsabilidad del departamento de QA, y que se pueden dividir en dos grandes grupos: pruebas funcionales y pruebas no funcionales.

TIPOS DE PRUEBAS FUNCIONALES Y NO FUNCIONALES

En el nivel de pruebas de sistema, se distinguen por tanto las pruebas funcionales y pruebas no funcionales.

– Pruebas funcionales: hay muchos tipos y mucha teoría al respecto, como son pruebas exploratorias, de regresión, de compatibilidad, de aceptación, etc. No obstante, conviene 'centrar el tiro' y pensar que siempre se deben pasar un mínimo de casos de test de regresión (para asegurar que las nuevas funcionalidades dejen inoperativas otras funciones antiguas del sistema, dejen de ser operativas), y pruebas funcionales básicas que aseguren que se cumplen con todos los requisitos del cliente.

Estas pruebas suelen realizarse de forma manual, si bien conviene automatizar los procesos de negocio que son claves en la aplicación, y que constituyen el 'core' (núcleo) de la misma. Así se elimina el posible error humano al pasar la prueba, y también se ahorra tiempo en posteriores ejecuciones, ya que son cosas que siempre se deberían probar en cada entrega. No obstante, automatizar o no, depende de cada caso concreto, si va a merecer la pena el tiempo invertido en la automatización, y si es posible incluso dicha automatización.

Esta automatización de las pruebas funcionales es lo que se conoce como el desarrollo de proyectos de pruebas de UI (*User Interface*) codificadas o 'robots'. Se pueden usar herramientas como Visual Studio, TestComplete, SeeTest, etc. que suelen utilizar un grabador de las acciones realizadas, para poder luego combinarlas, y realizar pruebas modulares. Siempre va a interesar que cada prueba sea configurable, que pueda utilizar un datasource como entrada de datos (*Data Driven Coded UI Test*), y sean reutilizables, para poder así formar como un 'puzzle' que, combinando las piezas, permitan probar la aplicación de extremo a extremo. Por tanto, se recomienda grabar menos de 10 acciones cada vez, para hacerlo más modular y manejable. Además, hay que poner nombres significativos, para luego poder identificar las pruebas fácilmente, y evitar así problemas a la hora de reutilizarlas, ejecutarlas, etc.

Para este cometido de realizar pruebas funcionales se suele realizar un test plan, en el que se indican los casos de test a pasar, a partir de los requisitos del producto. Dichos casos de test, deben tener detallados los pasos para su ejecución y los resultados esperados. También conviene indicar su prioridad, y si es una prueba a realizar de forma manual o si está automatizada. Se pueden usar herramientas como Jira, TestLink, una hoja Excel, etc. para realizar toda esta tarea.

– Pruebas no funcionales: como en el punto anterior hay mucha variedad aquí también, y están entre otras, pruebas de seguridad, usabilidad, escalabilidad, portabilidad, etc. No obstante, conviene cubrir al menos las llamadas pruebas de rendimiento en su conjunto más general, identificando los escenarios claves en la aplicación que son más susceptibles de tener un rendimiento adecuado.

Conviene destacar aquí que pruebas funcionales puede haber cientos, para probar que toda la interfaz de usuario funciona correctamente y cumple con las especificaciones, sean aplicaciones web, de escritorio, para dispositivos móviles, etc. Sin embargo, pruebas no funcionales habrá del orden de decenas, identificando los escenarios más críticos.

Todas las incidencias ('bugs') convienen que estén reportadas por escrito en alguna herramienta de gestión, tipo Bugzilla, Jira, GForge o Team Foundation Server, que facilitará luego búsquedas, seguimiento y reparto de trabajo. Se debe poner el escenario donde se reproduce el bug, los pasos a reproducir, resultados actuales y resultados esperados. También es importante indicar severidad y criticidad del asunto en cuestión, para establecer prioridades y unos criterios mínimos de calidad.

PRUEBAS DE RENDIMIENTO

En la ingeniería del *software*, las pruebas de rendimiento sirven para ver si la aplicación va a ser lo suficientemente rápida en el uso cotidiano.

Es importante pensar en este tema al inicio del desarrollo del proyecto, porque cuanto más avanza el mismo, luego será más costoso solucionar un problema en general, y más aún, uno de rendimiento, porque suele conllevar un cambio de arquitectura en muchos casos.

También es crucial que el escenario donde se realiza la prueba, sea lo más parecido posible al montaje real, aunque esto suele ser difícil de conseguir, dado el carácter aleatorio que tiene el escenario en cliente final, por mucho que se haya intentado definir desde el principio en las especificaciones.

El objetivo final es medir los tiempos de respuesta de todas las transacciones importantes y localizar así los posibles 'cuellos de botella' del sistema.

Una vez que se comprueba que la aplicación resulta útil respondiendo con rapidez en el tiempo esperado, conviene dar un paso más, y realizar pruebas de carga, para conocer qué sucede en el caso en que todos los clientes esperados, se encuentren trabajando al mismo tiempo, y saber así cómo responde la aplicación en tales circunstancias, en estos 'picos de trabajo'.

Y por último, y de forma así muy resumida, es fundamental saber responder a la pregunta de cuál es el límite de usuarios que la aplicación puede soportar. Para esto están las pruebas de estrés. Van a permitir conocer de antemano, antes de la puesta en producción, cuál debe ser el dimensionamiento del entorno real, en cuanto a equipos, red, escalabilidad, repartidores de carga, especificaciones del hardware necesario, etc. para cubrir así hasta un cierto límite de usuarios.

Todo este trabajo, requiere de un costoso proceso de planificación inicial, para elaborar el plan de pruebas a seguir.

Una vez elaborado, llega el turno de la preparación del entorno, con la instalación de las herramientas a utilizar (ejemplo: servidores web, servidores de bases de datos, vmwares, sniffers, etc.), el diseño y desarrollo de los casos de prueba, montar el entorno de pruebas y dotarlo de los suficientes datos de entrada, implementar las tareas automatizadas para la posterior ejecución de la prueba (ejemplo: Visual Studio, Selenium, scripts tipo AutoIt, etc.) , y por último, antes de pasar a la fase de ejecución, realizar una prueba de concepto, aunque sea con un solo usuario, también llamada 'prueba de humo', para ver que todo funciona correctamente, antes de empezar a incrementar el número de usuarios.

Ya en la fase de ejecución, habría que realizar las pruebas de rendimiento, carga

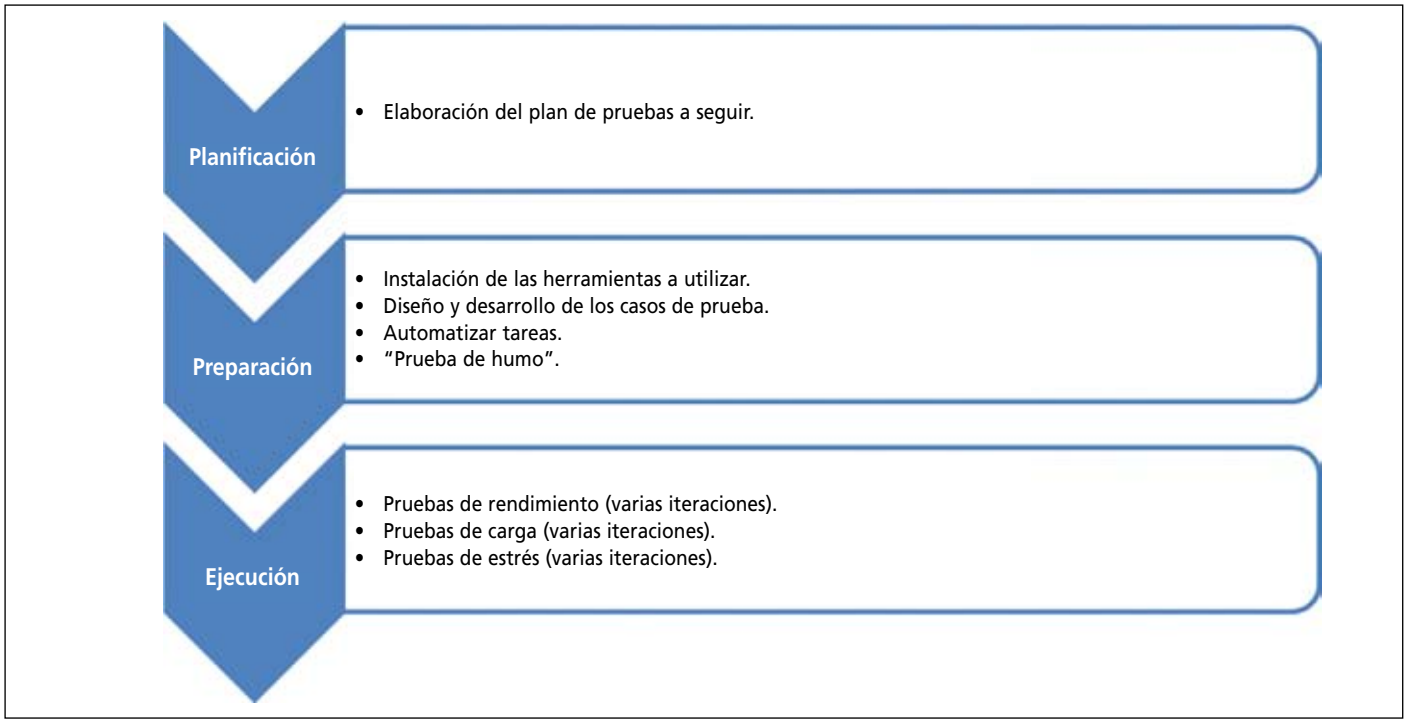

Figura 2: Etapas en el desarrollo de las pruebas de rendimiento.

y estrés, con múltiples ejecuciones de cada una de ellas, para aislar posibles comportamientos puntuales, debidos a circunstancias del entorno en un momento particular. Todos los datos relativos a la ejecución se deben apuntar, para elaborar posteriormente el informe de resultados, así como para futuras ejecuciones en otras versiones de la aplicación. Es fundamental partir de las mismas características para poder comparar resultados en el futuro, y ver si posteriores características añadidas a la aplicación han podido afectar al rendimiento de la funcionalidad previa.

CONCLUSIONES

Se puede concluir que todo este conjunto de pruebas, tanto funcionales como no funcionales, resulta muy necesario antes del paso a producción. El objetivo primordial siempre es tener al cliente satisfecho, y el cliente actual es cada vez más exigente.

En este punto, un requisito de cliente no cumplido no es aceptable, aunque obviamente no todos los requisitos tienen la misma importancia. Por tanto, hay que realizar las pruebas funcionales para asegurar que se cumplen todas las funcionalidades pedidas por el cliente.

Pero además de funcionar y cumplir las especificaciones, tiene que funcionar bien y rápido. Así pues, en el ámbito de las pruebas de rendimiento (pruebas no funcionales), cabe destacar que la tecnología avanza rápido, y lo que hace unos años podía ser normal, por ejemplo que hubiese que esperar 30 segundos a que cargase una página web, hoy no es aceptable, y todos estos parámetros que espera el cliente deben estar claros desde el principio, y deben ser marcados como objetivos principales, para asegurar así la calidad de la entrega final.

REFERENCIAS

- *http://msdn.microsoft.com/en-us/library/ms182561(v=vs.90).aspx*
- *http://www.perftesting.co.uk/*
- *http://msdn.microsoft.com/es-es/library/dd293540.aspx*