El Pepito Grillo de la eficiencia: la cobertura física de las pruebas funcionales

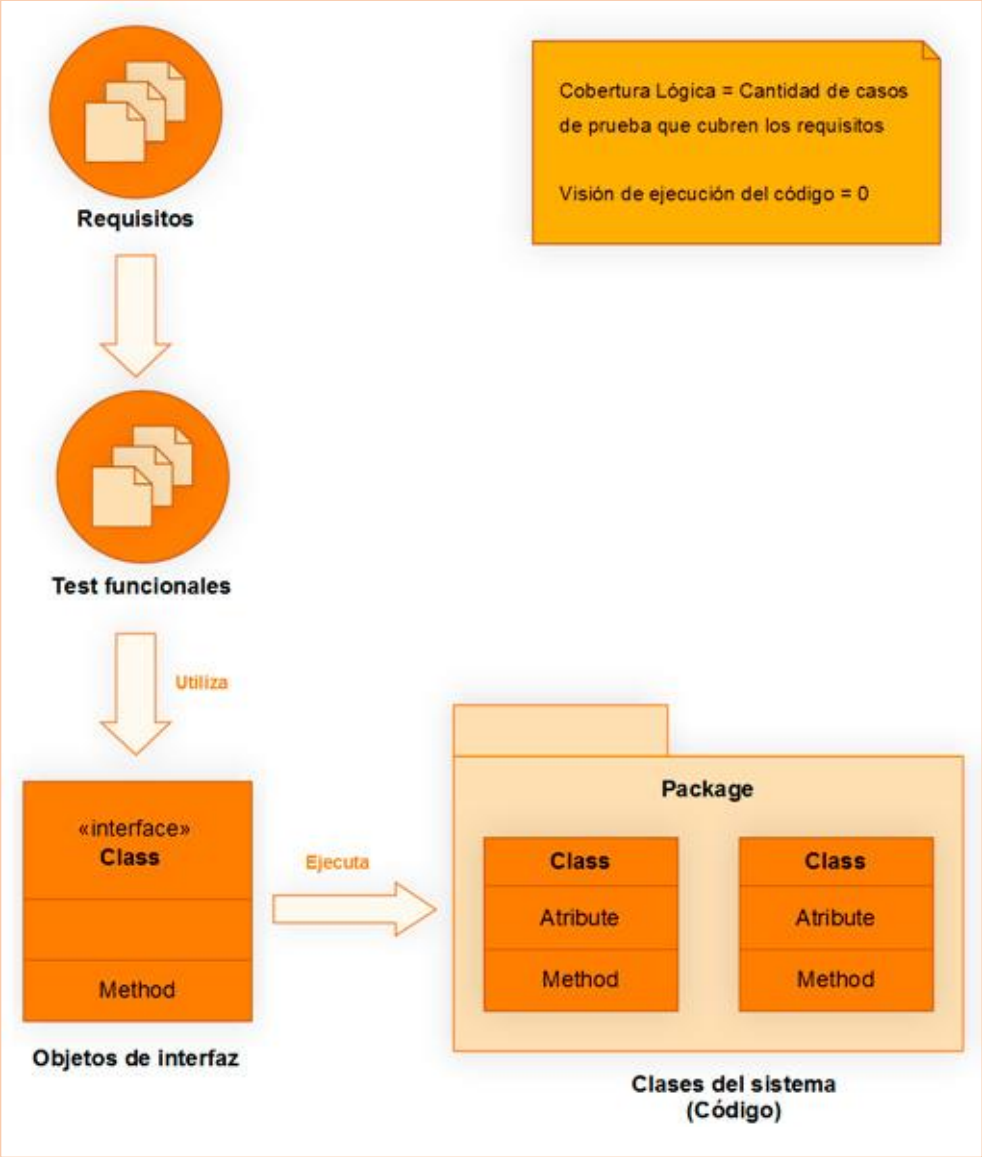
Hoy venimos a hablar de la cobertura en el ámbito del testing. Muchos conocen la cobertura de código, ese indicador porcentual que expresa el grado en que el código fuente de un programa ha sido comprobado. Sin embargo en el mundillo de las pruebas de software, cada vez se pone más de manifiesto la utilización de este indicador aplicado a otros niveles de pruebas, siendo así la aparición de la Cobertura de Testing Funcional representado como el % de casos de pruebas que cubren las funcionalidades del sistema y que muchas empresas aplican pensando que a mayor cobertura, mayor calidad.

Ya en la década de los años 60 Joan C. Miller y Clifford J. Maloney de Communications of the ACM publicaron en su libro «Systematic mistake analysis of digital computer programs» lo que serían las primeras referencias asociadas a los métodos de cobertura de código. A lo largo de estas décadas se ha trabajado potenciando la automatización de estos métodos, como herramienta para la generación de información vital, con vistas a aplicar técnicas de optimización sobre el código no ejecutado por las pruebas, también llamado código muerto; sin embargo en la actualidad, los mecanismos de prueba que utilizan algunas empresas no son capaces de dejar al descubierto este “código muerto” sencillamente por la inexistencia de un test unitario que lo compruebe.

Con la evolución de los procesos y procedimientos de desarrollo de software, incentivados por el sentir agilista, vemos cada día el aumento del rigor en cuanto a las expectativas del % de cobertura de testing y son más las empresas que se suman al movimiento X-Driven Development (XDD), ya sea como metodología de descripción de requisitos, o como metodología de implementación de código. Este movimiento va al alza debido a la reputación que se ha granjeado al potenciar la detección de errores en etapas tempranas de desarrollo, donde estos suelen repercutir en un coste menor, por el aumento del porcentaje de cobertura de testing regresivo. Por otra parte, están las compañías en las que se gestionan las pruebas desde el punto de vista tradicional, dejando el diseño y la implementación de testing de las mismas hacia el final de cada período o etapa, confiando que mientras mayor sea el porcentaje de cobertura de las pruebas sobre la cantidad de funcionalidades, requisitos, métodos o clases, mayor calidad se está asegurando. Sin embargo al igual que se han referido muchos expertos sobre el porcentaje de cobertura en el código, ¿cuándo podemos decir que es fiable el porcentaje de cobertura funcional?

Ante todo, tendríamos que posicionarnos según uno de los dos puntos de vista que hay para aplicar Cobertura de Testing Funcional:

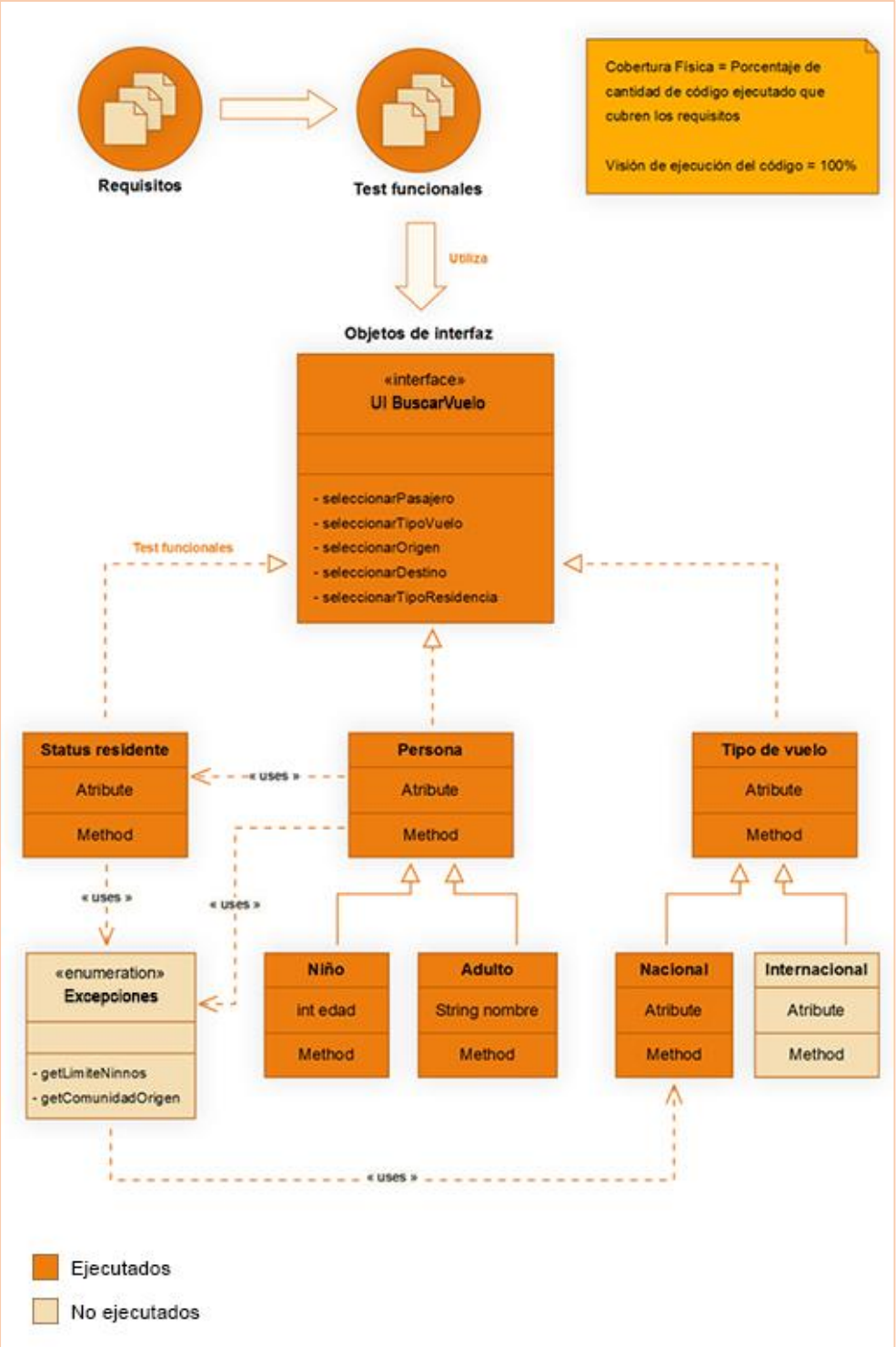
- Cobertura Lógica: porcentaje que se obtiene como resultado de la correspondencia entre los casos de prueba con los requisitos del software.



Aquí estaríamos en consonancia con la Verificación de Software, pero esto no representa un número matemático completamente cerrado, porque X requisitos podrían corresponderse al menos con un número 3X de casos de prueba, teniendo en cuenta la validación de errores, falta de datos, valores límites, entre otros. Este % de cobertura es más conocido como porcentaje de Cobertura de Testing Funcional. En este artículo le hemos llamado Cobertura Lógica porque va directamente relacionado con la lógica del negocio implementada como funcionalidad de la aplicación informática, de la misma manera que en niveles porcentuales de cobertura de código no existe un número exacto, mágico e inamovible para expresar los umbrales de calidad de fiabilidad del testing. Como especialista en la consultoría de calidad en el ámbito del software, lo que sí podría recomendar es que este % tenga como tope mínimo la cantidad de casos de prueba que cubren todos los requisitos de prioridad Alta o Crítica del proyecto, más sus

casos alternos. No obstante este umbral varía según el equipo de desarrollo, sus prioridades, recursos y tiempo.

- Cobertura Física: porcentaje que se obtiene de manera automatizada, mediante herramientas de cobertura que atacan a una build instrumentada mientras se ejecutan pruebas funcionales (automatizadas o no); se calcula de acuerdo a la ejecución de los casos de prueba funcionales sobre la ejecución de X partes del código de la aplicación.



Esto va más en correspondencia con la Validación de Software, en el que podemos detectar si nuestra implementación se está ejecutando de manera correcta, exponer además código muerto, o identificar las funcionalidades que se han implementado y sobre las cuales aún no se han creado casos de prueba que ejecuten esas partes del sistema. En este punto, más que analizar el número de cobertura en sí, se trata de analizar los descampados en el código que no se ejecutan, y encontrar la razón del porqué: ¿no existen casos de pruebas funcionales que realicen acciones que desencadenen este código?, ¿es código muerto?, ¿hay alguna otra razón por la que no se ejecute?

Eventualmente es mucho más fácil y cotidiano la primera versión de cobertura de código para las pruebas funcionales. No obstante desde el punto de vista agilista sería importante incrementar los mecanismos para darle mayor protagonismo a la Cobertura Física de las pruebas funcionales ya que aporta mayor visión sobre nuestro software en ambientes de pruebas, donde muchas veces no se cuenta con herramientas de monitorización de la aplicación en etapas de testing.

Un ejemplo práctico de lo que estamos expresando:

Tenemos una aplicación con la funcionalidad de enviar ficheros, con una dirección de correo electrónico (por poner un ejemplo). Lanzamos los tests funcionales correspondientes a esta funcionalidad y vemos que efectivamente envía los ficheros de manera correcta y sus flujos alternos. Con esto ya nos sentiríamos satisfechos, pero ¿qué ocurre dentro de la aplicación?, ¿se están ejecutando los procesos y procedimientos que se han implementado para ello? En caso de encontrarnos código no ejecutado ¿faltan los casos de prueba que prueben funcionalidades para ejecutar ese código?, ¿es código basura que no hace nada realmente o podría existir un fallo que hace que no se ejecute?

Tristemente, cuando no hemos podido implementar estos mecanismos, nos queda darnos cuenta, en producción, de algunos errores que les van ocurriendo a nuestros usuarios y verlos latentes con la utilización de herramientas de DevOps que nos ayudan a registrar y analizar los logs de errores, pero para entonces, ¿no es un poco tarde?

En términos de eficiencia, ¿no nos gustaría tener un Pepito Grillo que nos indique lo que podemos analizar para ser “más mejores”?

Liudmila Sánchez Almenares

Líder técnico de Centro Experto de Testing & QA de atSistemas

atsistemas
Consulting, IT Services & Software Development

atSistemas cuenta con más de 800 profesionales dedicados desde 1994, a la consultoría, servicios de Tecnologías de la Información y desarrollo de software. Combina especialización en tecnologías de vanguardia con un amplio conocimiento de los retos de negocio de sus clientes.

Más de 300 empresas, entre las principales del país y con presencia en todos los sectores de actividad, han confiado en atSistemas para ofrecerles soluciones innovadoras y acompañarles en su transformación tecnológica.

Desde sus oficinas de Madrid, Barcelona, Cádiz, A Coruña y Zaragoza, realiza proyectos de arquitectura, desarrollo, integración de sistemas y servicios gestionados, adoptando y promoviendo las mejores prácticas del mercado.

Valle de Alcudia, 3 - 1º planta
28232 Las Rozas, Madrid

Passeig de Gràcia 55, 8º - 4ª
08007 Barcelona

902 888 902

atSistemas.com

info@atsistemas.com