FLAKY TESTS

Tras escuchar el podcast de Software Engineering radio donde Gregory Kapfhammer era entrevistado sobre los Flaky Tests, hemos querido destacar y desarrollar los aspectos que consideramos más importantes.

1. Introducción

Un Flaky Test es un caso de prueba que puede pasar unas veces y fallar otras sin modificar el código fuente. Seguramente esto lo hayamos experimentado todos en nuestra vida como desarrolladores. Esto provoca incertidumbre, a menudo obstaculizando la integración y el desarrollo continuo.

Tipos de flaky test:

- Order-dependent flaky test: es un caso de prueba que tiende a pasar y fallar de manera inconsistente cuando se ejecuta en un orden diferente al que originalmente se pretendía.
- Non-Order-dependent flaky test: son casos de prueba que pasan y fallan de manera inconsistente por razones no relacionadas con el orden. Estamos de acuerdo con Gregory, en qué son más difíciles de detectar y provocan más dolores de cabeza que los order-dependent flaky test.

Aspectos que pueden producir fallos en los Non-order-dependent flaky tests

- Fallos en nuestro código: los flaky test no siempre son debido a elementos externos a nuestro código. Algunas veces los flaky test pueden aparecer por errores en nuestro código.
- Cambios significativos en la CPU o la memoria: cambios en estos componentes pueden llevar a que nuestros tests se comporten de manera inestable.
- Modelado de la fecha y la hora: Otro aspecto que Gregory menciona que nos llamó la atención es que, como desarrollador, hay que modelar y manejar correctamente la fecha y la hora, aspecto que puede resultar de gran dificultad.

2. Cómo detectar Flaky Tests

Una vez sabemos en qué consisten los Flaky Tests y lo que conllevan, habrá que intentar detectarlos para poder corregirlos. Para ello, Gregory menciona un procedimiento sencillo que nos pareció fácil de entender y aplicar, que comenzará ejecutando todos los casos de prueba un número determinado de veces.

Si se detecta algún caso de prueba que tiende a pasar o fallar de manera no determinista, es posible que esto sea un Flaky Test. Entonces, habrá que ejecutar estos casos de prueba problemáticos de manera aislada un número determinado de veces para que sea más sencillo detectar por qué no funcionan. Una vez se ejecuten correctamente de manera aislada, habrá que ir ejecutando conjuntamente más tests de forma progresiva.

Cuando se cumplan los pasos anteriores, se podrán variar aspectos del contexto en el que se ejecutan, como la memoria o las CPUs del entorno en el que se ejecutan los casos de prueba. Por último, volveremos a ejecutar todos los tests un número determinado de veces, que nos permitirá saber si hemos conseguido corregir algún Flaky Test o al menos detectarlo.

No obstante, en algunas ocasiones podría resultar tedioso hacer este proceso de forma manual. Para facilitar y automatizar esta labor existen diferentes herramientas y técnicas, como algunas que sugiere Gregory. Kapfhammer menciona herramientas muy interesantes como PyTest, herramienta para Python que ejecuta tests de forma aleatoria; o el uso de Docker y máquinas virtuales que permiten simular entornos de ejecución controlados y aislados, donde se puedan modificar parámetros como la memoria, por ejemplo.

Además de diferentes herramientas, Gregory menciona distintos plugin como Surefire para Maven y hace bastante hincapié en metodologías, por ejemplo, el Machine Learning.

Este método se basa en el aprendizaje automático y creemos que puede tener mucho futuro. Existen varios modelos, aunque todos funcionan de manera similar. Se basa en pasarle información acerca de nuestro programa y nuestros tests para que el algoritmo pueda hacer una predicción. Como es de suponer, cuanta más información se le facilite, más probabilidades habrá de que la predicción sea lo más acertada posible.

De esta manera, teniendo en cuenta tanto el proceso explicado más arriba como todas estas alternativas para intentar automatizar el proceso de detección, nos será más sencillo determinar la presencia de Flaky Tests en nuestro proyecto, para así poder intentar corregirlos.

3. Cómo solucionar Flaky Tests

Si nos decidimos por ver por qué falla un flaky test, ¿a qué deberíamos prestar especial atención?

A la hora de arreglar un flaky test, debemos prestar especial atención a los métodos setUp y tearDown, los cuales deben de evitar en medida de lo posible la inestabilidad. Sin embargo, introducir grandes cantidades de código en estos métodos que se ejecutan antes y después de cada prueba puede provocar ineficiencia a la hora de ejecutar los tests.

¿Qué pasa si la estabilidad del test está fuera de nuestro control, por ejemplo, por usar servicios de terceros?

En esos casos, podemos realizar los tests con objetos que simulen el comportamiento de estos servicios. Si no podemos utilizar estos objetos, otra opción es utilizar estos tests de forma más infrecuente.

¿Eliminar o arreglar un flaky test detectado? ¿Es rentable gastar tiempo en ello?

Una vez detectado un test inestable, surgen dos opciones: asumir que el test no aporta valor y eliminarlo, o invertir tiempo en él para hacerlo estable.

Gregory suele optar por aislar el test y dejar de utilizarlo continuamente, reservándolo para cambios grandes o nuevos lanzamientos. Sin embargo, un flaky test puede indicar un problema en la lógica, por lo que no está de más invertir tiempo en ver al menos por qué falla.

Recomendaciones para evitar la inestabilidad en los tests.

En cuanto a la creación de tests, estos deben de guiarse por eventos y no por tiempos, ya que los elementos pueden tardar más o menos en renderizarse. También debemos orientar los tests a comprobar la existencia de elementos y no su localizaciónm exacta. Por último, estos deben de ser simples y con pocos asertos. Además, en cuanto al diseño del código, se recomienda tener componentes con responsabilidad única, por su facilidad a la hora de probarlos.

4. Bibliografía

SE Radio 572: Gregory Kapfhammer on Flaky Tests

A survey of flaky tests

Herramienta PyTest

Página web Docker