ANÁLISIS DE COMPORTAMIENTO DE CÓDIGO

Jorge López

Martín Fernández

Juan Domínguez

INTRODUCCIÓN SOBRE ANÁLISIS DE COMPORTAMIENTO DE CÓDIGO Y CODESCENE

Cuando hablamos de análisis de código, tendemos a pensar en las técnicas tradicionales de análisis que únicamente se centran en estudiar el código, pero este código fuente es solo una propiedad de las muchas que hay dentro de un proyecto software. Solo con el código, no podremos ser capaces de recoger toda la información relevante ni responder a preguntas como si estamos mejorando o empeorando en términos de calidad. Por hacer un símil, el análisis de código estático solo refleja los síntomas que tiene nuestro código, no la enfermedad.

Es ahí donde nace el concepto de análisis de comportamiento de código, que no solo se centra en el código en sí, sino que identifica patrones en la forma en que un equipo de desarrollo interactúa con la base de código que se está creando. Es decir, en el proyecto son importantes tanto las propiedades del código, como saber de qué forma llegó a ese estado.

Toda esta información se utiliza para detectar dependencias implícitas que son visibles en el propio código y para priorizar la deuda técnica, un concepto bastante relacionado con el análisis y que si recordáis de una de las presentaciones anteriores se trata del trabajo adicional en un proyecto causado por escoger una solución rápida y fácil y que conllevará a una posible refactorización del código en un futuro.

Para llevar a cabo un análisis de comportamiento de código, una de las herramientas que se puede utilizar es CodeScene. Es la herramienta número uno en el mundo para este tipo de análisis desarrollada por Empear AB que se centra en la evolución de todo el sistema en el contexto del trabajo de desarrollo. Nos permite llevar a cabo evaluaciones sobre nuestro proyecto para identificar, priorizar y reducir la deuda técnica.

TECNICAS DE ANALISIS DE COMPORTAMIENTO DE CÓDIGO

Según numerosas investigaciones una alta cantidad de cambios en un fichero implica una disminución en la calidad de este y por tanto errores. Estos cambios pueden venir producidos por tres motivos : Demasiadas responsabilidades en esa clase, se entiende mal el código o es un fichero muy afectado por la aplicación.

Existen tres principales cosas en las que fijarse a la hora de estudiar el comportamiento de nuestra aplicación.

1.-Puntos calientes: Se define como una región de un programa donde ocurre una alta proporción de instrucciones ejecutadas o donde se pasa la mayor parte del tiempo. Su estudio comenzó en los años sesenta con el llamado trazo de salto, el cual consiste en acelerar la velocidad de ejecución imprimiendo cosas en un fichero para ver donde pasa la aplicación su tiempo

2.- Luego, podemos comenzar a determinar la tendencia de complejidad de cada posible objetivo de refactorización. Al igual que los volcanes, podemos categorizar si el punto de acceso está activo (es decir, una

tendencia de complejidad creciente), inactivo (una tendencia de complejidad constante) o extinto (la tendencia de complejidad ha comenzado a disminuir).

3.-El cambio de acoplamiento Es una especie de lógica compartida entre dos funciones/archivos, en la que si cambia uno de ellos, probablemente tendrá que cambiar el otro. Un alto grado de acoplamiento no es necesariamente malo, de hecho, se espera dentro de las pruebas unitarias y sus respectivas clases, pero en otros contextos, debemos comenzar a preguntarnos si es fácilmente perceptible. Por ejemplo, siguiendo el Principio de Proximidad, agrupando funciones que cambian juntas, estamos transmitiendo información que no es posible expresar solo a través del código.

PROBLEMAS PRINCIPALES Y EJEMPLOS

El problema no es técnico, es social. Es decir, siempre habrá deuda técnica, lo importante es cómo lo enfrentamos. El problema es la normalización de la desviación: El proceso gradual a través del cual la práctica o los estándares inaceptables se vuelven aceptables. A medida que el comportamiento desviado se repite sin resultados catastróficos, se convierte en la norma social de la organización. Esta definición expresa exactamente cómo cuando los resultados son ‘aceptables’ y no se han producido errores colosales, tendemos a pensar que no debemos tocarlo porque ‘funciona’, entonces ese código que está ‘aceptable’ pasa a ser la nueva normalidad, lo cual en el futuro nos llevará a un código un poco menos ‘aceptable’, pero tampoco será malo del todo, entonces volverá a pasar a ser la nueva normalidad y así hasta que todo fracase.

Hay que arreglar la causa, no el síntoma. Incluso la NASA, donde un solo error puede provocar la pérdida de vidas, lo ha experimentado varias veces, por ejemplo: Challenger, Columbia. Es decir, si este fenómeno ha sucedido dentro de la Nasa, ¿por qué no iba a suceder en nuestros proyectos del día a día?

El desarrollador es el culpable. Porque se supone que nadie más debe reivindicar un código mantenible atractivo.

“La medida en que puede implementar nuevas funciones sin convocar una gran reunión de personal es la prueba definitiva del éxito de una arquitectura” por Adam Tornhill.

El desarrollo es cada vez más un trabajo en equipo. Sin embargo, la Sociología ha demostrado que cuantos más elementos tiene un equipo, más difícil es coordinarlo y es más probable que haya brechas de comunicación, conocidas como Pérdida de Proceso. Además, somos más susceptibles cuando trabajamos en grupo, porque nuestros valores y toma de decisiones se ven repentinamente influenciados, lo notes o no, por los comportamientos del grupo. Este fenómeno se conoce comúnmente como el efecto espectador y se explica a través de dos aspectos sociales principales:

- Ignorancia pluralista: Es un estado grupal en el que alguna decisión tomada por el grupo, que se acepta públicamente por todos los integrantes, se rechaza en privado por muchos (o todos). Sucede porque cada persona siente que es la única que piensa de esa manera y tiene miedo a comunicarlo por si es la única que piensa así. De esta manera ninguno del grupo hace nada. Por ejemplo, es común en las clases de estudiantes, cuando se tiene miedo de exponer dudas en clase, e incluso en emergencias, donde las víctimas tienen más posibilidades de sobrevivir si hay un solo transeúnte, en lugar de una multitud.
- Difusión de la responsabilidad: representa cómo cuando se trabaja en grupo (sobre todo si son grupos grandes) algunos miembros pueden pensar que al ser un grupo, ellos tienen menos responsabilidad, de tal manera que si todos se sienten así, al final no se hará el trabajo.

Más que centrarnos en qué refactorizar, no debemos subestimar el aparato organizativo. Los estudios han demostrado que practicar la revisión del código y tener un mantenedor principal tiene efectos positivos en la calidad del código. Podemos hacer uso de heurísticas objetivas e imparciales derivadas del análisis de comportamiento que dicen lo que sucede detrás de escena.