BEHAVIORAL CODE ANALYSIS

Introducción

Según Adam Torhill, el BCA (o Behavioral Code Analysis) consiste en el análisis del código desde el lado de las personas, lo que se traduce en un análisis que da mayor importancia a las personas que están detrás del código, que al código en sí.

Este planteamiento a la hora de analizar se distingue de un análisis estático del código en dos principales aspectos:

- Podemos conocer el impacto del mal código, no solo identificarlo como haríamos en un análisis estático.
- Tenemos una nueva dimensión de estudio, las personas, pudiendo hacer un análisis más a fondo de cómo trabajan los equipos y cómo se organizan.

Objetivos

- Priorizar deudas técnicas: El BCA nos permite distinguir deudas técnicas costosas y poco rentables de aquellas cuya resolución nos puede proporcionar un beneficio rentable a su costo.
- Favorecer la comunicación, proporcionando un lenguaje común entre desarrolladores y managers de producto, para mantener a todo el mundo al tanto de los problemas.
- Hacer análisis sociales, midiendo brechas de conocimiento. Estas mediciones consisten en medir la familiaridad del equipo con un código, para distinguir códigos complejos y que deben ser refactorizados, de aquellos no tan complejos, pero con los que el equipo está poco familiarizado. En este último caso, refactorizar no es más que una pérdida de tiempo, y sale más rentable llevar a cabo un proceso de familiarización con el código, ahorrando grandes cantidades de tiempo.

Fuentes de datos

Hay varias fuentes de datos a emplear para hacer los análisis, pero la principal y más importante la tratamos los desarrolladores a diario:

- Sistemas de control de versiones: Son una fuente de datos ideal, nos dice quién ha contribuido, cómo, qué efecto ha tenido, además de mucha más información relevante para poder ser analizada.
- El código en sí.
- Herramientas de gestión de producto.

Cabe mencionar que, en caso de sistemas green-field, tendremos que esperar el tiempo que consideremos necesario para tener una cantidad de datos suficiente como para llevar a cabo un análisis consistente.

Análisis de hotspots

Los análisis de hotspots son la principal forma de conocer y priorizar nuestras deudas técnicas. Consisten en observar el patrón de comportamiento de los desarrolladores, buscando qué partes del código son cambiadas con mayor frecuencia, siendo estos puntos los hotspots, zonas de código complicado donde se trabaja a menudo. Si centramos los esfuerzos en estos hotspots, garantizamos que será un esfuerzo bien empleado y que obtendremos buenos resultados.

Otro aspecto interesante de la identificación de los hotspots es que son lugares donde la calidad del código debe ser priorizada, no como en lugares donde el código es más estable y lleva mucho tiempo sin cambiarse, ya que una refactorización en estos lugares puede dar lugar a otros fallos debido a que el código puede ser antiguo y quizás no recordemos con certeza su funcionamiento, por lo que es mejor hacer un control de daños aunque la calidad no sea la mejor, “si funciona, no lo toques”.

Limitaciones

Otro aspecto interesante que no se ha mencionado hasta ahora es cómo el BCA permite detectar casos de change coupling. Como su propio nombre indica, son cambios emparejados por así decirlo, lugares acoplados del código que hacen que, cuando se cambia código de un sitio, se cambia del otro. Sin embargo, la limitación viene dada por esto, ya que puede identificar que existe esta anomalía, pero no puede distinguir causa de efecto.

Implicaciones éticas

Un gran poder conlleva una gran responsabilidad, y el BCA no es una excepción, debe ser utilizado para buscar la mejora de los equipos, no para analizar individuos y despedir empleados, ya que puede tener efectos adversos. Un claro efecto de ello sería que perderíamos la colaboración del equipo, además de la dinámica de este, ya que los integrantes centrarían su trabajo en la métrica y nada más.

Code Maat

Code Maat es una herramienta open-source creada por Adam Tornhill para realizar estos análisis. Se recomienda emplearla con Git, pero funciona con funcionalidad limitada para otros VCSs. Si queremos integrarla en nuestros proyectos, podemos hacerlo mediante CodeScene, integrándolo en nuestro repositorio para que haga análisis periódicos, bajo demanda o por medio de su integración con Pull Requests.

Bibliografía

1. **Tornhill, Adam.** SE Radio 554: Adam Tornhill on Behavioral Code Analysis. *Software Engineering Radio*.

2. **CodeScene Doc. [En línea] https://codescene.io/docs/index.html.**