Pablo Pérez Álvarez UO294197

Bruno Pérez Cuervo UO295445

Daniel Flores López UO282894

Legacy Code

Podríamos definir el “Legacy Code” como el código de valor que requiere cierto cuidado cambiarlo ya que tiene un gran impacto sobre las personas ya que es el código que se encuentra en producción.

Reescribir o refactorizar

En el podcast se hace una reflexión sobre si es mejor reescribir o refactorizar código, la respuesta corta es que refactorizar es mejor, sin embargo, reescribir partes pequeñas del código del sistema que estemos desarrollando no es una mala práctica y muchas veces será necesario.

La comunicación es importante

Es importante saber comunicar y concienciar de la importancia de refactorizar para así tener un código mantenible, debemos explicar esto en términos de negocio, ya sea resaltando el dinero o la reputación que nos puede costar, incluso los clientes que se pueden perder.

Todo esto es más fácil si tenemos una posición privilegiada dentro de la empresa, podríamos realizar gráficos mostrando la gente que está ayudando a contribuir simplemente dedicando 10 minutos al final de cada Spring, de esta manera se sumará mas gente a hacerlo.

Hotspot analysis

Es una técnica que identifica los archivos o módulos de código que cambian con más frecuencia y que además son complejos. Estos "hotspots" suelen ser buenos candidatos para refactorización porque son frágiles y difíciles de mantener. Se basa en datos históricos del control de versiones (como Git).

Pablo Pérez Álvarez UO294197

Bruno Pérez Cuervo UO295445

Daniel Flores López UO282894

Automated refactoring

Es el proceso de modificar el código usando herramientas del editor o IDE que aseguran que el comportamiento del programa no cambie. Permite tareas como renombrar variables, extraer funciones o mover archivos de forma segura y rápida. Mejora la calidad del código con menor riesgo de errores.

Tu código como una escena del crimen

Es una metáfora,además de ser un libro que invita a analizar el código como lo haría un detective: observando pistas en el historial de cambios, commits, y errores para entender por qué está escrito así. Esta mentalidad ayuda a tomar decisiones informadas antes de modificar un sistema heredado o confuso.

Método Mikado

El método Mikado funciona de la siguiente manera:

Te pones un objetivo, lo escribes. Pones un temporizador de entre 10 a 15 min e intentas completar la tarea. Lo suyo es que sea imposible, por algo te planteas hacer eso.

En ese punto descubres qué has descubierto, que te ha bloqueado para completar la tarea y hasta donde has llegado. Con eso, se descompone la tarea en partes más pequeñas. La idea para la siguiente iteración es escoger una subtarea y todo lo hecho en la anterior iteración desecharlo. Eventualmente, completaras una tarea en 10 min.

Beneficios: Habrás conseguido resolver la mayoría de los bloqueos, ordenando las subtareas a hacer para cumplir el objetivo inicial, tienes un artefacto visualpara saber qué hacer y compartir con tus compañeros la visión que tienes de esa tarea.

Se habla de hacer iteraciones de 10 a 15 min ya que la gracia es que en cada iteración se plantea qué hacer de una manera más atómica y con un mayor contexto

de los problemas, así que mantener el código de cada iteración es contraproducente. Por lo general, las siguientes iteraciones llevan a mejor código.

Moldable development

Pablo Pérez Álvarez UO294197

Bruno Pérez Cuervo UO295445

Daniel Flores López UO282894

Moldable Development, o desarrollo moldeable, es un enfoque del desarrollo de software en el que el entorno de trabajo —como el IDE, las herramientas y las visualizaciones— se personaliza activamente según las necesidades del desarrollador y del problema que se está resolviendo. En lugar de trabajar con herramientas genéricas, el desarrollador adapta (o "moldea") las herramientas para que presenten la información de manera más relevante y útil para cada caso particular, permitiendo una comprensión más profunda y eficiente del sistema.