Tidy First

Acoplamiento

El acoplamiento se refiere a la dependencia entre módulos o componentes. Cuando dos elementos están acoplados, un cambio en uno de ellos obliga a modificar el otro. Beck enfatiza la importancia de reducir el acoplamiento siempre que sea posible. Un alto grado de acoplamiento hace que el sistema sea frágil y costoso de modificar. Según Beck, la clave no es eliminarlo completamente, sino gestionarlo de manera efectiva para equilibrar flexibilidad y funcionalidad.

Cohesión

La cohesión mide cuán relacionadas están las partes internas de un componente. Un módulo cohesionado contiene elementos que cambian juntos. Para mejorar la cohesión, hay que organizar el código de tal manera que los cambios no afecten a elementos no relacionados. Si un módulo tiene baja cohesión, significa que está compuesto por partes dispares. Mantener una alta cohesión facilita la comprensión y modificación del software.

Tidyings

Las *tidyings* son pequeñas mejoras en la estructura del código sin afectar su comportamiento. Realizar estos cambios constantemente ayuda a limpiar el código y modificarlo en el futuro. La filosofía de *Tidy First?* enfatiza que, antes de agregar nuevas funcionalidades, es fundamental asegurarse de que el código es claro y manejable. Esta práctica ayuda a reducir la deuda técnica y evita problemas de mantenimiento a largo plazo.

Diseño de Software Empírico

El diseño de software empírico se basa en la observación y experimentación. Este enfoque sugiere que los cambios deben hacerse en pequeños pasos y evaluarse constantemente para determinar su impacto en la estructura del software. En este método nunca deben mezclarse cambios estructurales y de comportamiento, enfatizando que. En resumen, el diseño empírico se apoya en la iteración continua y el aprendizaje basado en la práctica para mejorar la calidad del software.

Principio de Pareto

El principio de Pareto, también conocido como la regla del 80/20, afirma que aproximadamente el 80% de los resultados en cualquier ámbito son fruto del 20%. Esto, en el desarrollo de software, indica que el 80% de los cambios que se deben realizar sobre nuestro código, se van a hacer sobre un 20% de este. Esto significa que debemos enfocarnos en mejorar esas partes más utilizadas, de forma que el código que modificamos con más frecuencia sea de mayor calidad y requiera menos trabajo realizar los cambios.

Limpieza gradual o grandes refactorizaciones

Estos cambios se podrían hacer en una gran refactorización, ponerse un mes a organizar y limpiar todo el código, pero esto afectaría a la impresión que tienen los clientes o jefes. Estos pueden pensar que no se está trabajando dado que el proyecto no avanza. Una alternativa es hacer los cambios de una forma incremental, mientras se realizan avances visibles en el proyecto. De esta manera, las relaciones entre trabajadores y los interesados en el proyecto no se verían resentidas.

Cambios estructurales y cambios de comportamiento

Kent divide los cambios en dos tipos: los estructurales, que afectan a la organización interna del código, pero no modifican su funcionamiento. Estos cambios tienden a ser fácilmente reversibles, y, por otro lado, los cambios de comportamiento, que requieren más cuidado, puesto que no se pueden revertir tan fácilmente. Modifican lo que realmente hace el software. Kent recomienda hacer primero los cambios estructurales, que facilitan las futuras modificaciones de comportamiento.

Planificación a largo plazo

La planificación a largo plazo es algo esencial, sin embargo Kent Beck opina que se suele “abusar” de ella, muchas veces se toman decisiones de antemano sin todo el contexto por lo que es mejor intentar tomar las decisiones más en el momento aunque eso conlleve hacer algunos cambios, a lo que no hay que tenerle miedo.

La industria impone unas restricciones que van en contra de la planificación a largo plazo. Si se pasa demasiado tiempo sin obtener ingresos, puede que el proyecto se cancele antes incluso de empezar a recuperar lo invertido.

Al decidir si planificar a largo plazo o no, se debe hacer un balance entre los beneficios futuros y el costo inmediato así como su coste de oportunidad.

La equivalencia de Constantine

Toda refactorización para disminuir el acoplamiento conlleva un coste, es clave encontrar un equilibrio, pues lo que importa en la mayoría de productos no es cómo de acoplados esté sino su coste final.

Para finalizar, se habla brevemente de la importancia de dividir los cambios funcionales y estructurales en distintas pull requests y de cómo la IA no nos sustituirá sino que será una herramienta a nuestra disposición.