Zinedine Álvarez Sais
Darío Cristóbal González
Daniel Alonso Fernández

El software como una disciplina de ingeniería.

Desarrollo Software vs Ingeniería de Software

El desarrollo de software y la ingeniería de software son conceptos que, aunque relacionados, presentan diferencias fundamentales. Mientras que el desarrollo de software se centra en la creación de código y la implementación rápida de funcionalidades, la ingeniería de software adopta un enfoque más estructurado y metódico. La clave radica en la rigurosidad, que permite construir sistemas escalables, mantenibles y alineados con los objetivos de negocio.

Para lograr esto, es esencial aplicar principios de diseño bien definidos, metodologías ágiles correctamente entendidas y una planificación que evite caer en la trampa de la improvisación. Sin este enfoque, los proyectos corren el riesgo de acumular deuda técnica, dificultando futuras modificaciones y provocando costos elevados de mantenimiento.

Un recurso ampliamente reconocido en la industria para definir los fundamentos de la ingeniería de software es el SWEBOK (Software Engineering Body of Knowledge), un conjunto de conocimientos que establece las mejores prácticas, principios y estándares de la disciplina. Este marco ofrece una visión integral de las áreas clave que todo ingeniero de software debe dominar, desde la gestión de requisitos hasta el aseguramiento de calidad y la arquitectura de sistemas.

- Información más detallada sobre el SWEBOK: https://www.computer.org/education/bodies-of-knowledge/software-engineering

Resistencia al cambio

Un desafío recurrente en el ámbito del software es la resistencia al cambio, tanto por parte de los equipos de desarrollo como de las organizaciones. A menudo, se prioriza la entrega rápida de características sin considerar las consecuencias a largo plazo. Esto genera una falsa sensación de velocidad, donde los avances iniciales parecen prometedores, pero la acumulación de problemas técnicos ralentiza el desarrollo con el tiempo.

Para mitigar este problema, es fundamental equilibrar la rapidez con la calidad. No se trata de evitar cambios, sino de gestionarlos inteligentemente mediante prácticas como el diseño modular, el uso de patrones de arquitectura y una cultura de pruebas continua. En lugar de hacer "parches rápidos" que solucionen problemas a corto plazo, pero deterioren el sistema, es preferible adoptar un enfoque estratégico que permita adaptarse sin comprometer la estabilidad del software.

La Rigurosidad en la Ingeniería de Software

El software bien diseñado no surge de la casualidad, sino de la aplicación de principios rigurosos. La ingeniería de software debe abordarse con la misma seriedad que otras disciplinas de ingeniería, donde la precisión y la planificación son esenciales. Conceptos como la separación de responsabilidades, la documentación clara y el aseguramiento de

calidad deben ser una prioridad en todo proyecto. Aplicar rigurosidad en el desarrollo de software no significa ser inflexible, sino establecer una base sólida sobre la cual se puedan realizar cambios sin comprometer la estabilidad del sistema. Al adoptar metodologías que permitan detectar errores en fases tempranas, se reducen significativamente los costos de corrección y se mejora la calidad del producto final.

Planificación y estimación

Un aspecto crucial de la ingeniería de software es la capacidad de prever tiempos, costos y riesgos. La planificación no es una simple formalidad, sino una herramienta que permite a los equipos tomar decisiones informadas y evitar desviaciones inesperadas. La tendencia de evitar estimaciones bajo la excusa de la agilidad es contraproducente. Las estimaciones bien realizadas ayudan a detectar problemas antes de que se conviertan en crisis y permiten a las empresas gestionar mejor sus recursos. La clave no está en hacer estimaciones perfectas, sino en utilizarlas como una guía flexible para ajustar los proyectos de manera eficiente.

Diseñar para el cambio

Durante el desarrollo software uno de los aspectos más importantes es diseñar para el cambio. El cambio es la llave principal del desarrollo, en muchas ocasiones mientras nos encontramos en un proyecto conseguimos alcanzar un diseño inicial válido y funcional, pero a lo largo del tiempo este diseño se va degradando y se vuelve más complicado añadir funcionalidades nuevas y cambiar módulos sin destruirlo por completo.

Nos debemos centrar en cómo hacer esto, lidiando con las personas que piensan que no es necesario diseñar para el cambio mientras evitamos caer en especulaciones acerca de cambios que nunca ocurrirán y sin intentar abarcar cualquier posible transformación. Una prueba que podemos hacer para determinar si puede ser fruto de un futuro cambio es decir “Nunca vamos a necesitar eso”.

Una buena metodología para ello es definir de forma clara los requisitos del cliente centrándonos en lo verdaderamente esencial. Diseñando las distintas partes de forma individual y de manera iterativa ir uniéndolas mientras observamos sus conexiones, sus posibles problemas, por ejemplo: "Si ese cambio ocurriera, ¿cómo afectaría esto al diseño del sistema?". Al acabar es ideal atacar el diseño desde distintos ángulos y por distintas personas, es decir ver si resiste bien los cambios y observar sus puntos más débiles, los ingenieros que se encuentran más cerca del código suelen ser capaces de encontrar las dificultades que ocurrirán durante la implementación de dicho diseño, cabe recordar que la ingeniería es un trabajo en equipo.

Todo esto se tiene que llevar a cabo en un periodo razonable, podríamos iterar indefinidamente sin mejorar el resultado o incluso consiguiendo un diseño perfecto pero imposible de implementar.

En muchas ocasiones nos encontramos con partes del diseño que tienen un toque personal y finalmente termina volviéndolo confuso e inconsistente por ello es importante seguir ciertos estándares o convenciones que están comprobadas que funcionan, ahí entran en juego los patrones de diseño, que servirán una base sólida para evitar revisar todo una y otra vez.

- Información en detalle sobre los patrones de diseño y su impacto en la arquitectura del software: https://refactoring.guru/design-patterns/history

Pese a lo que la mayoría suelen pensar no son todo errores de software, sino que los errores en los requisitos y diseño son responsables de más del 50% de los defectos.

- ”*El lugar más barato para encontrar un defecto es en los requisitos.”*- Gerald Weinberg en su libro Exploring Requirements

Según sus estudios es entre mil y diez mil veces más caro encontrar un error en producción que detectarlo en la etapa de requisitos.

Encontrar “bugs” en la fase de producción no es bueno ni para los desarrolladores ni para los clientes, aun así, casi nadie pone energía para encontrar problemas en la fase de definición de los requisitos. Una vez en el error ya se encuentra en el código es mucho más complicado arreglarlo incluso si fuese en una “pull request”. Muchas veces estos errores se solucionan mediante un “hotfix”, es decir una solución rápida al problema, pero lo más probable es que después debamos realizar una corrección más profunda.

De hecho, lo más probable es que mientras vayamos solucionando problemas mediante ”hotfix” estemos introduciendo otros problemas nuevos.

Refactorización vs Rework

En cuanto a la diferencia entre refactorización y reelaboración(rework), la reelaboración es inevitable, nunca vamos a poder reducirla a 0 pero queremos reducirla lo más posible, si pasamos el 50% de nuestro tiempo en reelaboración, los proyectos casi van a costar el doble en términos prácticos. Sin embargo, la refactorización sigue una estrategia de arquitectura evolutiva, básicamente, comienzas con algo funcional y simplemente sigues refactorizando. Eso es casi lo mismo que no planificar. La tendencia natural del software es hacia el desorden y requiere un gran esfuerzo mantenerlo organizado por tanto si no comenzamos con orden y una buena base difícilmente conseguiremos refactorizar hasta convertirlo en algo excelente.

- Refactorización por Martin Fowler: https://refactoring.com/

La *hipótesis de resistencia del diseño* de Martin Fowler muestra en un gráfico la línea del buen diseño en la que puedes entrar y, con suerte, mantenerte. También menciona que existe una trayectoria de mal diseño, donde al principio puedes avanzar rápidamente y mostrar resultados de forma temprana, pero con el tiempo esto conduce al desorden, dificultando la incorporación de nuevas características y ralentizando el desarrollo. Por ello queremos evitar hacer grandes refactorizaciones en los sistemas solo para admitir un nuevo requisito o cambio, es decir queremos mantener esos cambios lo más pequeños posible.

- Explicada en detalle en el blog de Martin Fowler: https://martinfowler.com/bliki/DesignStaminaHypothesis.html

Otra visión de ello es que la refactorización ocurre cuando se mantienen las pruebas y la abstracción no cambia. Si las pruebas deben modificarse, significa que la abstracción está cambiando, lo que ya no es refactorización, sino rework. El problema es que muchas veces se usa el término “refactor” cuando en realidad se están rehaciendo pruebas y cambiando

abstracciones, o peor aún, cuando el código está tan acoplado que cualquier cambio implica una reelaboración significativa.

Aseguramiento de calidad vs control de calidad

Por una parte, el aseguramiento de calidad se enfoca en prevenir los defectos desde que comenzamos el desarrollo y asegurar que se sigan buenas prácticas en todo momento

Por otra parte, el control de calidad se enfoca en identificar defectos después de que el producto este construido, como pueden ser las pruebas unitarias.

Control de calidad

Hablando del control de calidad, un avance muy importante fue el Cordón de Andon, desarrollado y popularizado por Toyota, en este cualquier trabajador de la línea de producción podía detener el proceso si detectaba algún problema. Transfiriéndolo a nuestro campo, los equipos de desarrollo deberían poder detener el proceso si encuentran un problema de calidad. Ya que, si algo no funcionase más adelante, por culpa de estos problemas de calidad, fuese mucho más costoso arreglarlo.

- Explicación más detallada sobre el Cordón de Andón y su aplicación a nuestra disciplina: https://optimizan.com/andon-cord-en-equipos-de-desarrollo-de-software/

Esta calidad debe ser una responsabilidad compartida por todos en la organización, desde los ingenieros de software hasta los gerentes de proyecto y los diseñadores. Si alguna persona de la organización no cumple con los estándares de calidad impuestos podría afectar negativamente en el producto final. Esta calidad debería ser un objetivo constante desde el inicio como bien se habló en el aseguramiento de la calidad.

Rol de “Ingeniero Jefe” y preparación de equipos

Una figura muy importante para asegurar la calidad es el “Ingeniero Jefe”, esta persona debe tener una visión general de todo el proyecto asegurándose que todos los aspectos (requisitos, diseño…) se estén realizando correctamente. Este también tiene que formar a los nuevos miembros ya que en un equipo de software hay cambios constantes de los individuos que lo conforman. Este rol es similar al de un entrenador de futbol ya que estos tienen que guiar tanto a su equipo como estar preparados para que lleguen nuevas incorporaciones y tengan que explicarles todo.

Aparte de un “Ingeniero Jefe” que forme a los nuevos miembros, también es necesario que se prepare a las personas para los cambios, es decir, como hablamos anteriormente hay cambios constantes en los equipos de software por lo que es necesario que haya lideres que puedan ayudar a integrarse a los nuevos miembros en el equipo y para que todos avancen en la misma dirección.

Avances en la disciplina de la ingeniería del software

Zinedine Álvarez Sais
Darío Cristóbal González
Daniel Alonso Fernández

En los últimos años las expectativas de los usuarios han aumentado exponencialmente debido a los avances tecnológicos, lo que hace 20 años era un producto aceptable, ahora se considera un producto insuficiente. Esto añade complejidad al desarrollo software ya que ahora los equipos deben cumplir con estándares más altos y con funcionalidad más avanzadas.

Finalmente, se debe hablar sobre el avance de la disciplina de la ingeniería del software, este es bastante complicado a la par que divertido ya que está en constante cambio, lo que se hacía hace 10 años, ahora puede estar obsoleto. Para el avance de la disciplina es esencial contar con líderes que promuevan las mejores prácticas, la formación continua y la evaluación de las competencias. Esto también ayuda en mejorar la madurez del desarrollo de software. Además, dentro de la comunidad de desarrolladores debería haber una colaboración y aprendizaje en conjunto para poder seguir creciendo y mejorando