DAVID FARLEY

MODERN SOFTWARE ENGINEERING

Doing What Works to **Build Better Software Faster**

Foreword by **TRISHA GEE**

Elogios a la ingeniería de software moderna

“La ingeniería de software moderna lo hace bien y describe las formas en que los profesionales capacitados realmente diseñan el software en la actualidad. Las técnicas que presenta Farley no son rígidas, prescriptivas o lineales, pero están disciplinadas exactamente en la forma que requiere el software: empíricas, iterativas, basadas en retroalimentación, económicas y centradas en ejecutar código”.

—Glenn Vanderburg, director de ingeniería de Nubank

“Hay muchos libros que le dirán cómo seguir una práctica particular de ingeniería de software; este libro es diferente. Lo que Dave hace aquí es exponer la esencia misma de lo que define la ingeniería de software y en qué se diferencia de la simple artesanía. Explica por qué y cómo para dominar la ingeniería de software hay que convertirse en un maestro tanto del aprendizaje como de la gestión de la complejidad, cómo las prácticas que ya existen respaldan eso y cómo juzgar otras ideas según sus méritos de ingeniería de software. Este es un libro para cualquiera que se tome en serio el tratamiento del desarrollo de software como una verdadera disciplina de ingeniería, ya sea que esté comenzando o haya estado creando software durante décadas”.

—Dave Hounslow, ingeniero de software

"Estos son temas importantes y es fantástico tener un compendio que los reúna en un solo paquete".

—Michael Nygard, autor de Release IT, programador profesional y arquitecto de software

“He estado leyendo la copia reseña del libro de Dave Farley y es lo que necesitamos. Debería ser una lectura obligatoria para cualquier persona que aspire a ser ingeniero de software o quiera dominar el oficio. Consejos pragmáticos y prácticos sobre ingeniería profesional. Debería ser una lectura obligatoria en universidades y campamentos de entrenamiento”.

—Bryan Finster, ingeniero distinguido y Arquitecto de flujo de valor en USAF Platform One

Esta página se dejó en blanco intencionalmente.

INGENIERÍA DE SOFTWARE MODERNA

Esta página se dejó en blanco intencionalmente.

INGENIERÍA DE SOFTWARE MODERNA

HACER LO QUE FUNCIONA PARA CONSTRUIR

MEJOR SOFTWARE MÁS RÁPIDO

David Farley

Addison-Wesley

Boston • Columbus • Nueva York • San Francisco • Ámsterdam • Ciudad del Cabo
Dubái • Londres • Madrid • Milán • Múnich • París • Montreal • Toronto • Delhi • Ciudad de México
São Paulo • Sídney • Hong Kong • Seúl • Singapur • Taipei • Tokio

Número de control de la Biblioteca del Congreso: 2021947543

Imagen de portada: spainter_vfx/Shutterstock

ISBN-13: 978-0-13-731491-1
ISBN-10: 0-13-731491-4

El compromiso de Pearson con la diversidad, la equidad y la inclusión

Pearson se dedica a crear contenido libre de prejuicios que refleje la diversidad de todos los estudiantes. Aceptamos las muchas dimensiones de la diversidad, incluidas, entre otras, la raza, el origen étnico, el género, el nivel socioeconómico, la capacidad, la edad, la orientación sexual y las creencias religiosas o políticas.

La educación es una fuerza poderosa para la equidad y el cambio en nuestro mundo. Tiene el potencial de ofrecer oportunidades que mejoren vidas y permitan la movilidad económica. Mientras trabajamos con autores para crear contenido para cada producto y servicio, reconocemos nuestra responsabilidad de demostrar inclusión e incorporar estudios diversos para que todos puedan alcanzar su potencial a través del aprendizaje.
Como empresa de aprendizaje líder en el mundo, tenemos el deber de ayudar a impulsar el cambio y cumplir con nuestro propósito de ayudar a más personas a crear una vida mejor para sí mismos y un mundo mejor.

Nuestra ambición es contribuir decididamente a un mundo donde:

- Todos tienen una oportunidad equitativa y permanente de tener éxito a través del aprendizaje.
- Nuestros productos y servicios educativos son inclusivos y representan la rica diversidad de estudiantes.
- Nuestro contenido educativo refleja con precisión las historias y experiencias de los estudiantes que atender.
- Nuestro contenido educativo genera debates más profundos con los alumnos y los motiva a ampliar su propio aprendizaje (y su visión del mundo).

Si bien trabajamos arduamente para presentar contenido imparcial, queremos escuchar sus inquietudes o necesidades con este producto de Pearson para que podamos investigarlas y abordarlas.

- Comuníquese con nosotros si tiene inquietudes sobre posibles sesgos en https://www.pearson.com/report-bias.html.

Esta página se dejó en blanco intencionalmente.

Me gustaría dedicar este libro a mi esposa Kate y a mis hijos, Tom y Ben.

Kate ha apoyado incondicionalmente mis escritos y mi trabajo durante muchos años y siempre es una compañera intelectualmente estimulante, además de mi mejor amiga.

Tom y Ben son jóvenes a quienes admiro y amo como padre, y mientras trabajaba en este libro, ha sido un placer para mí haber tenido el privilegio de trabajar junto a ellos en varias empresas conjuntas.

Gracias por su ayuda y apoyo.

Esta página se dejó en blanco intencionalmente.

Contenido

Prefacio

Estudié informática en la universidad y, por supuesto, completé varios módulos llamados “ingeniería de software” o variaciones del nombre.

No era nuevo en programación cuando comencé mis estudios y ya había implementado un sistema de inventario completamente funcional para la biblioteca de carreras de mi escuela secundaria. Recuerdo estar extremadamente confundido por la "ingeniería de software". Todo parecía diseñado para obstaculizar la escritura de código y la entrega de una aplicación.

Cuando me gradué a principios de este siglo, trabajé en el departamento de TI de una gran empresa automovilística. Como era de esperar, eran grandes en ingeniería de software. Fue aquí donde vi mi primer diagrama de Gantt (¡pero ciertamente no el último!), y es donde experimenté el desarrollo en cascada. Es decir, vi a los equipos de software dedicar cantidades significativas de tiempo y esfuerzo en las etapas de diseño y recopilación de requisitos y mucho menos tiempo en la implementación (codificación), lo que por supuesto se extendió al tiempo de prueba y luego a las pruebas... bueno, no hubo No queda mucho tiempo para eso.

Parecía que lo que nos dijeron que era “ingeniería de software” en realidad se estaba interponiendo en la creación de aplicaciones de calidad que fueran útiles para nuestros clientes.

Como muchos desarrolladores, sentí que debía haber una manera mejor.

Leí sobre programación extrema y Scrum. Quería trabajar en un equipo ágil y cambié de trabajo varias veces tratando de encontrar uno. Muchos dijeron que eran ágiles, pero a menudo esto se reducía a poner requisitos o tareas en fichas, pegarlas en la pared, llamar a una semana un sprint y luego exigir que el equipo de desarrollo entregara "x" muchas tarjetas en cada sprint para cumplir con algunos plazo arbitrario. Deshacerse del enfoque tradicional de “ingeniería de software” tampoco pareció funcionar.

Diez años después de mi carrera como desarrollador, me entrevisté para trabajar en una bolsa financiera en Londres.
El jefe de software me dijo que hacían programación extrema, incluida TDD y programación por pares.
Me dijo que estaban haciendo algo llamado entrega continua, que era como una integración continua pero hasta llegar a la producción.

Había estado trabajando para grandes bancos de inversión donde la implementación tomaba un mínimo de tres horas y estaba "automatizada" mediante un documento de 12 páginas con pasos manuales a seguir y comandos para escribir. La entrega continua parecía una idea encantadora, pero seguramente no era posible.

El jefe de software era Dave Farley y estaba en el proceso de escribir su libro de entrega continua.
libro cuando me uní a la empresa.

Trabajé con él allí durante cuatro años que me cambiaron la vida y marcaron mi carrera. Realmente hicimos programación en pares, TDD y entrega continua. También aprendí sobre desarrollo impulsado por el comportamiento, pruebas de aceptación automatizadas, diseño impulsado por dominios, separación de preocupaciones, capas anticorrupción, simpatía mecánica y niveles de indirección.

Aprendí a crear aplicaciones de alto rendimiento y baja latencia en Java. Finalmente entendí lo que realmente significaba la notación O grande y cómo se aplicaba a la codificación del mundo real. En resumen, todo lo que había aprendido en la universidad y leído en los libros realmente se utilizó.

Se aplicó de una manera que tenía sentido, funcionó y entregó una aplicación de muy alta calidad y alto rendimiento que ofrecía algo que no estaba disponible anteriormente. Más que eso, estábamos felices en nuestro trabajo y satisfechos como desarrolladores. No trabajamos horas extras, no tuvimos momentos difíciles cerca de los lanzamientos, el código no se volvió más enredado y difícil de mantener durante esos años, y entregamos nuevas funciones y "valor comercial" de manera consistente y regular.

¿Cómo logramos esto? Siguiendo las prácticas que Dave describe en este libro. No se formalizó de esta manera, y Dave claramente ha aportado sus experiencias de muchas otras organizaciones para limitarse a conceptos específicos que son aplicables a una gama más amplia de equipos y dominios comerciales.

Lo que funciona para dos o tres equipos ubicados conjuntamente en una bolsa financiera de alto rendimiento no será exactamente lo mismo que funcione para un gran proyecto empresarial en una empresa manufacturera o para una startup de rápido crecimiento.

En mi función actual como defensor de los desarrolladores, hablo con cientos de desarrolladores de todo tipo de empresas y dominios comerciales, y escucho sus puntos débiles (muchos de ellos, incluso ahora, no muy diferentes a mis propias experiencias hace 20 años) y historias de éxito. Los conceptos que Dave ha cubierto
en este libro son lo suficientemente generales para funcionar en todos estos entornos y lo suficientemente específicos como para ser prácticamente útiles.

Curiosamente, fue después de que dejé el equipo de Dave que comencé a sentirme incómodo con el título de ingeniero de software. No pensé que lo que hacemos como desarrolladores es ingeniería; No pensé que fuera la ingeniería lo que había hecho que ese equipo tuviera éxito. Pensé que la ingeniería era una disciplina demasiado estructurada para lo que hacemos cuando desarrollamos sistemas complejos. Me gusta la idea de que sea un “artesanía”, ya que resume la idea de creatividad y productividad, incluso si no pone suficiente énfasis en el trabajo en equipo que se necesita para trabajar en problemas de software a escala. Leer este libro me ha hecho cambiar de opinión.

Dave explica claramente por qué tenemos ideas erróneas sobre lo que es la ingeniería "real". Muestra cómo la ingeniería es una disciplina basada en la ciencia, pero no tiene por qué ser rígida. Explica cómo se aplican los principios científicos y las técnicas de ingeniería al desarrollo de software y habla de por qué las técnicas basadas en producción que pensábamos que eran ingeniería no son apropiadas para el desarrollo de software.

Lo que me encanta de lo que Dave ha hecho con este libro es que toma conceptos que pueden parecer abstractos y difíciles de aplicar al código real con el que tenemos que trabajar en nuestros trabajos y muestra cómo usarlos como herramientas para pensar en nuestros problemas específicos. .

El libro abarca la complicada realidad del desarrollo de código, o debería decir, de la ingeniería de software: no existe una única respuesta correcta. Las cosas cambiarán. Lo que fue correcto en un momento determinado a veces resulta muy incorrecto incluso poco tiempo después.

La primera mitad del libro ofrece soluciones prácticas no sólo para sobrevivir a esta realidad sino también para prosperar en ella. La segunda mitad aborda temas que algunos podrían considerar abstractos o académicos y muestra cómo aplicarlos para diseñar un código mejor (por ejemplo, más robusto o más mantenible u otras características de “mejor”).

Aquí, diseño no significa en absoluto páginas y páginas de documentos de diseño o diagramas UML, sino que puede ser tan simple como "pensar en el código antes o durante su escritura". (Una de las cosas que noté cuando emparejé la programación con Dave fue el poco tiempo que dedica a escribir el código.

Resulta que pensar en lo que escribimos antes de escribirlo puede ahorrarnos mucho tiempo y esfuerzo).

Dave no evita, ni intenta explicar, ninguna contradicción en el uso de las prácticas juntas o la posible confusión que puede ser causada por una sola. En cambio, debido a que se toma el tiempo para hablar sobre las compensaciones y las áreas comunes de confusión, comprendí por primera vez que es precisamente el equilibrio y la tensión entre estas cosas lo que crea “mejores” sistemas.
Se trata de comprender que estas cosas son pautas, comprender sus costos y beneficios, y pensar en ellas como lentes que se pueden usar para observar el código/diseño/arquitectura y, ocasionalmente, diales para girar, en lugar de ser binarios, en blanco y negro, ¿verdad? -o-reglas incorrectas.

Leer este libro me hizo comprender por qué tuvimos tanto éxito y satisfacción como “ingenieros de software” durante el tiempo que trabajé con Dave. Espero que al leer este libro se beneficie de la experiencia y los consejos de Dave, sin tener que contratar a Dave Farley para su equipo.

¡Feliz ingeniería!

—Trisha Gee, defensora de los desarrolladores y defensora de Java

Esta página se dejó en blanco intencionalmente.

Prefacio

Este libro devuelve la ingeniería a la ingeniería de software. En él, describo un enfoque práctico para el desarrollo de software que aplica un estilo de pensamiento científico conscientemente racional para resolver problemas. Estas ideas surgen de la aplicación consistente de lo que hemos aprendido sobre el desarrollo de software durante las últimas décadas.

Mi ambición con este libro es convencerle de que la ingeniería tal vez no sea lo que cree que es y que es completamente apropiada y eficaz cuando se aplica al desarrollo de software. Luego procederé a describir los fundamentos de este enfoque de ingeniería del software y cómo y por qué funciona.

No se trata de las últimas modas en procesos o tecnología, sino de enfoques prácticos y probados en los que tenemos los datos que nos muestran qué funciona y qué no.

Trabajar de forma iterativa en pequeños pasos funciona mejor que no hacerlo. Organizar nuestro trabajo en una serie de pequeños experimentos informales y recopilar comentarios para informar nuestro aprendizaje nos permite proceder de manera más deliberada y explorar los espacios de problemas y soluciones que habitamos. Compartimentar nuestro trabajo para que cada parte esté enfocada, clara y comprensible nos permite hacer evolucionar nuestros sistemas de manera segura y deliberada incluso cuando no entendemos el destino antes de comenzar.

Este enfoque nos proporciona orientación sobre dónde centrarnos y en qué centrarnos, incluso cuando no sabemos las respuestas. Mejora nuestras posibilidades de éxito, cualquiera que sea la naturaleza del desafío que se nos presente.

En este libro, defino un modelo de cómo nos organizamos para crear software excelente y cómo podemos hacerlo de manera eficiente y a cualquier escala, tanto para sistemas genuinamente complejos como para sistemas más simples.

Siempre ha habido grupos de personas que han hecho un trabajo excelente. Nos hemos beneficiado de pioneros innovadores que nos han mostrado lo que es posible. Sin embargo, en los últimos años nuestra industria ha aprendido a explicar mejor lo que realmente funciona. Ahora entendemos mejor qué ideas son más genéricas y pueden aplicarse más ampliamente, y tenemos datos para respaldar este aprendizaje.

Podemos crear software de manera más confiable, mejor y más rápido, y tenemos datos para respaldarlo. Podemos resolver problemas difíciles de clase mundial, y tenemos experiencia con muchos proyectos y empresas exitosos que también respaldan esas afirmaciones.

Este enfoque reúne una colección de ideas fundamentales importantes y se basa en el trabajo anterior. En cierto nivel, no hay nada nuevo aquí en términos de prácticas novedosas, pero el enfoque que describo reúne ideas y prácticas importantes en un todo coherente y nos brinda principios sobre los cuales se puede construir una disciplina de ingeniería de software.

Esta no es una colección aleatoria de ideas dispares. Estas ideas están íntimamente entrelazadas y se refuerzan mutuamente. Cuando se combinan y se aplican consistentemente a la forma en que pensamos, organizamos y llevamos a cabo nuestro trabajo, tienen un impacto significativo en la eficiencia y la calidad de ese trabajo. Esta es una forma fundamentalmente diferente de pensar sobre qué es lo que hacemos, aunque cada idea aisladamente pueda resultarnos familiar. Cuando estas cosas se juntan y se aplican como principios rectores para la toma de decisiones en software, representa un nuevo paradigma para el desarrollo.

Estamos aprendiendo lo que realmente significa la ingeniería de software y no siempre es lo que esperábamos.

La ingeniería consiste en adoptar un enfoque científico y racionalista para resolver problemas prácticos dentro de limitaciones económicas, pero eso no significa que dicho enfoque sea teórico o burocrático. Casi por definición, la ingeniería es pragmática.

Los intentos anteriores de definir la ingeniería de software han cometido el error de ser demasiado proscriptivos y definir herramientas o tecnologías específicas. La ingeniería de software es más que el código que escribimos y las herramientas que utilizamos. La ingeniería de software no es ingeniería de producción de ninguna forma; ese no es nuestro problema. Si cuando digo ingeniería te hace pensar en burocracia, lee este libro y piénsalo de nuevo.

La ingeniería de software no es lo mismo que la informática, aunque a menudo las confundimos.
Necesitamos tanto ingenieros de software como informáticos. Este libro trata sobre la disciplina, el proceso y las ideas que debemos aplicar para crear un mejor software de manera confiable y repetible.

Para ser digno de ese nombre, esperaríamos una disciplina de ingeniería para el software que nos ayudara a resolver los problemas que enfrentamos con mayor calidad y más eficiencia.

Este enfoque de ingeniería también nos ayudaría a resolver problemas en los que aún no hemos pensado, utilizando tecnologías que aún no se han inventado. Las ideas de tal disciplina serían generales, duraderas y omnipresentes.

Este libro es un intento de definir una colección de ideas estrechamente relacionadas e interrelacionadas. Mi objetivo es reunirlos en algo coherente que podamos tratar como un enfoque que informe casi todas las decisiones que tomamos como desarrolladores de software y equipos de desarrollo de software.

La ingeniería de software como concepto, para que tenga algún significado, debe brindarnos una ventaja, no simplemente una oportunidad de adoptar nuevas herramientas.

No todas las ideas son iguales. Hay buenas ideas y hay malas ideas, entonces, ¿cómo podemos notar la diferencia? ¿Qué principios podríamos aplicar que nos permitan evaluar cualquier idea nueva en software y desarrollo de software y decidir si probablemente será buena o mala?

Cualquier cosa que pueda clasificarse justificadamente como un enfoque de ingeniería para resolver problemas de software será de aplicación general y de alcance fundamental. Este libro trata sobre esas ideas. ¿Qué criterios debería utilizar para elegir sus herramientas? ¿Cómo deberías organizar tu trabajo? ¿Cómo debería organizar los sistemas que construye y el código que escribe para aumentar sus posibilidades de éxito en su creación?

¿Una definición de ingeniería de software?

En este libro afirmo que deberíamos pensar en la ingeniería de software en estos términos:

> La ingeniería de software es la aplicación de un enfoque científico empírico para encontrar soluciones económicas y eficientes a problemas prácticos de software.

Mi objetivo es ambicioso. Quiero proponer un esquema, una estructura, un enfoque que podríamos considerar como una auténtica disciplina de ingeniería del software. En el fondo, esto se basa en tres ideas clave.

- La ciencia y su aplicación práctica “ingeniería” son herramientas vitales para lograr un progreso efectivo en disciplinas técnicas.
- Nuestra disciplina es fundamentalmente de aprendizaje y descubrimiento, por lo que necesitamos convertirnos en expertos en aprender para tener éxito, y la ciencia y la ingeniería son las formas en que aprendemos de manera más efectiva.
- Finalmente, los sistemas que construimos son a menudo complejos y lo son cada vez más. Es decir, para hacer frente a su desarrollo, debemos convertirnos en expertos en gestionar esa complejidad.

¿Qué hay en este libro?

La Parte I, “¿Qué es la ingeniería de software?”, comienza analizando lo que realmente significa ingeniería en el contexto del software. Se trata de los principios y la filosofía de la ingeniería y de cómo podemos aplicar estas ideas al software. Esta es una filosofía técnica para el desarrollo de software.

La Parte II, “Optimizar para el aprendizaje”, analiza cómo organizamos nuestro trabajo para permitirnos avanzar en pequeños pasos. ¿Cómo evaluamos si estamos logrando buenos avances o simplemente creando hoy el sistema heredado del mañana?

La Parte III, “Optimizar para gestionar la complejidad”, explora los principios y técnicas necesarios para gestionar la complejidad. En este se explora con más profundidad cada uno de estos principios y su significado y aplicabilidad en la creación de software de alta calidad, cualquiera que sea su naturaleza.

La sección final, Parte IV, “Herramientas para respaldar la ingeniería en software”, describe las ideas y enfoques de trabajo que maximizan nuestras oportunidades de aprender y facilitan nuestra capacidad de progresar en pequeños pasos y gestionar la complejidad de nuestros sistemas a medida que crecen. .

A lo largo de este libro, como barras laterales, se encuentran reflexiones sobre la historia y la filosofía de la ingeniería de software y cómo ha progresado el pensamiento. Estos encartes proporcionan un contexto útil para muchas de las ideas de este libro.

Esta página se dejó en blanco intencionalmente.

Expresiones de gratitud

Escribir un libro como este requiere mucho tiempo, mucho trabajo y la exploración de numerosas ideas. Las personas que me ayudaron durante ese proceso me ayudaron de muchas maneras diferentes, a veces estando de acuerdo conmigo y reforzando mis convicciones y otras veces estando en desacuerdo y obligándome a fortalecer mis argumentos o a cambiar de opinión.

Me gustaría comenzar agradeciendo a mi esposa, Kate, quien me ha ayudado en todo tipo de formas. Aunque Kate no es una profesional del software, leyó gran parte de este libro, lo que me ayudó a corregir mi gramática y perfeccionar mi mensaje.

Me gustaría agradecer a mi cuñado, Bernard McCarty, por intercambiar ideas sobre el tema de la ciencia y hacerme profundizar más para pensar por qué quería hablar sobre experimentación y empirismo, así como sobre muchas otras cosas.

Me gustaría agradecer a Trisha Gee no sólo por escribir un prólogo tan bonito, sino también por mostrarse entusiasmada con este libro cuando necesitaba un impulso.

Me gustaría agradecer a Martin Thompson por estar siempre ahí para intercambiar opiniones sobre informática y por responder generalmente a mis pensamientos bastante aleatorios en minutos.

Me gustaría agradecer a Martin Fowler, quien a pesar de estar demasiado comprometido con otros proyectos, me dio consejos que ayudaron a fortalecer este libro.

Muchos más de mis amigos han contribuido indirectamente a lo largo de los años para ayudarme a dar forma a mi pensamiento sobre estos temas, y muchos más: Dave Hounslow, Steve Smith, Chris Smith, Mark Price, Andy Stewart, Mark Crowther, Mike Barker y muchos otros.

Me gustaría agradecer al equipo de Pearson por su ayuda y apoyo durante el proceso de publicación de este libro.

También me gustaría agradecer a un montón de personas (no todas las que conozco) que me han apoyado, discutido, desafiado y reflexivo. He compartido muchas de estas ideas en Twitter y en mi canal de YouTube durante algunos años y, como resultado, he participado en excelentes conversaciones. ¡Gracias!

Esta página se dejó en blanco intencionalmente.

Sobre el Autor

David Farley es un pionero de la entrega continua, líder intelectual y profesional experto en entrega continua, DevOps, TDD y desarrollo de software en general.

Dave ha sido programador, ingeniero de software, arquitecto de sistemas y líder de equipos exitosos durante muchos años, desde los primeros días de la informática moderna, tomando esos principios fundamentales de cómo funcionan las computadoras y el software y dando forma a enfoques innovadores que han cambiado la forma en que vivimos. abordar el desarrollo de software moderno. Ha desafiado el pensamiento convencional y ha liderado equipos para crear software de clase mundial.

Dave es coautor del libro ganador del premio Jolt Continuous Delivery, es un popular conferencista y dirige el popular y exitoso canal de YouTube "Continuous Delivery" sobre el tema de la ingeniería de software. Creó uno de los intercambios financieros más rápidos del mundo y es pionero de BDD, autor del Reactive Manifesto y ganador del premio Duke por software de código abierto con LMAX Disruptor.

A Dave le apasiona ayudar a los equipos de desarrollo de todo el mundo a mejorar el diseño, la calidad y la confiabilidad de su software compartiendo su experiencia a través de su consultoría, su canal de YouTube y sus cursos de capacitación.

Gorjeo:	@davefarley77
Canal de Youtube:	https://bit.ly/CDonYT
Blog:	http://www.davefarley.net
Página Web de la compañía:	https://www.entrega-continua.co.uk

Esta página se dejó en blanco intencionalmente.

3

Fundamentos de un enfoque de ingeniería

La ingeniería en diferentes disciplinas varía. La construcción de puentes no es lo mismo que la ingeniería aeroespacial, ni tampoco la ingeniería eléctrica o la ingeniería química, pero todas estas disciplinas comparten algunas ideas comunes. Todos ellos están firmemente basados en el racionalismo científico y adoptan un enfoque pragmático y empírico para lograr avances.

Si queremos lograr nuestro objetivo de tratar de definir una colección de pensamientos, ideas, prácticas y comportamientos duraderos que podríamos agrupar colectivamente bajo el nombre de ingeniería de software, estas ideas deben ser bastante fundamentales para la realidad del software. desarrollo y solidez frente al cambio.

¿Una industria de cambio?

Hablamos mucho sobre el cambio en nuestra industria. Nos entusiasman las nuevas tecnologías y los nuevos productos, pero ¿estos cambios realmente "mueven el dial" en el desarrollo de software? Muchos de los cambios que nos ejercitan no parecen hacer tanta diferencia como a veces pensamos que harán.

Mi ejemplo favorito de esto fue demostrado en una hermosa presentación de conferencia por "Christin Gorman". 1 En ella, Christin demuestra que cuando se utiliza la entonces popular biblioteca de mapeo relacional de objetos de código abierto Hibernate, en realidad era más código para escribir que el equivalente. comportamiento escrito en SQL, al menos subjetivamente; el SQL también fue más fácil de entender. Christin continúa contrastando de manera divertida el desarrollo de software con la elaboración de pasteles. ¿Haces tu pastel con una mezcla para pastel o eliges ingredientes frescos y lo haces desde cero?

1. Fuente: "Gordon Ramsay no usa mezclas para pasteles" por Christin Gorman, https://bit.ly/3g02cWO

Gran parte del cambio en nuestra industria es efímero y no mejora las cosas. Algunos, como en el ejemplo de Hibernate, en realidad empeoran las cosas.

Mi impresión es que nuestra industria lucha por aprender y por progresar. este pariente
La falta de avances ha quedado enmascarada por el increíble progreso que se ha logrado en el hardware en el que se ejecuta nuestro código.

No quiero decir que no haya habido avances en el software (ni mucho menos), pero sí creo que el ritmo del progreso es mucho más lento de lo que muchos de nosotros pensamos. Considere, por un momento, qué cambios en su carrera han tenido un impacto significativo en su manera de pensar y practicar el desarrollo de software. ¿Qué ideas marcaron la diferencia en la calidad, escala o complejidad de los problemas que puedes resolver?

La lista es más corta de lo que solemos suponer.

Por ejemplo, he empleado entre 15 y 20 lenguajes de programación diferentes durante mi carrera profesional. Aunque tengo preferencias, sólo dos cambios de lenguaje han cambiado radicalmente mi forma de pensar sobre el software y el diseño.

Esos pasos fueron el paso de Assembler a C y el paso de la programación procedimental a OO.
En mi opinión, los lenguajes individuales son menos importantes que el paradigma de programación. Esos pasos representaron cambios significativos en el nivel de abstracción con el que podía lidiar al escribir código. Cada uno representó un cambio radical en la complejidad de los sistemas que podríamos construir.

Cuando Fred Brooks escribió que no había ganancias de orden de magnitud, se le pasó algo por alto. Puede que no haya ganancias 10x, pero ciertamente hay pérdidas 10x.

He visto organizaciones que estaban paralizadas por su enfoque del desarrollo de software, a veces por la tecnología, más a menudo por el proceso. Una vez hice consultoría en una gran organización que no había lanzado ningún software a producción durante más de cinco años.

No sólo parece que nos resulta difícil aprender nuevas ideas; Parece que nos resulta casi imposible descartar viejas ideas, por muy desacreditadas que puedan haberse vuelto.

La importancia de la medición

Una de las razones por las que nos resulta difícil descartar malas ideas es que en realidad no medimos nuestro desempeño en el desarrollo de software de manera muy efectiva.

La mayoría de las métricas aplicadas al desarrollo de software son irrelevantes (velocidad) o, a veces, positivamente dañinas (líneas de código o cobertura de pruebas).

En los círculos de desarrollo ágil se ha mantenido durante mucho tiempo la opinión de que no es posible medir el rendimiento del equipo de software o del proyecto. Martin Fowler escribió sobre un aspecto de esto en su muy leído Bliki en 2003.2

2. Fuente: “No se puede medir la productividad” de Martin Fowler, https://bit.ly/3mDO2fB

El punto de Fowler es correcto; No tenemos una medida defendible para la productividad, pero eso no es lo mismo que decir que no podemos medir nada útil.

El valioso trabajo realizado por Nicole Fosgren, Jez Humble y Gene Kim en los informes “State of DevOps”3 y en su libro Accelerate: The Science of Lean Software & DevOps4 representa un importante paso adelante para poder generar evidencia más sólida y sólida. -Decisiones basadas en Presentan un modelo interesante y convincente para la medición útil del desempeño de los equipos de software.

Curiosamente, no intentan medir la productividad; más bien, evalúan la eficacia de los equipos de desarrollo de software en función de dos atributos clave. Luego, las medidas se utilizan como parte de un modelo predictivo. No pueden probar que estas medidas tengan una relación causal con el desempeño de los equipos de desarrollo de software, pero pueden demostrar una correlación estadística.

Las medidas son estabilidad y rendimiento. Los equipos con alta estabilidad y alto rendimiento se clasifican como "de alto rendimiento", mientras que los equipos con puntuaciones bajas en estas medidas son "de bajo rendimiento".

Lo interesante es que si se analizan las actividades de estos grupos de alto y bajo desempeño, están consistentemente correlacionadas. Los equipos de alto rendimiento comparten comportamientos comunes. Del mismo modo, si observamos las actividades y comportamientos de un equipo, podemos predecir su puntuación en comparación con estas medidas, y esto también está correlacionado. Algunas actividades se pueden utilizar para predecir el desempeño en esta escala.

Por ejemplo, si su equipo emplea automatización de pruebas, desarrollo basado en troncales, automatización de implementación y unas diez prácticas más, su modelo predice que practicará la entrega continua. Si practica la entrega continua, el modelo predice que tendrá un "alto rendimiento" en términos de desempeño de entrega de software y desempeño organizacional.

Alternativamente, si observamos organizaciones que se consideran de alto desempeño, entonces hay comportamientos comunes, como la entrega continua y la organización en equipos pequeños, que comparten.

Las medidas de estabilidad y rendimiento, entonces, nos brindan un modelo que podemos usar para predecir los resultados del equipo.

La estabilidad y el rendimiento se rastrean mediante dos medidas.

La estabilidad se rastrea mediante lo siguiente:

- Tasa de fracaso del cambio: la tasa a la que un cambio introduce un defecto en un punto particular del el proceso
- Tiempo de falla de recuperación: cuánto tiempo se tarda en recuperarse de una falla en un punto particular del proceso.

---

3. Fuente: Nicole Fosgren, Jez Humble, Gene Kim, https://bit.ly/2PWyjw7

4. El Accelerate Book describe cómo los equipos que adoptan un enfoque más disciplinado para el desarrollo gastan “44% más tiempo en nuevos trabajos” que los equipos que no lo hacen. Consulte https://amzn.to/2YYf5Z8.

Medir la estabilidad es importante porque en realidad es una medida de la calidad del trabajo realizado. No dice nada sobre si el equipo está construyendo las cosas correctas, pero sí mide su efectividad en la entrega de software con una calidad mensurable.

El rendimiento se rastrea mediante lo siguiente:

- Lead Time: Una medida de la eficiencia del proceso de desarrollo. ¿Cuánto dura una sola línea? ¿Cambiar para pasar de “idea” a “software funcional”?
- Frecuencia: Una medida de velocidad. ¿Con qué frecuencia se implementan los cambios en producción?

El rendimiento es una medida de la eficiencia de un equipo a la hora de entregar ideas, en forma de software funcional.

¿Cuánto tiempo lleva hacer que un cambio llegue a manos de los usuarios y con qué frecuencia se logra? Esto es, entre otras cosas, una indicación de las oportunidades de aprendizaje de un equipo. Es posible que un equipo no aproveche esas oportunidades, pero sin una buena puntuación en rendimiento, las posibilidades de aprendizaje de cualquier equipo se reducen.

Estas son medidas técnicas de nuestro enfoque de desarrollo. Responden a las preguntas “¿cuál es la calidad de nuestro trabajo?” y "¿con qué eficiencia podemos producir un trabajo de esa calidad?"

Estas son ideas significativas, pero dejan algunos vacíos. No dicen nada sobre si estamos construyendo las cosas correctas, sólo si las estamos construyendo bien, pero el hecho de que no sean perfectas no disminuye su utilidad.

Curiosamente, el modelo correlativo que describí va más allá de predecir el tamaño del equipo y si se está aplicando la entrega continua. Los autores de Accelerate tienen datos que muestran correlaciones significativas con cosas mucho más importantes.

Por ejemplo, las organizaciones formadas por equipos de alto rendimiento, basadas en este modelo, ganan más dinero que las organizaciones que no lo hacen. Aquí hay datos que dicen que existe una correlación entre un enfoque de desarrollo y el resultado comercial para la empresa que lo practica.

También disipa la creencia común de que "se puede tener velocidad o calidad, pero no ambas". Esto simplemente no es cierto. La velocidad y la calidad están claramente correlacionadas en los datos de esta investigación.
El camino hacia la velocidad es el software de alta calidad, el camino hacia el software de alta calidad es la velocidad de retroalimentación y el camino hacia ambos es la gran ingeniería.

Aplicar estabilidad y rendimiento

La correlación de buenas puntuaciones en estas medidas con resultados de alta calidad es importante. Nos ofrece la oportunidad de utilizarlos para evaluar cambios en nuestro proceso, organización, cultura o tecnología.

Imaginemos, por ejemplo, que nos preocupa la calidad de nuestro software. ¿Cómo podríamos mejorarlo? Podríamos decidir hacer un cambio en nuestro proceso. Agreguemos un tablero de aprobación de cambios (CAB).

Es evidente que la adición de revisiones y aprobaciones adicionales tendrá un impacto adverso en el rendimiento, y tales cambios inevitablemente ralentizarán el proceso. Sin embargo, ¿aumentan la estabilidad?

Para este ejemplo en particular, los datos están disponibles. Quizás sea sorprendente que los comités de aprobación de cambios no mejoren la estabilidad. Sin embargo, la desaceleración del proceso afecta negativamente a la estabilidad.

> Descubrimos que las aprobaciones externas se correlacionaban negativamente con el tiempo de entrega, la frecuencia de implementación y el tiempo de restauración, y no tenían correlación con la tasa de fracaso de los cambios. En resumen, la aprobación por parte de un organismo externo (como un gerente o un CAB) simplemente no funciona para aumentar la estabilidad de los sistemas de producción, medida por el tiempo necesario para restaurar el servicio y cambiar la tasa de fallas. Sin embargo, ciertamente ralentiza las cosas. De hecho, es peor que no tener ningún proceso de aprobación de cambios.5

Mi verdadero objetivo aquí no es burlarme de los comités de aprobación de cambios, sino más bien mostrar la importancia de tomar decisiones basadas en evidencia y no en conjeturas.

No es obvio que los CAB sean una mala idea. Parecen sensatas y, en realidad, así es como muchas organizaciones, probablemente la mayoría, intentan gestionar la calidad. El problema es que no funciona.

Sin una medición eficaz, no podemos decir que no funciona; sólo podemos hacer conjeturas.

Si vamos a empezar a aplicar un enfoque científicamente más racional y basado en la evidencia para la toma de decisiones, no deberían confiar en mi palabra, ni en la de Forsgren y sus coautores, sobre esto ni sobre cualquier otra cosa.

En cambio, podrías realizar esta medición tú mismo, en tu equipo. Mida el rendimiento y la estabilidad de su enfoque actual, cualquiera que sea. Haz un cambio, sea cual sea.
¿El cambio mueve el dial en alguna de estas medidas?

Puede leer más sobre este modelo correlativo en el excelente libro Accelerate. Describe el enfoque de medición y el modelo que está evolucionando a medida que continúa la investigación. Lo que quiero decir aquí no es duplicar esas ideas, sino señalar el impacto importante, tal vez incluso profundo, que esto debería tener en nuestra industria. Por fin tenemos una útil vara de medir.

Podemos utilizar este modelo de estabilidad y rendimiento para medir el efecto de cualquier cambio.

Podemos ver el impacto de los cambios en la organización, los procesos, la cultura y la tecnología. "Si adopto este nuevo lenguaje, ¿aumenta mi rendimiento o mi estabilidad?"

También podemos utilizar estas medidas para evaluar diferentes partes de nuestro proceso. "Si tengo una cantidad significativa de pruebas manuales, ciertamente serán más lentas que las pruebas automatizadas, pero ¿mejora la estabilidad?"

Todavía tenemos que pensar detenidamente. Necesitamos considerar el significado de los resultados. ¿Qué significa si algo reduce el rendimiento pero aumenta la estabilidad?

Sin embargo, contar con medidas significativas que nos permitan evaluar acciones es importante, incluso vital, para adoptar un enfoque más basado en evidencia para la toma de decisiones.

---

5. Acelerar por Nicole Forsgren, Jez Humble y Gene Kim, 2018

Los fundamentos de una disciplina de ingeniería de software

Entonces, ¿cuáles son algunas de estas ideas fundamentales? ¿Cuáles son las ideas que podríamos esperar que sean correctas dentro de 100 años y aplicables cualquiera que sea nuestro problema y cualquiera que sea nuestra tecnología?

Hay dos categorías: proceso, o tal vez incluso enfoque filosófico, y técnica o diseño.

Más simplemente, nuestra disciplina debería centrarse en dos competencias básicas.

Deberíamos convertirnos en expertos en aprender. Deberíamos reconocer y aceptar que nuestra disciplina es una disciplina de diseño creativo y no tiene una relación significativa con la ingeniería de producción y, en cambio, centrarnos en el dominio de las habilidades de exploración, descubrimiento y aprendizaje. Esta es una aplicación práctica de un estilo de razonamiento científico.

También debemos centrarnos en mejorar nuestras habilidades para gestionar la complejidad. Construimos sistemas que no caben en nuestras cabezas. Construimos sistemas a gran escala con grandes grupos de personas trabajando en ellos. Necesitamos convertirnos en expertos en la gestión de la complejidad para hacer frente a esto, tanto a nivel técnico como organizacional.

Expertos en aprendizaje

La ciencia es la mejor técnica de resolución de problemas de la humanidad. Si queremos convertirnos en expertos en aprendizaje, debemos adoptar y volvernos expertos en el tipo de enfoque práctico basado en la ciencia para la resolución de problemas que es la esencia de otras disciplinas de ingeniería.

Debe adaptarse a nuestros problemas. La ingeniería de software será diferente de otras formas de ingeniería, específicas del software, de la misma manera que la ingeniería aeroespacial es diferente de la ingeniería química. Debe ser práctico, liviano y omnipresente en nuestro enfoque para resolver problemas de software.

Existe un consenso considerable entre las personas que muchos de nosotros consideramos líderes de opinión en nuestra industria sobre este tema. A pesar de ser bien conocidas, estas ideas no se practican universalmente ni siquiera ampliamente como base de cómo abordamos gran parte del desarrollo de software.

Hay cinco comportamientos vinculados en esta categoría:

- Trabajar de forma iterativa
- Emplear comentarios rápidos y de alta calidad
- Trabajando incrementalmente
- Ser experimental
- Ser empírico

Si no ha pensado en esto antes, estas cinco prácticas pueden parecer abstractas y bastante divorciadas de las actividades cotidianas del desarrollo de software, y mucho menos de la ingeniería de software.

El desarrollo de software es un ejercicio de exploración y descubrimiento. Siempre estamos tratando de aprender más sobre lo que nuestros clientes o usuarios quieren del sistema, cómo resolver mejor los problemas que se nos presentan y cómo aplicar mejor las herramientas y técnicas a nuestra disposición.

Aprendemos que nos hemos perdido algo y tenemos que arreglar las cosas. Aprendemos a organizarnos para trabajar mejor y aprendemos a comprender más profundamente los problemas en los que estamos trabajando.

El aprendizaje está en el centro de todo lo que hacemos. Estas prácticas son la base de cualquier enfoque eficaz para el desarrollo de software, pero también descartan algunos enfoques menos eficaces.

Los enfoques de desarrollo en cascada, por ejemplo, no presentan estas propiedades. Sin embargo, todos estos comportamientos están correlacionados con un alto rendimiento en los equipos de desarrollo de software y han sido el sello distintivo de los equipos exitosos durante décadas.

La Parte II explora cada una de estas ideas con más profundidad desde una perspectiva práctica: ¿Cómo nos convertimos en expertos en aprendizaje y cómo aplicamos eso a nuestro trabajo diario?

Expertos en gestionar la complejidad

Como desarrollador de software, veo el mundo a través de la lente del desarrollo de software. Como resultado, mi percepción de los fallos en el desarrollo de software y la cultura que los rodea puede considerarse en gran medida en términos de dos ideas de la ciencia de la información: concurrencia y acoplamiento.

Estos son difíciles en general, no sólo en el diseño de software. Entonces, estas ideas se escapan del diseño de nuestros sistemas y afectan la forma en que operan las organizaciones en las que trabajamos.

Esto se puede explicar con ideas como la ley de Conway,6 pero la ley de Conway es más bien una propiedad emergente de estas verdades más profundas.

Es rentable pensar en esto en términos más técnicos. Una organización humana es tanto un sistema de información como cualquier sistema informático. Es casi seguro que es más complejo, pero se aplican las mismas ideas fundamentales. Cosas que son fundamentalmente difíciles, como la concurrencia y el acoplamiento, también lo son en el mundo real de las personas.

Si queremos construir sistemas más complejos que el más simple de los ejercicios de programación de juguetes, debemos tomar estas ideas en serio. Necesitamos gestionar la complejidad de los sistemas que creamos a medida que los creamos, y si queremos hacerlo en cualquier tipo de escala más allá del alcance de un único y pequeño equipo, necesitamos gestionar la complejidad de los sistemas de información organizacionales. así como los sistemas de información software más técnicos.

Como industria, tengo la impresión de que prestamos muy poca atención a estas ideas, hasta el punto de que todos los que hemos dedicado algún tiempo al software estamos familiarizados con los resultados: sistemas como una gran bola de barro, fuera de control. Deuda técnica sin control, cantidad abrumadora de errores y organizaciones temerosas de realizar cambios en los sistemas que poseen.

---

6. En 1967, Mervin Conway observó que "Cualquier organización que diseñe un sistema (definido ampliamente) producirá un diseño cuya estructura es una copia de la estructura de comunicación de la organización". Ver https://bit.ly/3s2KZP2.

Percibo todo esto como un síntoma de equipos que han perdido el control de la complejidad de los sistemas en los que trabajan.

Si está trabajando en un sistema de software simple y desechable, entonces la calidad de su diseño importa poco. Si desea construir algo más complejo, debe dividir el problema para poder pensar en partes del mismo sin sentirse abrumado por la complejidad.

El lugar donde trazas esas líneas depende de muchas variables: la naturaleza del problema que estás resolviendo, las tecnologías que estás empleando y probablemente incluso qué tan inteligente eres, hasta cierto punto, pero debes trazar las líneas si quieres. para resolver problemas más difíciles.

Tan pronto como aceptas esta idea, estamos hablando de ideas que tienen un gran impacto en términos del diseño y la arquitectura de los sistemas que creamos. En el párrafo anterior fui un poco cauteloso a la hora de mencionar la "inteligencia" como parámetro, pero lo es. El problema que desconfiaba es que la mayoría de nosotros sobreestimamos nuestras capacidades para resolver un problema en código.

Esta es una de las muchas lecciones que podemos aprender de una visión informal de la ciencia. Es mejor empezar asumiendo que nuestras ideas son erróneas y trabajar según esa suposición. Por lo tanto, deberíamos ser mucho más cautelosos ante la potencial explosión de complejidad en los sistemas que creamos y trabajar para gestionarlos con diligencia y cuidado a medida que avanzamos.

También hay cinco ideas en esta categoría. Estas ideas están estrechamente relacionadas entre sí y vinculadas a las ideas involucradas en convertirse en expertos en el aprendizaje. Sin embargo, vale la pena pensar en estas cinco ideas si queremos gestionar la complejidad de forma estructurada para cualquier sistema de información:

- Modularidad
- Cohesión
- Separación de intereses
- Ocultación/abstracción de información
- Acoplamiento

Exploraremos cada una de estas ideas con mucha más profundidad en la Parte III.

Resumen

Las herramientas de nuestro oficio a menudo no son realmente lo que pensamos que son. Los lenguajes, herramientas y marcos que utilizamos cambian con el tiempo y de un proyecto a otro. Las ideas que facilitan nuestro aprendizaje y nos permiten abordar la complejidad de los sistemas que creamos son las verdaderas herramientas de nuestro oficio. Centrándonos en estas cosas, nos ayudará a elegir mejor los lenguajes, manejar las herramientas y aplicar los marcos de manera que nos ayuden a hacer un trabajo más eficaz a la hora de resolver problemas con el software.

Tener un “criterio” que nos permita evaluar estas cosas es una enorme ventaja si queremos tomar decisiones basadas en evidencia y datos, en lugar de modas o conjeturas. Al tomar una decisión, deberíamos preguntarnos: "¿Esto aumenta la calidad del software que creamos?" medido por las métricas de estabilidad. O "¿esto aumenta la eficiencia con la que creamos software de esa calidad" medida por el rendimiento? Si eso no empeora ninguna de estas cosas, podemos elegir lo que prefiramos; de lo contrario, ¿por qué elegiríamos hacer algo que empeore cualquiera de estas cosas?

Índice

A

B

C

D

mi

F

GH

I

JKL

METRO

norte

OP

q

R

S

t

ultravioleta

W X Y Z