Glosario

Este glosario recoge los términos técnicos, las técnicas y los conceptos del mundo del desarrollo que aparecen a lo largo del libro. Está pensado como consulta rápida: si un término no te resulta familiar, aquí encontrarás una definición breve. Las entradas están ordenadas alfabéticamente.

Análisis de causa raíz (root cause analysis). Buscar el origen real de un problema en lugar de quedarse en el síntoma. Su técnica más conocida son los *5 Whys* (cinco porqués): preguntar "por qué" de forma repetida (a menudo cinco veces) hasta dar con la causa sistémica que conviene corregir.

Andon. Sistema de señalización visual del Sistema de Producción Toyota que permite a cualquiera detener la línea al detectar un problema, dejando el fallo a la vista para resolverlo de inmediato. En software inspira la idea de "parar la línea" ante un defecto en lugar de dejarlo correr.

Apagar fuegos (firefighting). Modo reactivo en el que el equipo va resolviendo urgencias e incidentes sin tiempo para prevenir su causa, lo que perpetúa el problema.

Bancarrota técnica. Punto en el que la acumulación de deuda técnica y complejidad hace inviable seguir evolucionando un sistema sin reescribirlo: cada cambio cuesta tanto que el desarrollo se paraliza.

Bus factor. Número de personas que tendrían que faltar para que un proyecto se quedara bloqueado por falta de conocimiento. Un bus factor de uno (una sola persona sabe algo crítico) es un riesgo serio.

Cadena de valor (value stream). Secuencia completa de actividades desde que se entiende una necesidad del cliente hasta que se le entrega la solución. Mirarla entera ayuda a ver dónde se acumulan esperas y desperdicio.

Canary release (despliegue gradual). Estrategia de activar un cambio primero para un grupo pequeño de usuarios (los "canarios") y, si todo va bien, ampliarlo poco a poco al resto. Permite detectar problemas con impacto limitado.

Ciclo de feedback (feedback loop). Ciclo que devuelve información sobre el resultado de una acción. Cuanto más corto, antes se aprende y se corrige.

Cobertura de tests. Porcentaje del código que ejecutan las pruebas automáticas. Es una señal útil, no un objetivo en sí mismo: una cobertura alta no garantiza buenos tests.

Código legacy (legacy code). Código antiguo, a menudo sin tests y difícil de entender, que resulta costoso y arriesgado de modificar.

Coste basal (basal cost). Gasto permanente de capacidad del equipo por el simple hecho de que un sistema existe y hay que mantenerlo, con independencia de que se le añadan cambios o no. Concepto central del libro.

Dark launch. Desplegar una funcionalidad a producción manteniéndola invisible para el usuario, de modo que se pueda probar con tráfico real antes de activarla.

Despliegue continuo (Continuous Deployment). Llevar a producción de forma automática cada cambio que supera la validación, sin intervención manual. Va un paso más allá de la entrega continua.

Deuda técnica. Coste futuro que asumimos cuando, de forma consciente, tomamos un atajo en el diseño o la implementación para entregar valor antes. Como una deuda financiera, acumula "intereses": cada cambio posterior se vuelve más lento y arriesgado hasta que se "devuelve" (se refactoriza). No es lo mismo que el código descuidado o de baja calidad: la deuda es una decisión deliberada que se asume sabiendo que habrá que pagarla.

Diseño basado en conjuntos (set-based development). Práctica Lean de explorar varias soluciones posibles en paralelo e ir descartando las peores a medida que se aprende, en lugar de apostar pronto por una sola opción. Reduce el riesgo de comprometerse antes de tiempo con un diseño que luego resulte equivocado.

Eficiencia de flujo y eficiencia de recursos. Dos formas opuestas de medir. La eficiencia de recursos mira si las personas están siempre ocupadas; la eficiencia de flujo mira cuánto tiempo el trabajo avanza de verdad frente al que pasa esperando. Lean prioriza la segunda.

Ensemble programming. Práctica (antes llamada mob programming) en la que todo el equipo trabaja a la vez sobre el mismo problema y el mismo código. Es una extensión del pair programming a más personas.

Entrega continua (Continuous Delivery). Mantener el código siempre en un estado desplegable, de modo que cualquier cambio validado pueda llevarse a producción en cualquier momento (la decisión de hacerlo sigue siendo manual).

Extreme Programming (XP). Método de desarrollo Agile creado por Kent Beck, construido sobre un conjunto de valores (comunicación, simplicidad, feedback, coraje y respeto) que se concretan en principios y, finalmente, en prácticas. Combina prácticas técnicas (TDD, integración continua, pair programming, diseño simple) con prácticas de equipo y planificación: ahí reside su fuerza, frente a enfoques centrados solo en la gestión.

Feature flag. Interruptor en el código (también llamado feature toggle) que permite activar o desactivar una funcionalidad sin desplegar de nuevo. Separa el momento de desplegar del de hacer visible un cambio.

Flaky test. Test que unas veces pasa y otras falla sin que el código haya cambiado. Erosiona la confianza en la batería de pruebas y conviene arreglarlo o eliminarlo.

Gestión de incidentes sin culpables (blameless). Forma de analizar los fallos de producción centrándose en qué parte del sistema o del proceso permitió el error, no en señalar personas. Favorece el aprendizaje y la seguridad psicológica.

Handoff (traspaso). Momento en que el trabajo pasa de una persona o equipo a otro. Cada traspaso añade esperas y pérdida de contexto, por eso Lean busca reducirlos.

Integración continua (CI). Práctica en la que todo el equipo integra su trabajo en la rama principal compartida con mucha frecuencia (al menos una vez al día). Como exige integrar el código de todos a diario, es incompatible con trabajar en ramas que vivan más de un día. Para que sea viable, cada integración se valida de forma automática (build y tests), de modo que los problemas se detectan cuanto antes.

Jidoka. Principio del Sistema de Producción Toyota: dotar a máquinas y procesos de "un toque humano" para que detecten un defecto y detengan el flujo, evitando que el error se propague.

Just-in-Time (JIT). Principio Lean de producir solo lo necesario, en la cantidad necesaria y en el momento necesario, en lugar de acumular trabajo o inventario por adelantado.

Kaikaku. Mejora radical y disruptiva, normalmente impulsada desde la dirección, frente al cambio gradual del Kaizen.

Kaizen. Filosofía de mejora continua mediante muchos cambios pequeños y sostenidos en el tiempo. Significa "cambio para mejorar".

Kanban. Método para gestionar y mejorar el flujo de trabajo de forma evolutiva: hacer visible lo que está en curso, limitar la cantidad de trabajo simultáneo (WIP) y gestionar el flujo para detectar y resolver cuellos de botella. Toma su nombre de las "tarjetas" o señales visuales del Sistema de Producción Toyota.

KISS (Keep It Simple, Stupid). Principio que recuerda mantener las soluciones lo más simples posible y evitar la complejidad innecesaria.

Lean Manufacturing y Sistema de Producción Toyota (TPS). Sistema de producción nacido en Toyota que maximiza el valor para el cliente eliminando desperdicio, con principios como Jidoka, Just-in-Time y Kaizen. Es el origen del pensamiento Lean.

Lean Software Development. Adaptación al desarrollo de software de los principios Lean: eliminar desperdicio, amplificar el aprendizaje, entregar rápido y optimizar el conjunto. Marco central del libro.

Ley de Conway (Conway's Law). Observación de que la arquitectura de un sistema acaba reflejando la estructura de comunicación de la organización que lo construye. De ahí que diseñar cómo se organizan los equipos sea, en la práctica, diseñar también el software.

Linting. Análisis automático del código que detecta errores y desviaciones de estilo antes de integrarlo.

Manifiesto por el Desarrollo Ágil de Software (Manifiesto Ágil). Documento que recoge los cuatro valores y doce principios del desarrollo Agile. Sus valores priorizan los individuos e interacciones, el software funcionando, la colaboración con el cliente y la respuesta al cambio.

Método de la hamburguesa. Técnica de Gojko Adzic para descomponer una funcionalidad en sus "capas" y generar varias opciones por capa, eligiendo la combinación más sencilla que aporta valor. Ayuda a partir el trabajo en incrementos pequeños.

Métricas DORA (Accelerate). Cuatro métricas, popularizadas por los informes DORA y el libro *Accelerate*, que correlacionan el rendimiento de entrega de un equipo con sus resultados de negocio: frecuencia de despliegue, Lead Time de los cambios, tiempo de recuperación ante fallos (MTTR) y porcentaje de cambios fallidos.

MTTR (tiempo medio de recuperación). Tiempo medio que tarda un sistema en recuperarse tras un fallo. En entornos resilientes importa más bajar el MTTR que pretender evitar todo fallo a cualquier precio.

Muda, Muri y Mura. Los tres tipos de desperdicio en Lean: Muda (actividad que no aporta valor), Muri (sobrecarga de personas o procesos) y Mura (irregularidad o falta de uniformidad en el trabajo).

Mutation testing. Técnica que introduce pequeños cambios ("mutaciones") en el código para comprobar si los tests los detectan. Mide la calidad real de las pruebas, no solo su cantidad.

MVP (producto mínimo viable). Versión más simple de un producto que permite validar una hipótesis con usuarios reales y aprender con el mínimo esfuerzo.

Observabilidad. Capacidad de entender qué pasa dentro de un sistema y diagnosticar problemas que no habías previsto, sin tener que tocarlo ni desplegar código nuevo solo para poder hacerle una pregunta nueva.

Output y outcome. Output es lo que se entrega (funcionalidades, entregables); outcome es el impacto real que eso produce en usuarios o negocio. Lo que importa es el outcome, no el volumen de output.

Pair programming. Dos personas programando juntas sobre el mismo código: una escribe y la otra revisa y piensa en voz alta, intercambiando roles. Mejora la calidad y comparte conocimiento.

Poka-Yoke. Diseño a prueba de errores: mecanismos que hacen difícil o imposible cometer un fallo, en lugar de confiar en que la persona no se equivoque.

Pre-commit hooks. Validaciones automáticas (tests rápidos, linting) que se ejecutan en local antes de confirmar un cambio, para frenar errores cuanto antes.

Product Engineer. Ingeniero con mentalidad de producto: entiende el problema de negocio y el contexto del usuario, y participa en decidir qué construir, no solo en ejecutar tareas.

Product-market fit. Situación en la que un producto resuelve de verdad un problema que el mercado tiene y valora, hasta el punto de adoptarlo y pagarlo.

Radio de impacto (blast radius). Alcance del daño que provoca un fallo o un cambio: cuántos usuarios, servicios o datos se ven afectados. Reducirlo (por ejemplo con despliegues graduales) limita las consecuencias de un error.

Refactoring (refactorización). Mejorar la estructura interna del código sin cambiar su comportamiento externo, para que sea más claro y fácil de mantener.

Rollback automático. Mecanismo que revierte un despliegue a la versión anterior de forma automática cuando se detecta un problema tras publicar.

Scrum. Marco de trabajo Agile que organiza el desarrollo en iteraciones de duración fija (sprints) con roles y ceremonias definidos.

Segmentación técnica (technical slicing). Partir el trabajo por componentes o capas técnicas (base de datos, backend, frontend) en lugar de por valor entregable. Suele retrasar la entrega de algo útil, al contrario que la segmentación vertical.

Segmentación vertical (vertical slicing). Partir una funcionalidad en incrementos que atraviesan todas las capas y aportan valor por sí mismos, en lugar de cortarla por capas técnicas. Permite entregar y aprender antes.

Slack (holgura). Capacidad que se deja deliberadamente sin asignar para absorber imprevistos, aprender y mejorar. Sin holgura, cualquier incidente desborda al equipo.

Smoke test. Prueba rápida que se lanza tras un despliegue para comprobar que las funciones esenciales del sistema siguen operativas.

Spike. Investigación técnica acotada en tiempo cuyo objetivo es aprender o reducir incertidumbre (probar una idea, validar una tecnología), no producir código final.

Tablero Kanban (Kanban board). Herramienta visual que representa el trabajo en columnas correspondientes a las etapas del proceso (por ejemplo "pendiente", "en curso", "hecho"). Las tareas avanzan de columna en columna, lo que permite ver de un vistazo el estado del flujo y dónde se acumula el trabajo.

TDD (Test-Driven Development). Disciplina de escribir primero un test que falla, luego el código mínimo que lo hace pasar y después refactorizar. En su variante Outside-In TDD se empieza por un test de un caso de uso completo y se trabaja hacia dentro.

Team Topologies. Modelo de Matthew Skelton y Manuel Pais para organizar los equipos de forma que optimicen el flujo de valor, limitando la carga cognitiva de cada uno. Define cuatro tipos de equipo (alineado al flujo, de plataforma, facilitador y de subsistema complicado) y tres modos de interacción entre ellos (colaboración, X como servicio y facilitación).

Técnica Pomodoro. Método de gestión del tiempo basado en bloques de trabajo concentrado (habitualmente de 25 minutos) separados por descansos breves. En equipo, sincronizar esos bloques protege el tiempo de foco.

Throughput. Cantidad de valor o incrementos que un equipo entrega por unidad de tiempo. Junto al Lead Time, describe el rendimiento del flujo.

Tiempo de ciclo (Cycle Time). Tiempo desde que se empieza a trabajar en un incremento hasta que está desplegado y en uso. Métrica clave de la rapidez del flujo.

Tiempo de entrega (Lead Time). Tiempo total desde que se identifica una necesidad hasta que la solución llega al usuario. Mide la rapidez de extremo a extremo.

Toil. Trabajo operativo manual, repetitivo y sin valor duradero que crece con el sistema y consume capacidad del equipo. Conviene automatizarlo o eliminarlo.

Trunk Based Development. Estrategia en la que todo el equipo trabaja sobre una única rama principal (el "trunk"), integrando en ella cambios pequeños varias veces al día. Evita el uso de ramas separadas que divergen del tronco y dificultan la integración.

Último momento responsable (last responsible moment). Principio Lean de posponer una decisión hasta el punto en que retrasarla más eliminaría alguna opción importante. Decidir entonces, con la máxima información disponible, evita comprometerse antes de tiempo sin caer por ello en la parálisis.

WIP y límite de WIP. WIP (Work In Progress) es el trabajo empezado pero todavía no terminado ni entregado. Poner un límite de WIP (un máximo de tareas en curso) mejora el flujo y reduce el tiempo de entrega.

YAGNI (You Aren't Gonna Need It). Principio que aconseja no construir algo hasta que se necesite de verdad, evitando funcionalidad especulativa que infla el coste basal.