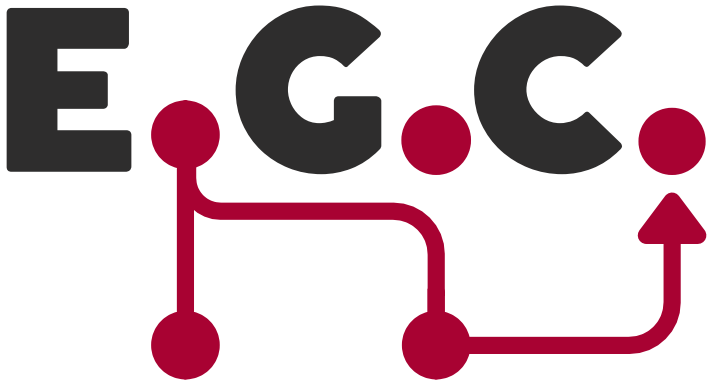


Grado en Ingeniería Informática – Ingeniería del Software

Evolución y Gestión de la Configuración

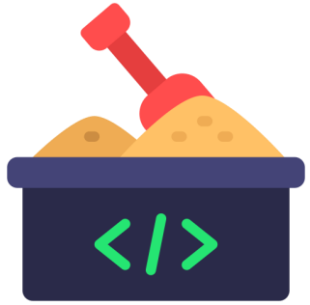
Escuela Técnica Superior de Ingeniería Informática

Gestión de la construcción e integración continua.

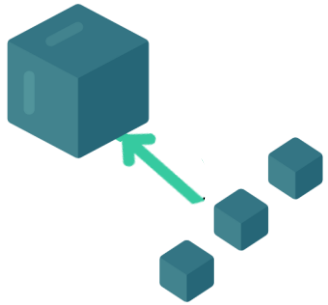
1. **Introducción**
2. Conceptos básicos
3. Integración continua
4. Resumen
5. Bibliografía

1. Introducción | ejemplo de otro dominio

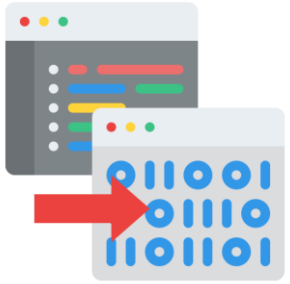
1. Introducción | construcción del software



Preparar entorno de desarrollo



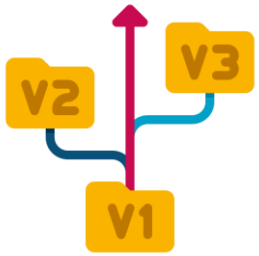
Gestión de dependencias



Compilación

Testing

Code review



Versionado

Empaquetado

Entrega

Despliegue

1. Introducción
2. Conceptos básicos
3. Integración continua
4. Resumen
5. Bibliografía

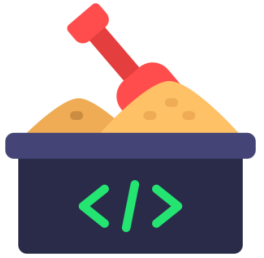
2. Conceptos básicos| construcción *vs* gestión

Construir software es el proceso técnico de convertir el código fuente y otros artefactos de desarrollo en un producto funcional listo para su uso. Incluye actividades como la compilación del código, la ejecución de pruebas automáticas, el empaquetado de archivos y la preparación para su despliegue.

Lo que no puedo automatizar, lo tengo que gestionar

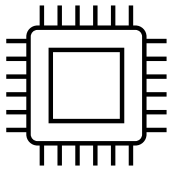
La **gestión de la construcción** de software es el proceso de organizar, automatizar y supervisar todas las actividades relacionadas con la preparación del software para su uso, asegurando que se sigan los procedimientos adecuados y se tomen las decisiones correctas a lo largo del ciclo de vida del software.

2. Conceptos básicos| configuraciones locales

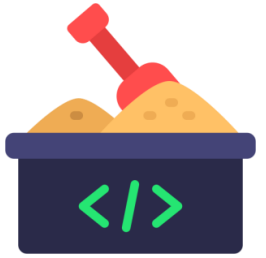


- **Automatiza la configuración:** se pueden establecer variables de entorno necesarias para la ejecución de pruebas y la construcción del software.

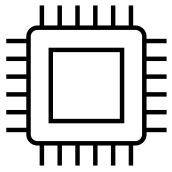
- **Provisiona recursos temporales:** automatizar la creación de entornos temporales o contenedores necesarios para ejecutar las pruebas de manera aislada y consistente.
- **Instalar IDEs**. Hay una tendencia para facilitar el *onboarding* que permite tener todo el entorno de desarrollo instalado rápidamente: *devcontainers*



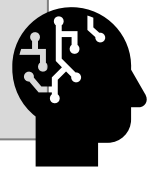
2. Conceptos básicos| configuraciones locales



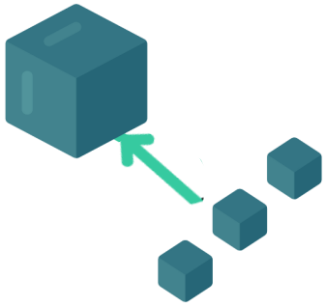
- **Automatiza la configuración:** se pueden establecer variables de entorno necesarias para la ejecución de pruebas y la construcción del software.
- **Provisiona recursos temporales:** automatizar la creación de entornos temporales o contenedores necesarios para ejecutar las pruebas de manera aislada y consistente.
- **Instalar IDEs**. Hay una tendencia para facilitar el *onboarding* que permite tener todo el entorno de desarrollo instalado rápidamente: *devcontainers*



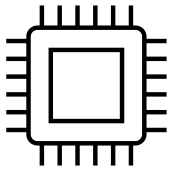
- **Instalación de herramientas de desarrollo:** instalar y configurar herramientas de construcción y desarrollo local, como compiladores o IDEs específicos.
- **Ajustes personalizados:** gestionar configuraciones personalizadas del entorno local, como ajustes de editor de código, preferencias de compilación, y extensiones necesarias para el flujo de trabajo diario.
- **Configuración de entornos locales específicos:** configurar el entorno de desarrollo para adaptarse a necesidades particulares del proyecto, como perfiles de compilación específicos o herramientas locales.



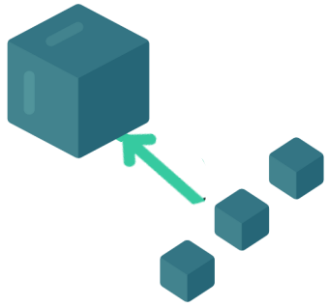
2. Conceptos básicos| gestión de dependencias



- **Instalación de dependencias:** ejecutar comandos para instalar las dependencias necesarias usando herramientas como `npm install` o `pip install`.
- **Automatización de instalación:** automatizar el proceso de instalación de dependencias como parte del flujo de integración continua.
- **Verificación de versiones:** instalar versiones específicas de dependencias definidas en los archivos de configuración del proyecto, como package.json o requirements.txt.

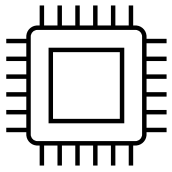


2. Conceptos básicos| gestión de dependencias



- **Instalación de dependencias:** ejecutar comandos para instalar las dependencias necesarias usando herramientas como npm install o pip install.
- **Automatización de instalación:** automatizar el proceso de instalación de dependencias como parte del flujo de integración continua.
- **Verificación de versiones:** instalar versiones específicas de dependencias definidas en los archivos de configuración del proyecto, como package.json o requirements.txt.

- **Selección de dependencias:** decidir qué dependencias deben ser utilizadas o cuáles deben ser añadidas o eliminadas del proyecto.
- **Actualización de versiones:** actualizar automáticamente las dependencias a sus últimas versiones.
- **Resolución de conflictos:** manejar conflictos de versiones entre diferentes dependencias. Por ejemplo, cuando dos librerías requieren versiones diferentes de una misma dependencia.



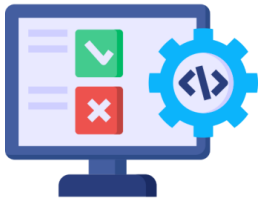
2. Conceptos básicos| compilación

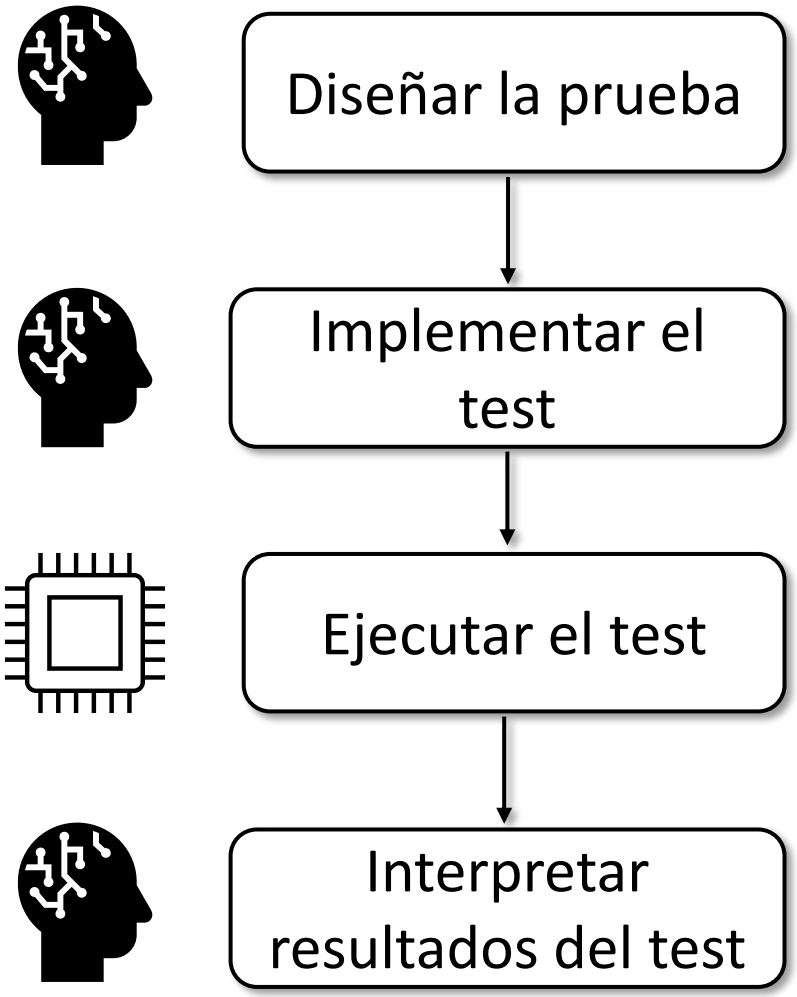
- **Automatiza el proceso de compilación**: ejecutar comandos predefinidos para compilar el código automáticamente, (mvn compile, npm run build, etc.).
- **Detecta errores de compilación**: detectar automáticamente errores de compilación en el código y detener el proceso.
- **Compilación multiplataforma**: por ejemplo, sistemas operativos o arquitecturas de hardware.
- **Minimiza y *transpila* código**: en proyectos front-end, se puede automatizar la transpilación de código a versiones más antiguas (por ejemplo, ES6 a ES5 con Babel) y la minificación de archivos CSS y JavaScript.

2. Conceptos básicos| compilación

- **Optimizar el código** : tomar decisiones sobre cómo optimizar el código para mejorar rendimiento, tamaño o compatibilidad.
- **Seleccionar los parámetros de compilación óptimos**: los parámetros y configuraciones de compilación (como los *flags* en un compilador) deben ser definidos por los desarrolladores.
- **Cumplir con requisitos de compilación**: si el código necesita cambios para poder compilarse correctamente en un nuevo entorno o bajo nuevas restricciones, esos ajustes deben hacerse manualmente.

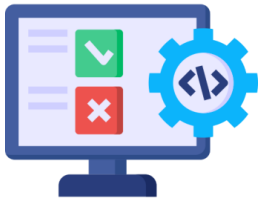
2. Conceptos básicos| testing

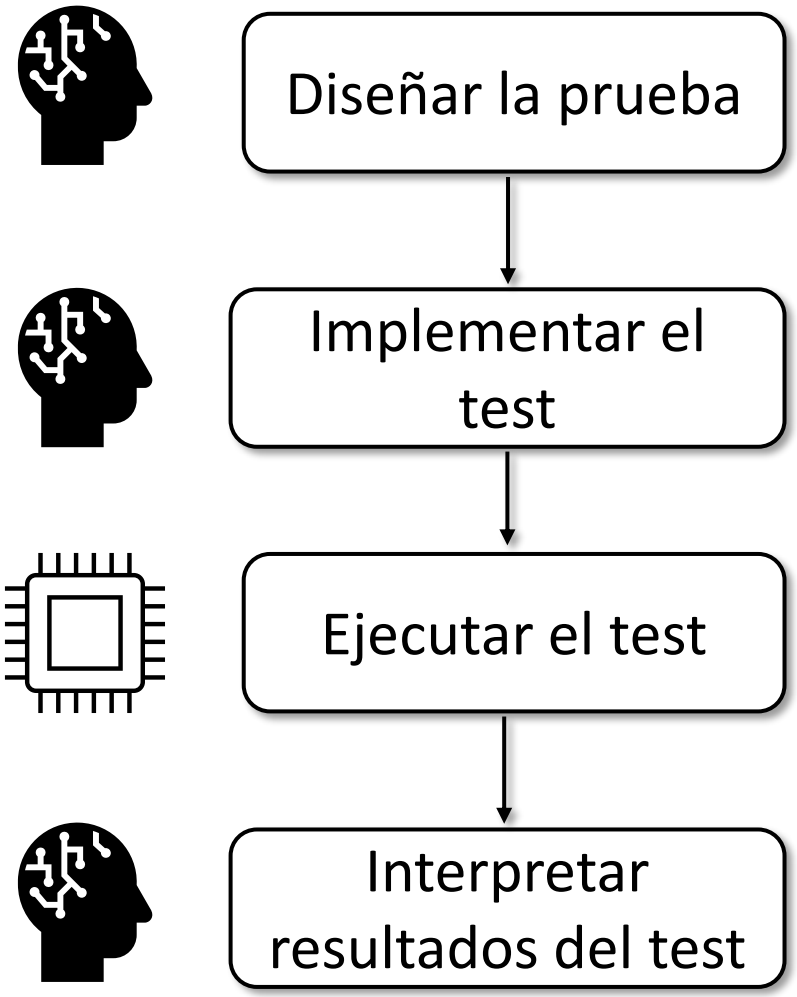




- Definir qué aspectos del software deben ser probados, cómo se estructurarán las pruebas y qué criterios se usarán para evaluar los resultados. Requiere conocimiento sobre el software, los objetivos de la prueba y la identificación de casos de prueba adecuados.

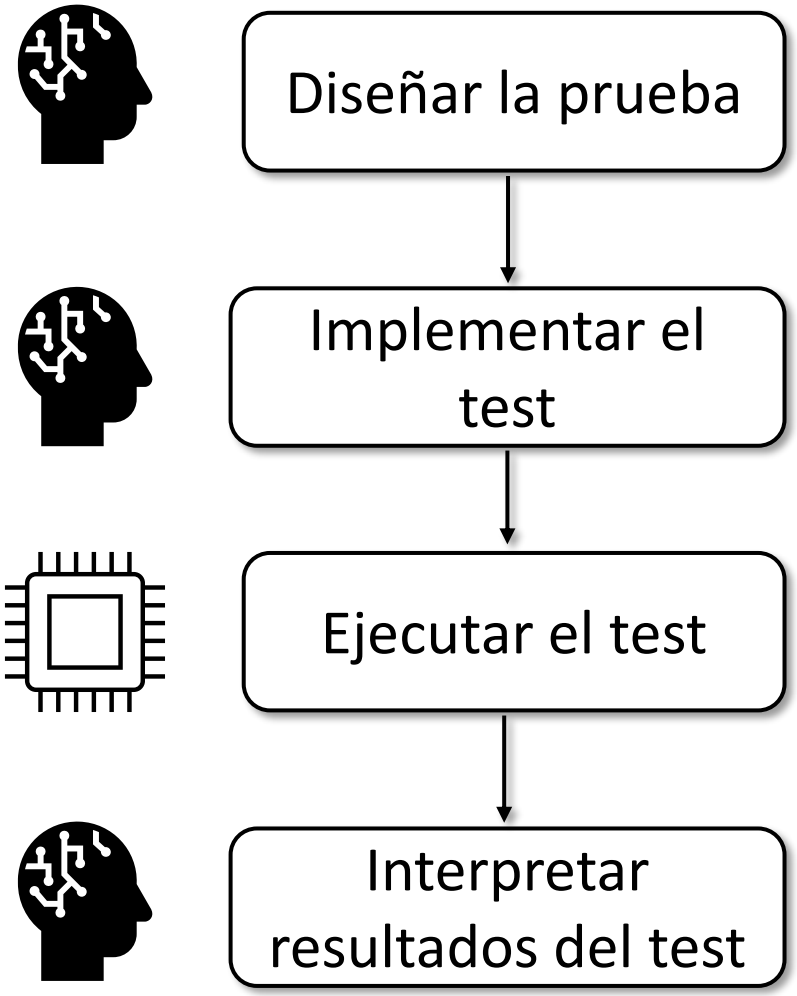
2. Conceptos básicos| testing





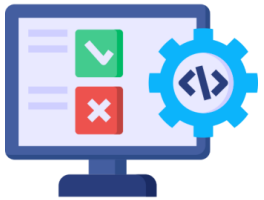
- Definir qué aspectos del software deben ser probados, cómo se estructurarán las pruebas y qué criterios se usarán para evaluar los resultados. Requiere conocimiento sobre el software, los objetivos de la prueba y la identificación de casos de prueba adecuados.
- Escribir el código para las pruebas, establecer las condiciones de prueba y preparar los datos necesarios. Implica decisiones sobre la estructura de las pruebas y la lógica para implementar diferentes escenarios de prueba.

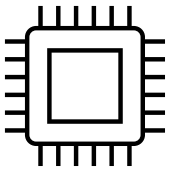
2. Conceptos básicos| testing



- Definir qué aspectos del software deben ser probados, cómo se estructurarán las pruebas y qué criterios se usarán para evaluar los resultados. Requiere conocimiento sobre el software, los objetivos de la prueba y la identificación de casos de prueba adecuados.
- Escribir el código para las pruebas, establecer las condiciones de prueba y preparar los datos necesarios. Implica decisiones sobre la estructura de las pruebas y la lógica para implementar diferentes escenarios de prueba.
- Analizar los resultados de las pruebas, identificar fallos o problemas y determinar su impacto en el software. Necesita habilidades analíticas para entender los resultados y tomar decisiones sobre la corrección de errores y mejoras.

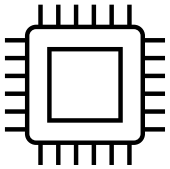
2. Conceptos básicos| testing





La ejecución de pruebas puede ser automatizada, pero el diseño, la implementación y la interpretación requieren intervención humana para asegurar pruebas efectivas y análisis precisos.

2. Conceptos básicos| code review



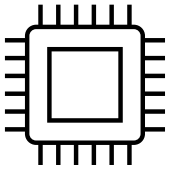
Análisis Estático de Código:

- Detección temprana de errores
- Consistencia del código
- Automatización y escalabilidad

Inspección de Código:

- Calidad del código
- Mentoría y mejora continua
- Detección de problemas complejos

2. Conceptos básicos| code review

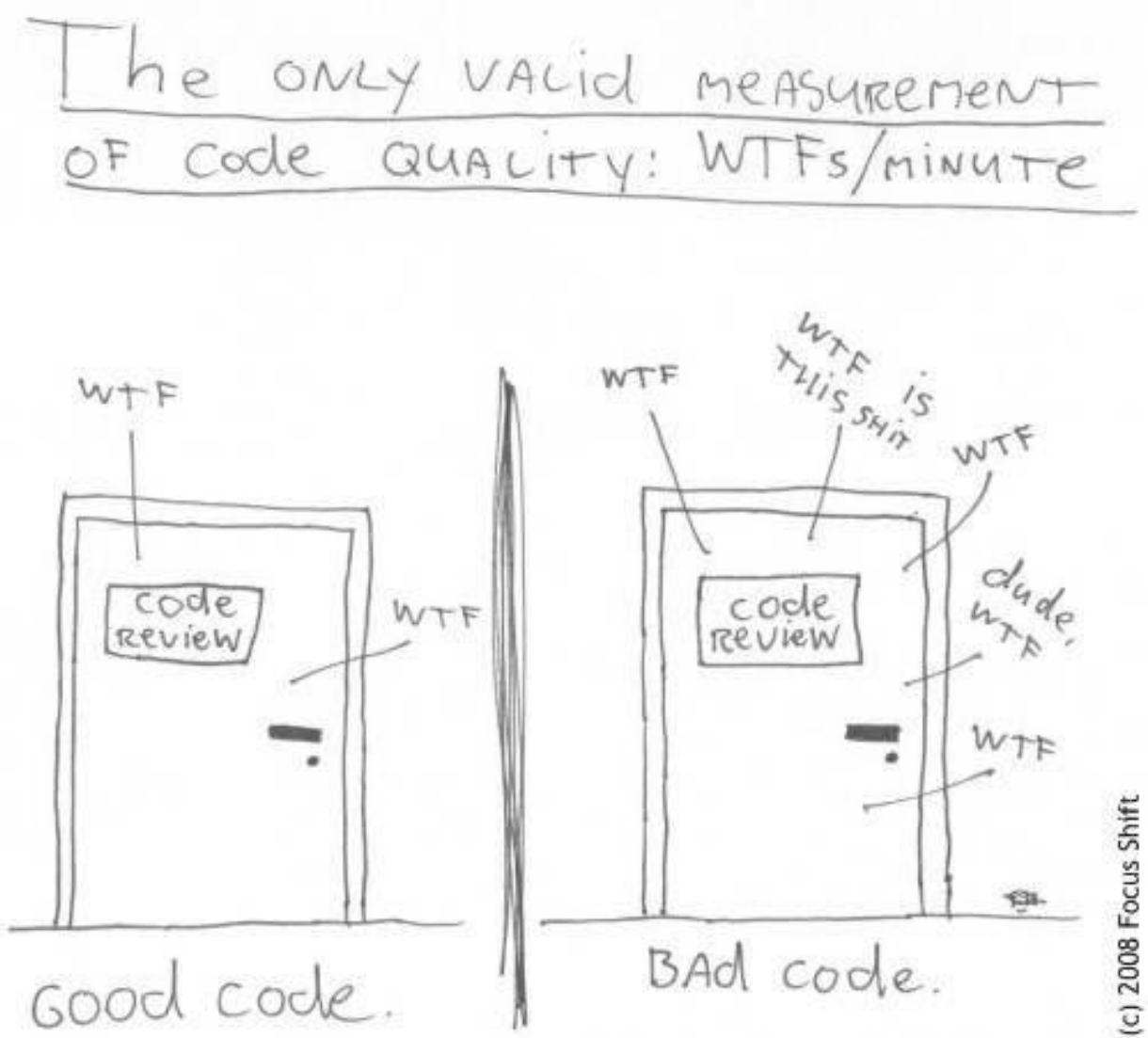


Análisis Estático de Código:

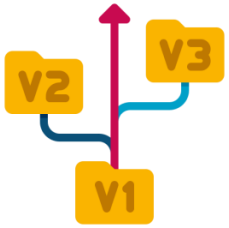
- Detección temprana de errores
- Consistencia del código
- Automatización y escalabilidad

Inspección de Código:

- Calidad del código
- Mentoría y mejora continua
- Detección de problemas complejos



2. Conceptos básicos | versionado



- Basado en secuencias:
  - Importancia del cambio:
    - X.Y.Z
      - X: cambios sustanciales en funcionalidad
      - Y: cambios menores en funcionalidad
      - Z: cambios menores, no hay cambios de funcionalidad
    - Estado de la versión:
      - Alpha → primera liberación: alpha, alpha1, alpha2
      - Beta → fase inicial: beta, beta1, beta2
      - Release candidate → candidata a versión final: rc, rc1, rc2
      - Final release → liberación final
  - Fecha de liberación:
    - Ubuntu 24.04.1, 22.04.5, 20.04.6, 23.10
    - Wine 20040505
- Estrategia de versionado:
  - Manual
  - Automático → automating semantic versioning : https://semver.org/lang/es/

2. Conceptos básicos| empaquetado

- **Generación de Artefactos:**
  - Crear artefactos (archivos JAR, ZIP, o contenedores Docker) automáticamente a partir del código fuente y otros recursos.
  - Aplicar configuraciones predefinidas para el empaquetado, como incluir archivos específicos o excluir otros.
- **Integración de Recursos:**
  - Incluir automáticamente las dependencias necesarias en el artefacto empaquetado.
  - Automatizar la preparación del artefacto para su distribución, asegurando que esté listo para el despliegue o distribución.

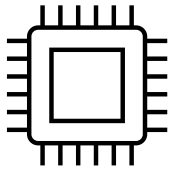
2. Conceptos básicos| empaquetado

- **Definición de Estrategias de Empaquetado:**
  - Decidir el formato de empaquetado más adecuado para el software o el entorno de despliegue.
  - Establecer políticas sobre cómo y cuándo empaquetar el software o las versiones de empaquetado.
- **Configuraciones Personalizadas:**
  - Realizar ajustes personalizados para empaquetado en contextos específicos o para requisitos especiales del proyecto (por ejemplo, optimización para plataformas específicas).
  - Solucionar problemas específicos que puedan surgir durante el proceso de empaquetado, como conflictos entre recursos o dependencias.

2. Conceptos básicos| entrega

Automatizado	No Automatizado
🟢 Automatiza el despliegue en entornos	❌ No decide cuándo es el momento adecuado
🟢 Genera versiones de entrega	❌ No realiza validaciones manuales
🟢 Sube artefactos a repositorios	❌ No maneja entregas parciales o rollbacks complejos

2. Conceptos básicos| despliegue



- Desplegar a entornos preconfigurados
- Ejecutar scripts post-despliegue
- Monitorizar del éxito del despliegue

- Configuraciones específicas del cliente
- Resolución de problemas inesperados en el servidor
- Decisión de retrocesos (*rollbacks*) tras errores críticos

2. Conceptos básicos| despliegue

- Desplegar a entornos preconfigurados
- Ejecutar scripts post-despliegue
- Monitorizar del éxito del despliegue

- Configuraciones específicas del cliente
- Resolución de problemas inesperados en el servidor
- Decisión de retrocesos (rollbacks) tras errores críticos

2. Conceptos básicos| construcción vs gestión de la construcción

Construir software es el proceso técnico de convertir el código fuente y otros artefactos de desarrollo en un producto funcional listo para su uso. Incluye actividades como la compilación del código, la ejecución de pruebas automáticas, el empaquetado de archivos y la preparación para su despliegue.

La **gestión de la construcción** de software es el proceso de organizar, automatizar y supervisar todas las actividades relacionadas con la preparación del software para su uso, asegurando que se sigan los procedimientos adecuados y se tomen las decisiones correctas a lo largo del ciclo de vida del software.

Lo que no puedo automatizar, lo tengo que gestionar:

La automatización **no** reemplaza la inteligencia, la planificación y la toma de decisiones.

2. Conceptos básicos| integración

Integración: combinar dos o más unidades de software. A menudo un subconjunto del total del proyecto.

¿Por qué hay que preocuparse de la integración?

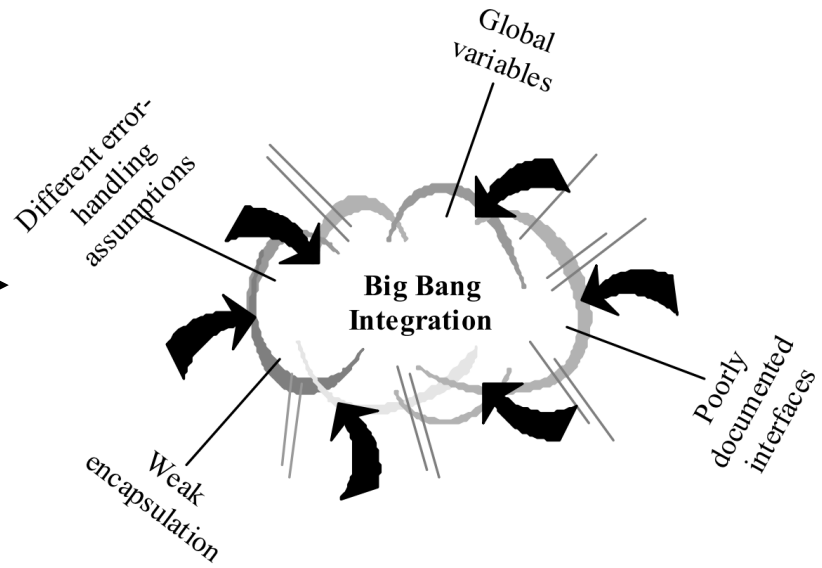
"If it hurts, do it more frequently, and bring the pain forward."

Jez Humble.

2. Conceptos básicos| integración por fases

Diseñar, codificar, probar y depurar cada clase/unidad/subsistema de manera separada.

Lo combinamos todo

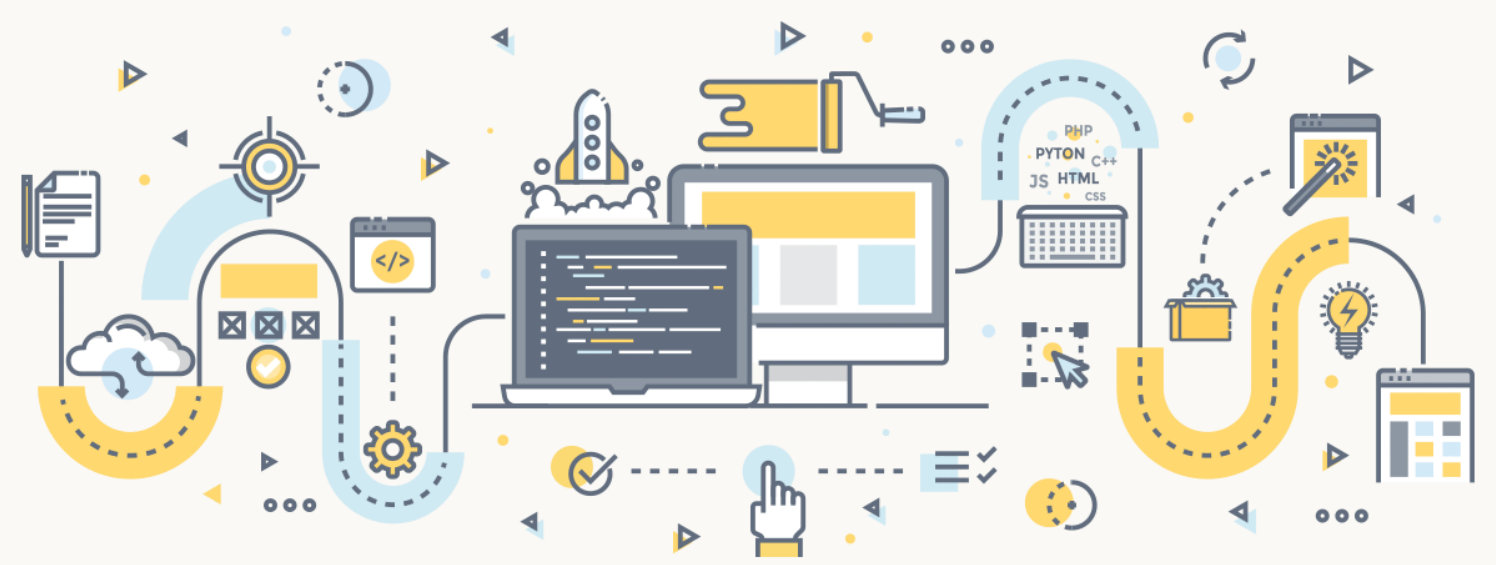


1. Introducción
2. Conceptos básicos
3. Integración continua
    - Definición
    - Arquitectura
    - Builds
    - Pruebas y CI
    - Inspección y análisis de código
    - Feedback
    - Lo esencial
4. Resumen
5. Bibliografía

3. Integración continua| definición

- Concepto popularizado por Martin Fowler (2010). La CI parte del XP y los métodos ágiles.

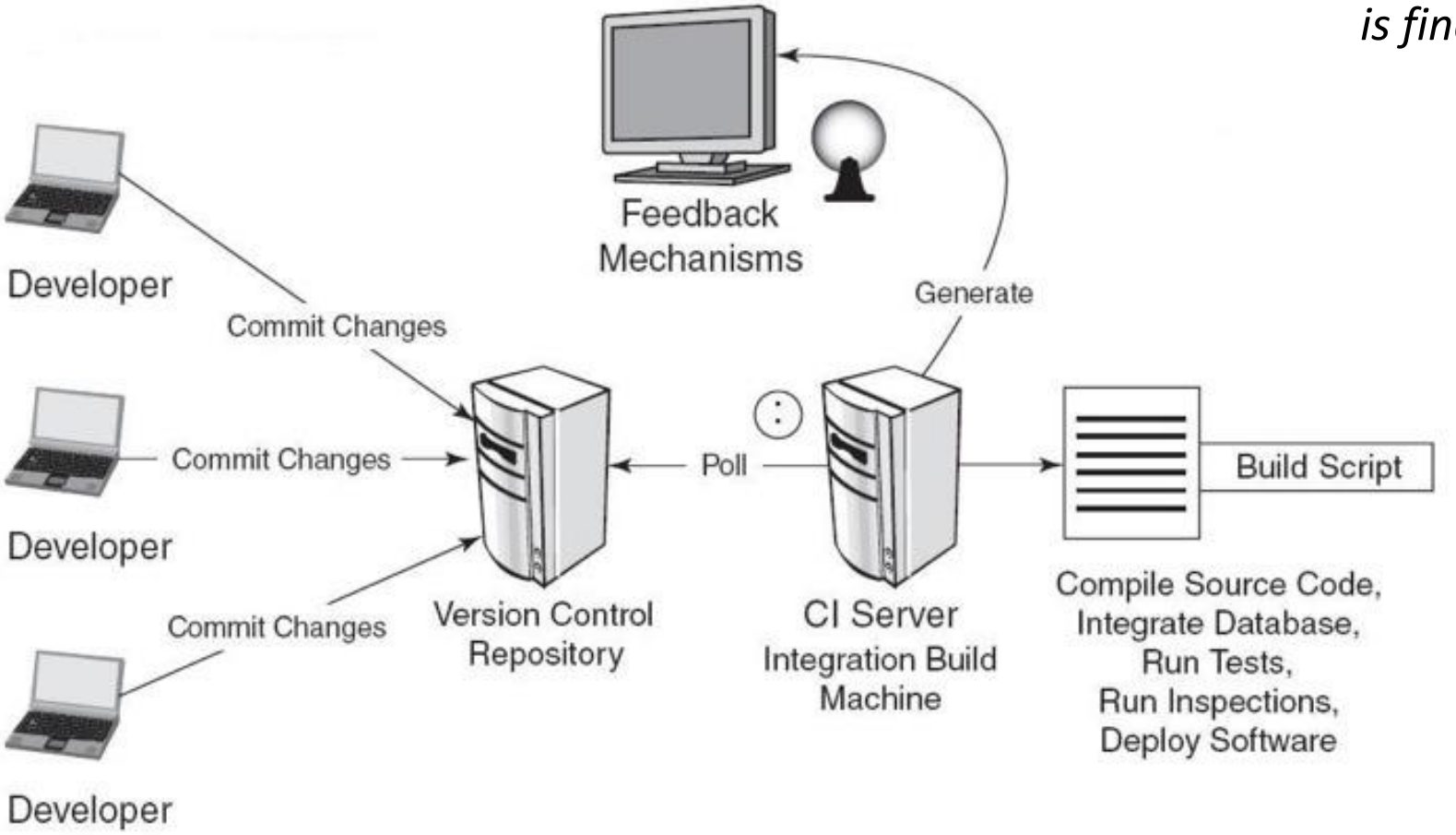
La integración continua (CI) es la práctica de construir y probar de manera regular, exhaustiva y automática las aplicaciones en el desarrollo de software.



PIPELINE

3. Integración continua| arquitectura

"The key to fixing problems quickly is finding them quickly."
Fowler.



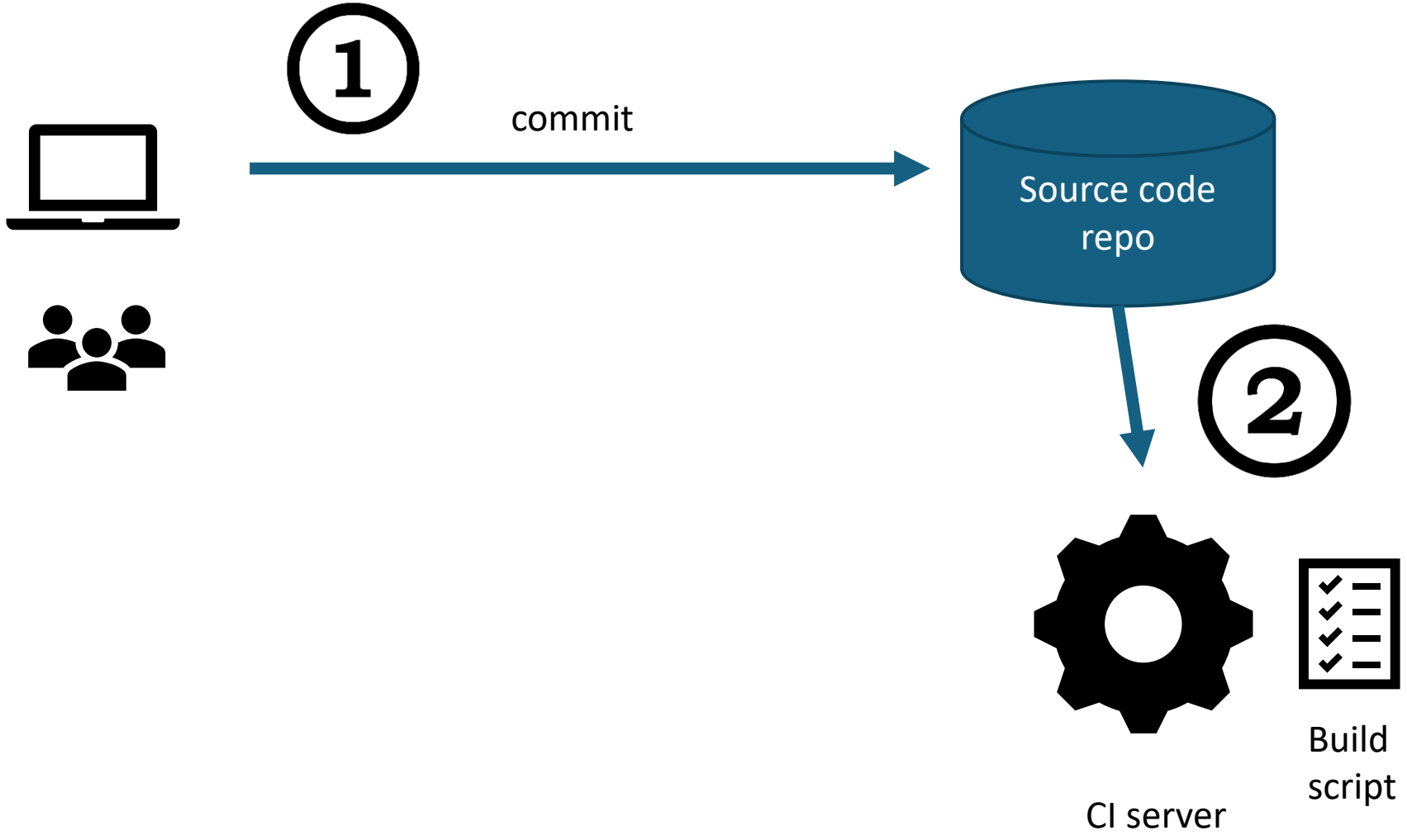
3. Integración continua| definición

- Concepto popularizado por Martin Fowler. La CI parte del XP y los métodos ágiles.

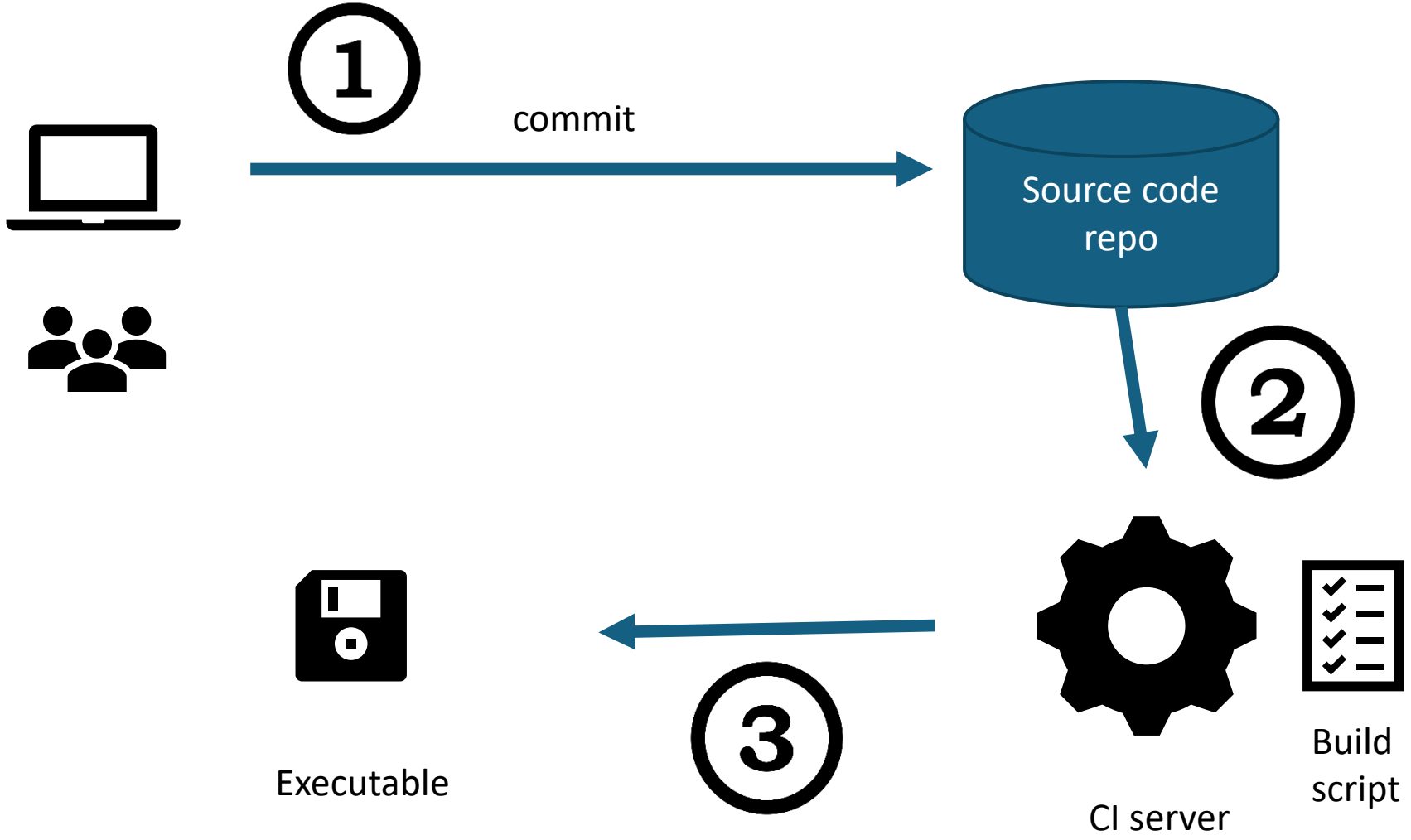
> La integración continua (CI) es la práctica de construir y probar de manera regular, exhaustiva y automática las aplicaciones en el desarrollo de software.

- Se basa en 10 principios:
  1. Mantener un único repositorio de código
  2. Automatizar los *build*
  3. Hacer que los *build* sean auto-testeables
  4. Todo el mundo hace *commit* a la línea principal *una vez al día*
  5. Todos los *commits* deben lanzar un trabajo de integración en una máquina de integración
  6. Hacer los *build* rápidos y cortos
  7. Hacer las pruebas en un clone del entorno de producción
  8. Hacer que sea fácil para todo el mundo acceder al último ejecutable
  9. Todo el mundo puede ver lo que está pasando
  10. Automatizar los despliegues

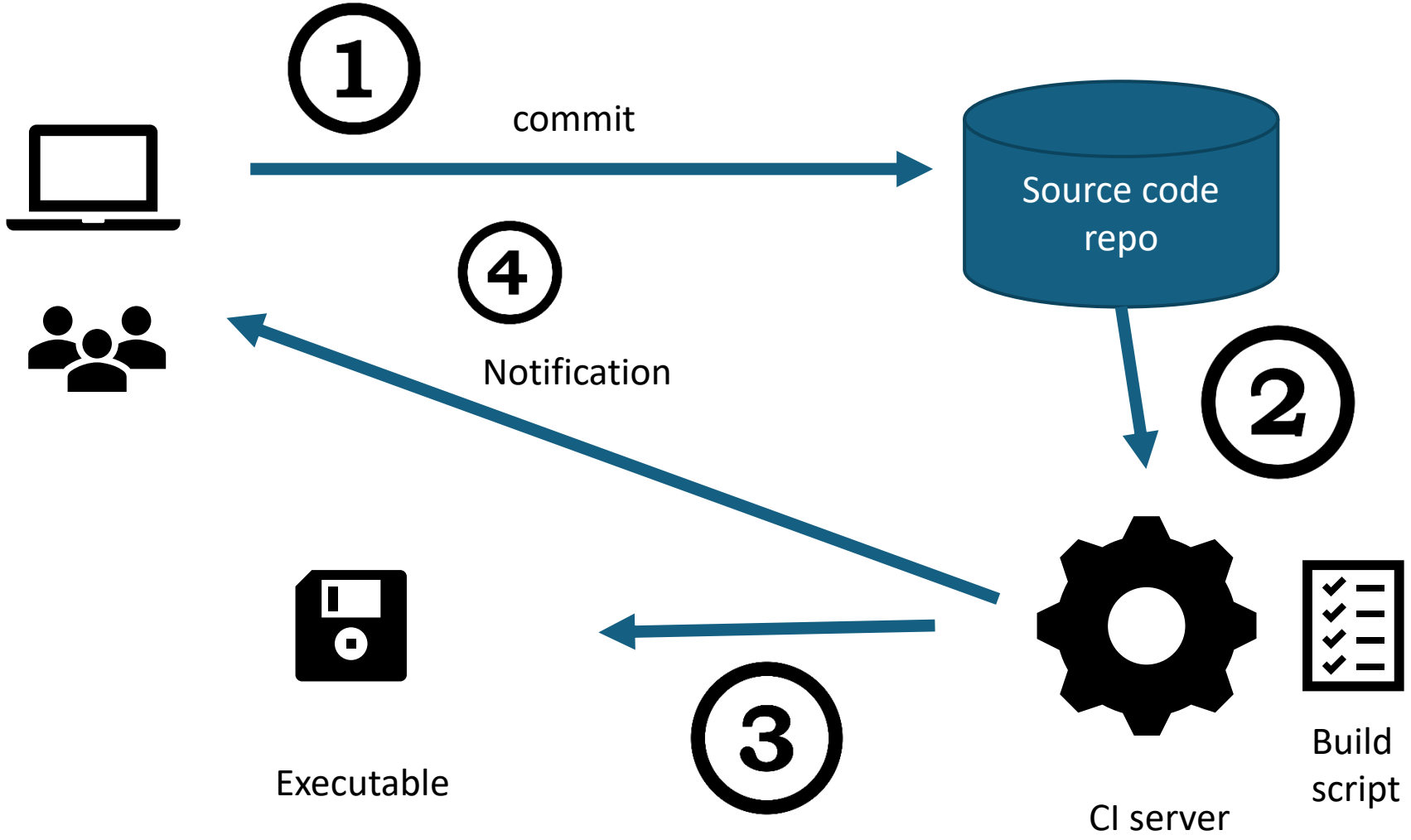
3. Integración continua| ciclo de CI



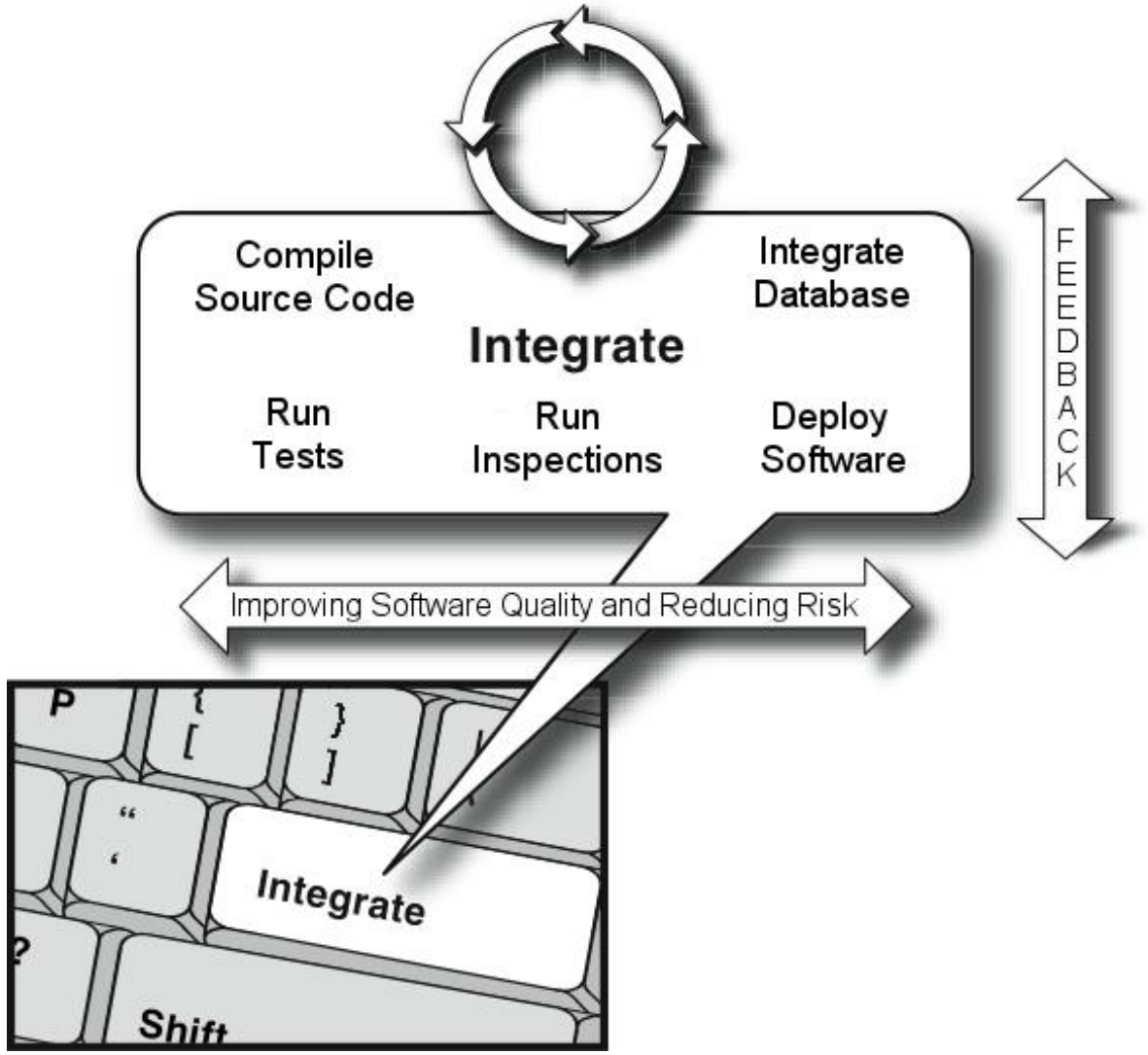
3. Integración continua| ciclo de CI



3. Integración continua| ciclo de CI



3. Integración continua| la metáfora del botón de integración



1. Introducción
2. Conceptos básicos
3. Integración continua
    - Definición
    - Arquitectura
    - Builds
    - Pruebas y CI
    - Inspección y análisis de código
    - Feedback
    - Lo esencial
4. Resumen
5. Bibliografía

3. Integración continua| builds

- Normalmente un *build* necesita algo más que código fuente:
  - Elementos de terceros o de otras partes del proyecto
  - Ficheros de configuración
  - Ficheros de datos o scripts de datos
  - Scripts de tests
  - Scripts de construcción

Todos los artefactos, excepto de los de terceros, deben estar almacenados en el repositorio de código.

¿Qué hacemos con los binarios?

3. Integración continua| daily commits

Todos hacemos **commit** a la línea principal **una vez al día**, de esta forma se reducen los conflictos de merge y se evitan los problemas de integración.

¿Y si mi código tiene errores o no pasa las pruebas?

Si el código no está listo para enviar al final del día:

- Se envía un subconjunto coherente
- Se es más flexible con la planificación

3. Integración continua| tipos de builds

Privados

Construcción interna del software para pruebas antes de la liberación pública.

- **Uso**: pruebas internas y desarrollo
- **Acceso**: restringido a desarrolladores
- **Objetivo**: validar cambios y correcciones de manera privada

3. Integración continua| tipos de builds

Privados

Construcción interna del software para pruebas antes de la liberación pública.

- **Uso**: pruebas internas y desarrollo
- **Acceso**: restringido a desarrolladores
- **Objetivo**: validar cambios y correcciones de manera privada

Integración

Construcción automática que integra cambios recientes en el código.

- **Uso**: validar integración continua
- **Acceso**: interno o limitado a equipos específicos
- **Objetivo**: detectar errores de integración tempranos

3. Integración continua| tipos de builds

Privados

Construcción interna del software para pruebas antes de la liberación pública.

- **Uso**: pruebas internas y desarrollo
- **Acceso**: restringido a desarrolladores
- **Objetivo**: validar cambios y correcciones de manera privada

Integración

Construcción automática que integra cambios recientes en el código.

- **Uso**: validar integración continua
- **Acceso**: interno o limitado a equipos específicos
- **Objetivo**: detectar errores de integración tempranos

Entrega

Construcción preparada para ser liberada a los usuarios o producción.

- **Uso**: preparación para despliegue
- **Acceso**: público o cliente
- **Objetivo**: asegurar calidad para lanzamiento final

¿Qué otros tipos de build podríamos hacer?

uvlhub.io

3. Integración continua| automatiza la construcción

- **Build diario**: construir ejecutables diariamente:
  - Permite probar la **calidad** de la integración hasta el momento
  - Ayuda a la **moral** del equipo → el producto ”funciona todos los días” y el progreso es visible
  - Facilita y agiliza la detección de cualquier ***issue*** que rompa la compilación

¿Cuánto tiempo crees que es el límite para una integración básica?

3. Integración continua| automatiza la construcción

- **Build diario**: construir ejecutables diariamente:
  - Permite probar la **calidad** de la integración hasta el momento
  - Ayuda a la **moral** del equipo → el producto ”funciona todos los días” y el progreso es visible
  - Facilita y agiliza la detección de cualquier **issue** que rompa la compilación

Kent Beck en “Extreme Programming Explained” (2004):

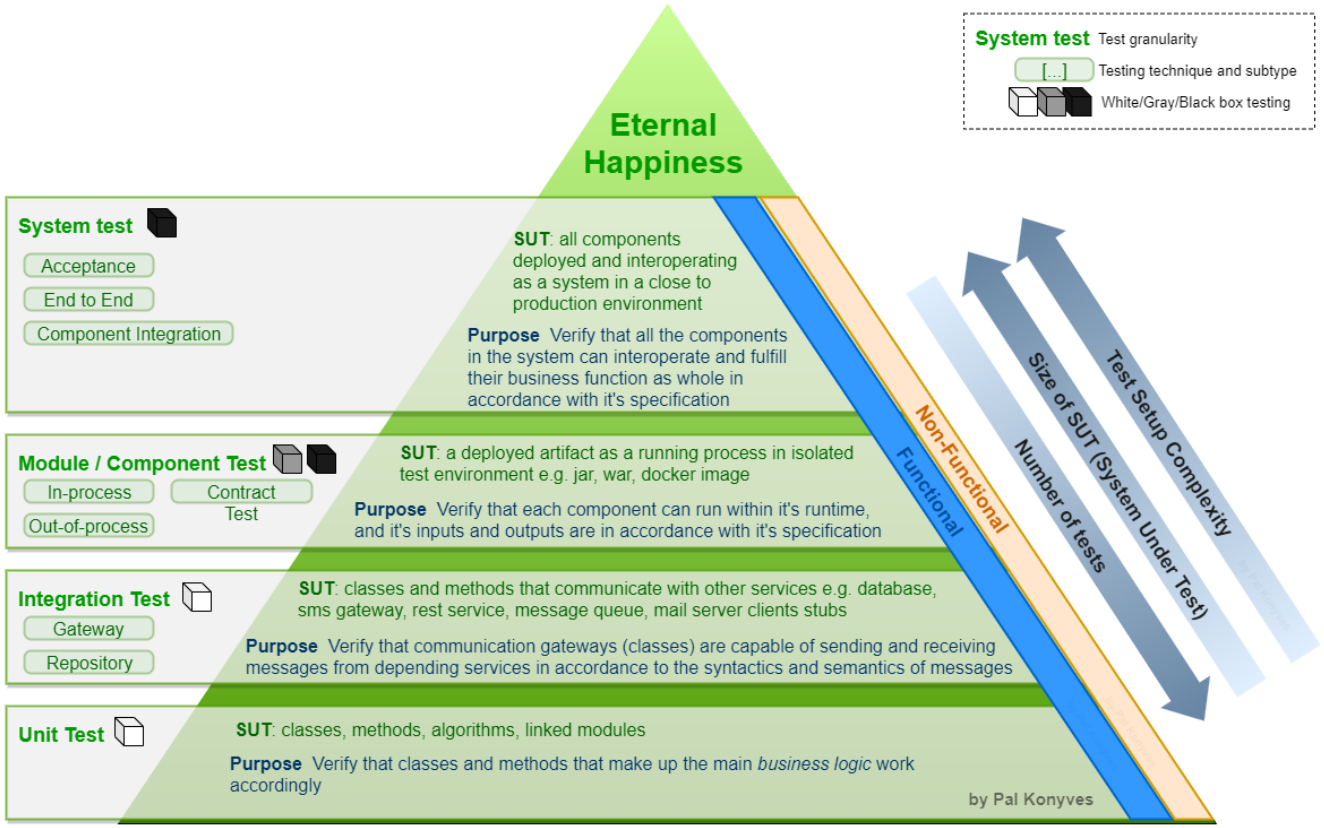
- Límite recomendado para una integración básica: 10 minutos.
- Nota: en algunos casos, incluso 10 minutos puede ser excesivo.

¿Cuánto tiempo crees que es el límite para una integración básica?

1. Introducción
2. Conceptos básicos
3. Integración continua
    - Definición
    - Arquitectura
    - Builds
    - Pruebas y CI
    - Inspección y análisis de código
    - Feedback
    - Lo esencial
4. Resumen
5. Bibliografía

3. Integración continua| automatiza los tests

“Make your build self-testing”



Recuerda la importancia de la línea de comandos

Inspección
y
Análisis de código

1. Introducción
2. Conceptos básicos
3. Integración continua
    - Definición
    - Arquitectura
    - Builds
    - Pruebas y CI
    - Inspección y análisis de código
    - Feedback
    - Lo esencial
4. Resumen
5. Bibliografía

3. Integración continua| feedback

Importancia del Feedback:

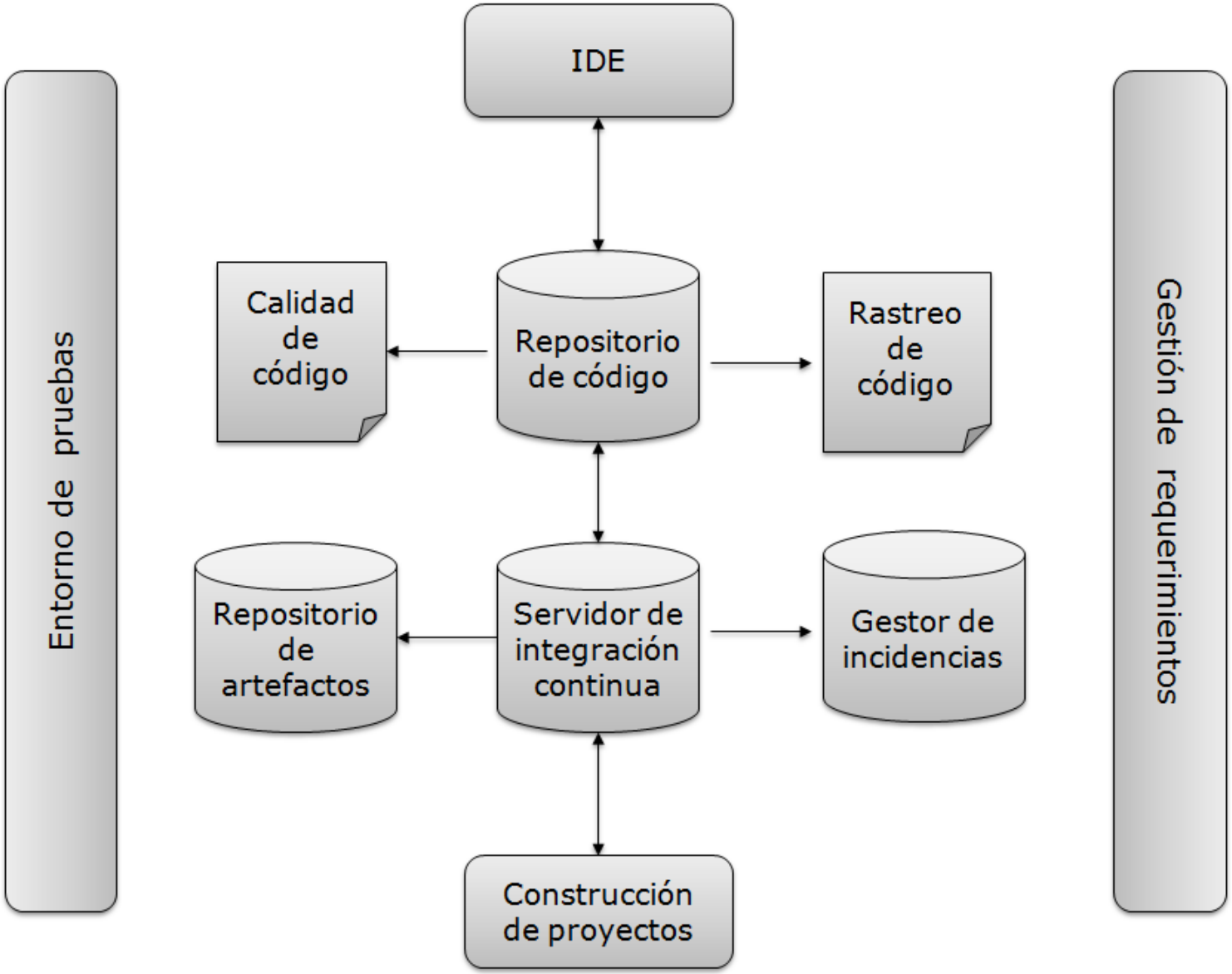
- Detectar errores tempranamente: permite identificar problemas rápidamente y abordarlos antes de que se conviertan en mayores.

- Mejorar la calidad del código: ayuda a mantener y mejorar la calidad del software al proporcionar información continua sobre el estado del código.

- Optimizar el proceso de desarrollo: facilita la adaptación y mejora del flujo de trabajo de desarrollo mediante información constante sobre el rendimiento del proceso.

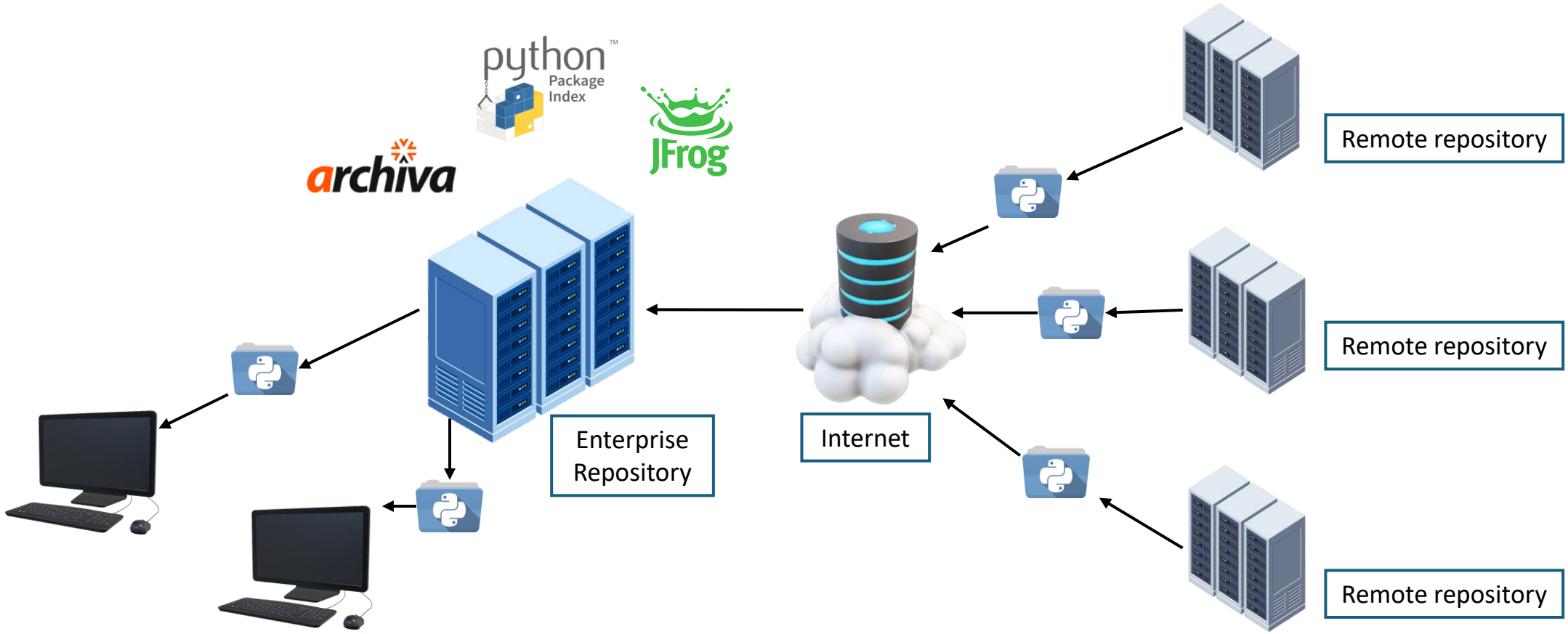
¿Qué información vais a proporcionar en vuestro ciclo de CI?

3. Integración continua| lo esencial



3. Integración continua| gestión de dependencias

Repositorio de artefactos



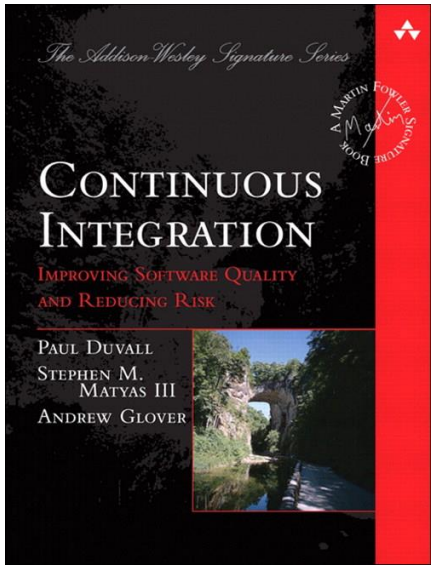
1. Introducción
2. Conceptos básicos
3. Integración continua
    - Definición
    - Arquitectura
    - Builds
    - Pruebas y CI
    - Inspección y análisis de código
    - Feedback
    - Lo esencial
4. Resumen
5. Bibliografía

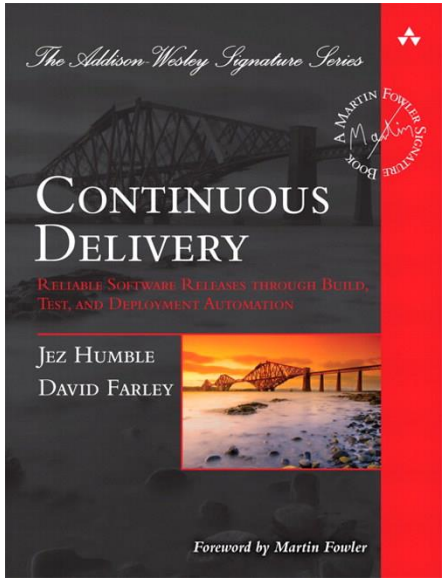
4. Resumen

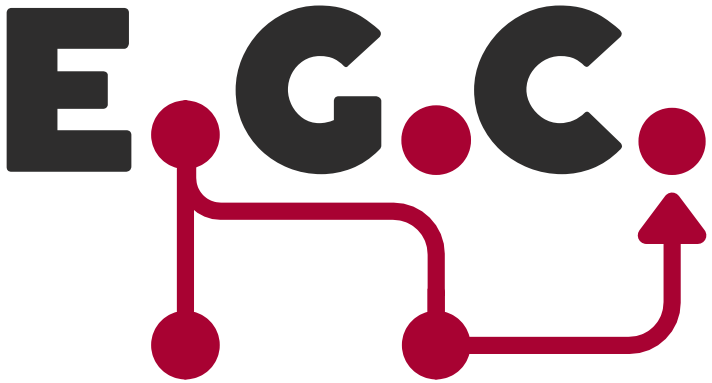
- **¿Qué hemos aprendido?**
  - Construir/ensamblar software es algo más que compilar
  - La integración es un problema que hay que tratar
  - La integración continua aboga por hacerlo desde el primer momento
  - Automatizar la construcción se convierte en fundamental

- **¿Qué veremos en las siguientes lecciones?**
  - Gestión de las incidencias, pruebas, código…
  - En prácticas: herramientas de construcción, servidores de integración continua

5. Bibliografía







Grado en Ingeniería Informática - Ingeniería del Software

Evolución y Gestión de la Configuración

Universidad de Sevilla 1505

Escuela Técnica Superior de Ingeniería Informática

¡Gracias!