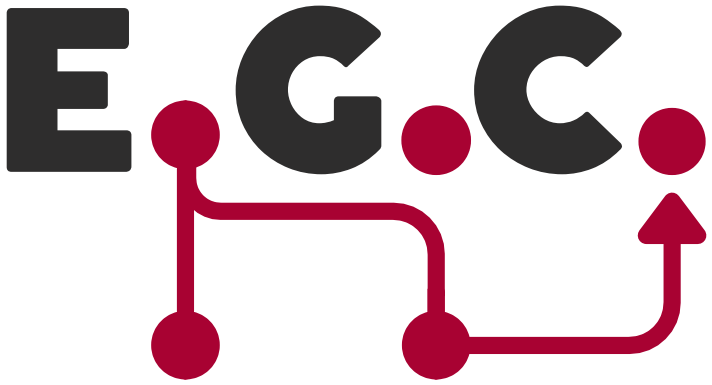


Grado en Ingeniería Informática - Ingeniería del Software

Evolución y Gestión de la Configuración

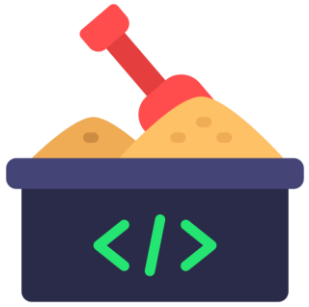
Escuela Técnica Superior de Ingeniería Informática

Tema 1: integración continua, entrega continua y despliegue continuo

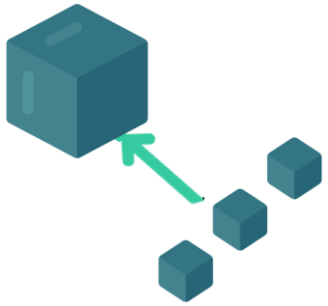
1. **Introducción**
2. Integración continua
3. Entrega continua
4. Despliegue continuo
5. ¿Y cómo afecta la IA a todo esto?

1. Introducción | ejemplo de otro dominio

1. Introducción | crear software



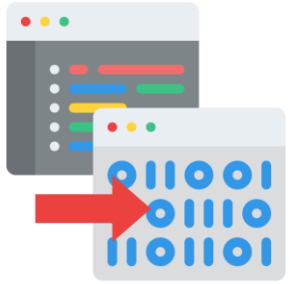
Preparar entorno de desarrollo



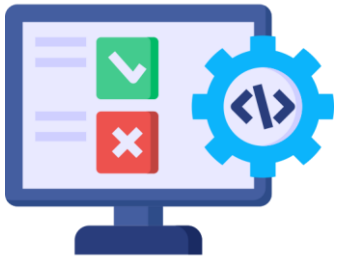
Dependencias

Análisis estático

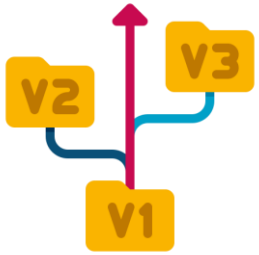
Code review



Compilación



Testing



Versionado

Empaquetado

Entrega

Despliegue

1. Introducción | crear software

Construir software es el proceso técnico de convertir código fuente, recursos, configuraciones y otros elementos del proyecto en un producto funcional que puede ejecutarse.

Lo que no puedo automatizar, lo tengo que gestionar

Gestionar la construcción de software es el proceso de organizar, automatizar y supervisar ese proceso para que ocurra de forma previsible y con criterios compartidos.

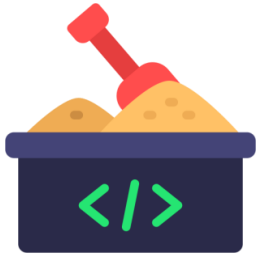
1. **Introducción**
2. **Integración continua**
3. Entrega continua
4. Despliegue continuo
5. ¿Y cómo afecta la IA a todo esto?

2. Integración continua

- La integración continua propone integrar cambios pequeños de manera muy frecuente en una línea principal compartida del repositorio de software, y validar automáticamente que el sistema sigue construyéndose correctamente y superando las pruebas.

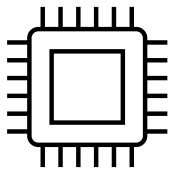
- Pero para hacer esto posible necesitamos entender antes los pasos principales de la integración continua: entornos locales, dependencias, análisis estático, revisión de código, compilación, pruebas y empaquetado.

2. Integración continua | entorno local

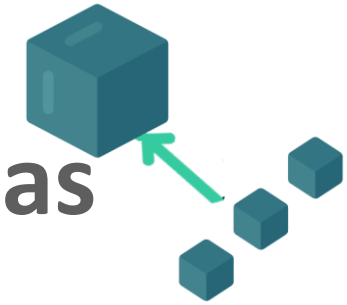


- **Configuración reproducible:** que el entorno pueda reconstruirse a partir de unas instrucciones, que mantendremos también en nuestro repositorio.
- Suele incluir ajustes de editor, extensiones, perfiles de compilación, variables de entorno y configuraciones específicas del proyecto
- Se puede automatizar la configuración del entorno usando scripts, ficheros de configuración o contenedores.

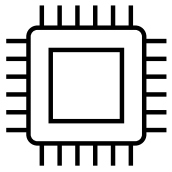
Black



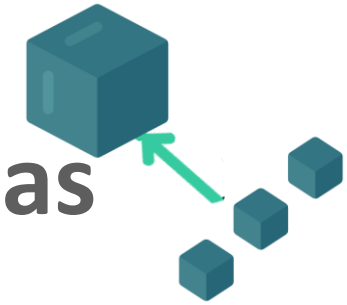
2. Integración continua| gestión de dependencias



- **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.

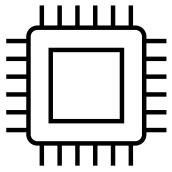


2. Integración continua| gestión de dependencias

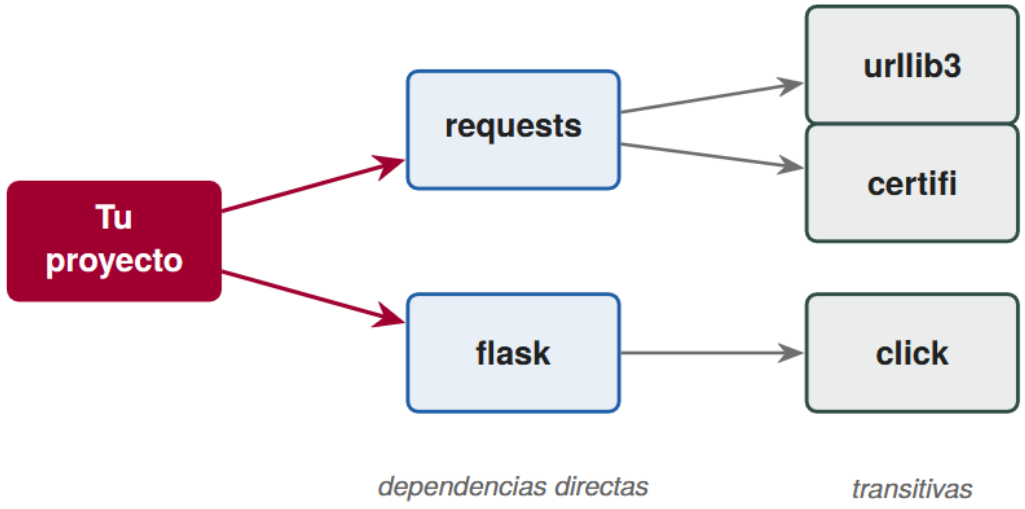


- **Instalación de dependencias:** ejecutar comandos para instalar las dependencias necesarias usando herramientas como `npm install` o `pip install`.
- **Automatización de instalación:** automatizar el proceso de instalación de dependencias como parte del flujo de integración continua.
- **Verificación 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.



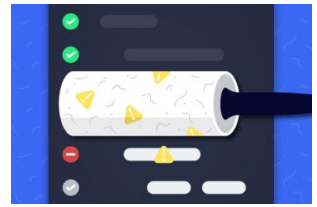
2. Integración continua | gestión de dependencias




```
$ pipdeptree from-index "fastapi<=0.115.2" starlette
fastapi==0.115.2
├── pydantic [candidate: 2.13.4]
│   ├── annotated-types [candidate: 0.7.0]
│   ├── pydantic-core [candidate: 2.46.4]
│   │   └── typing-extensions [candidate: 4.16.0]
│   ├── typing-extensions [candidate: 4.16.0]
│   └── typing-inspection [candidate: 0.4.2]
│       └── typing-extensions [candidate: 4.16.0]
├── starlette [candidate: 0.40.0]
│   └── anyio [candidate: 4.14.2]
│       └── idna [candidate: 3.18]
└── typing-extensions [candidate: 4.16.0]
```

pipdeptree

2. Integración continua| análisis estático



- Las herramientas de análisis estático de código, linters, analizan el código sin ejecutarlo para detectar **posibles problemas, inconsistencias de estilo y antipatrones**.
- El objetivo es detectar problemas de forma temprana, aplicar criterios de consistencia a todo el proyecto y escalar comprobaciones que serían tediosas de revisar manualmente en cada cambio.

if-with-same-arms (SIM114)

What it does

Checks for `if` branches with identical arm bodies.

Why is this bad?

If multiple arms of an `if` statement have the same body, using `or` better signals the intent of the statement.

Example

```
if x == 1:
    print("Hello")
elif x == 2:
    print("Hello")
```

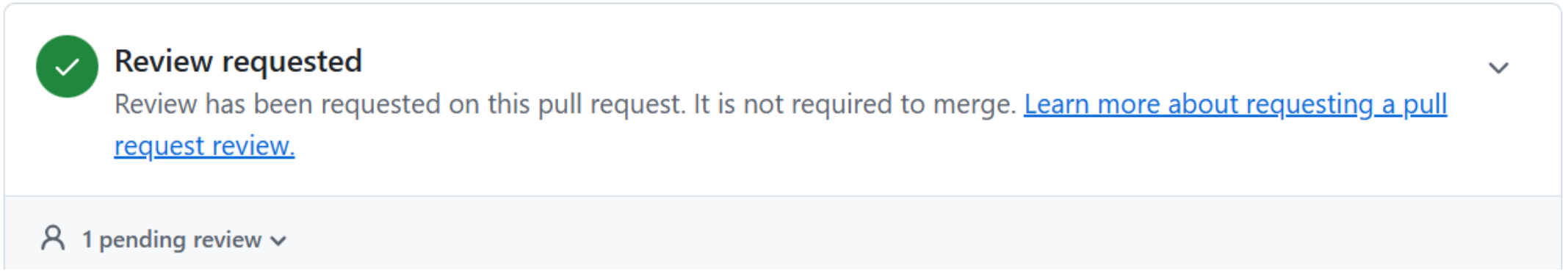
Use instead:

```
if x == 1 or x == 2:
    print("Hello")
```

2. Integración continua| revisiones de código

Inspección de Código:

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



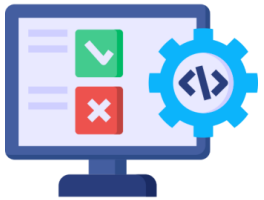
2. Integración continua | compilación

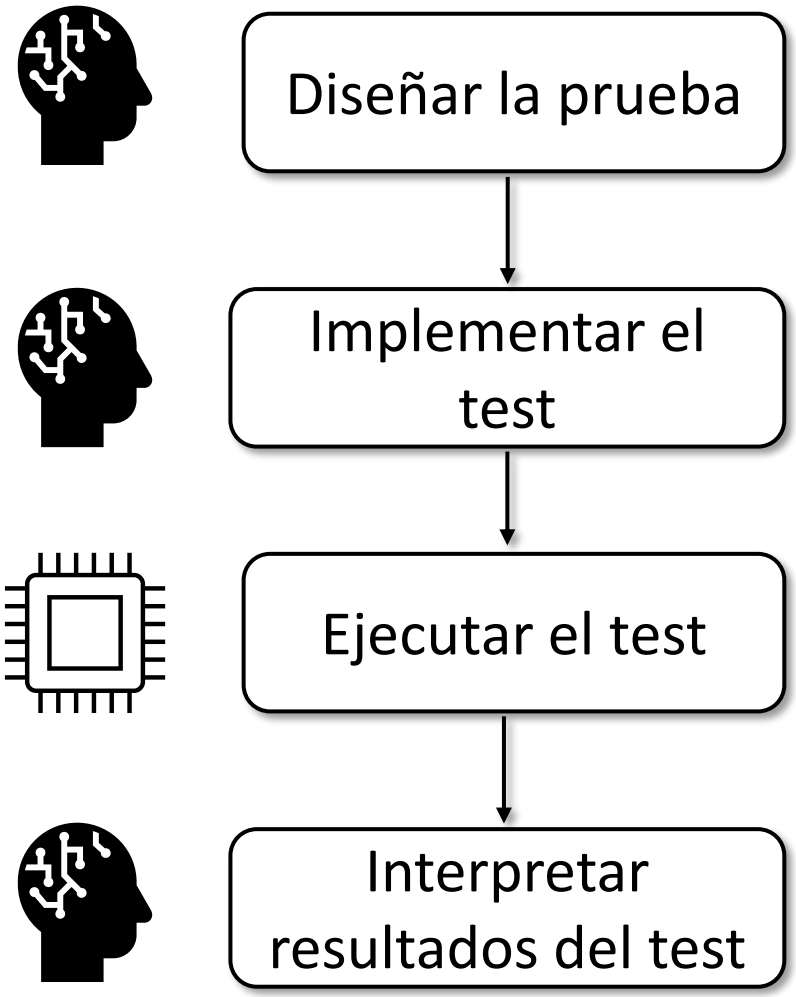
- **Automatiza el proceso de compilación**: ejecutar comandos predefinidos para compilar el código automáticamente, (mvn compile, npm run build, etc.).
- **Detecta errores de compilación**: detectar automáticamente errores de compilación en el código y detener el proceso.
- **Compilación multiplataforma**: 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. Integración continua| 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.

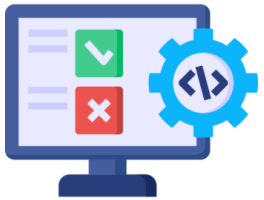
2. Integración continua| testing

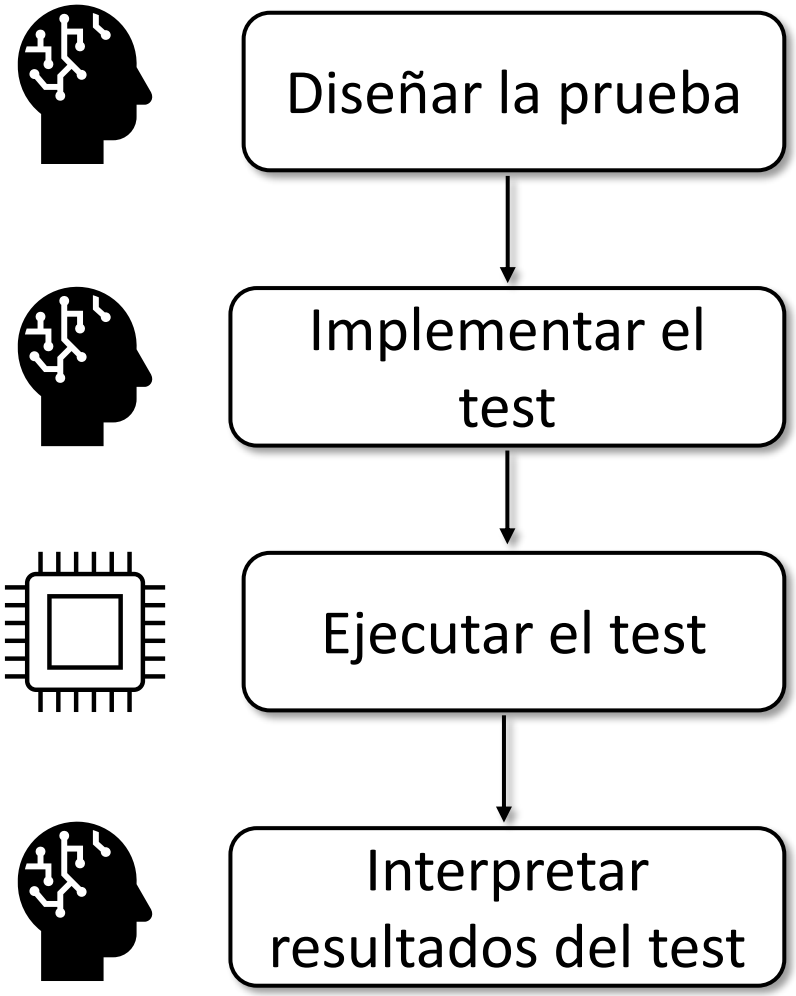




- Definir qué aspectos del software deben ser probados, cómo se estructurarán las pruebas y qué criterios se usarán para evaluar los resultados. Requiere conocimiento sobre el software, los objetivos de la prueba y la identificación de casos de prueba adecuados.

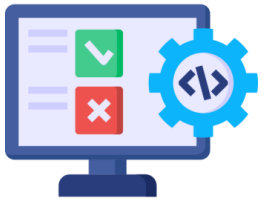
2. Integración continua| testing





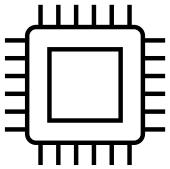
- Definir qué aspectos del software deben ser probados, cómo se estructurarán las pruebas y qué criterios se usarán para evaluar los resultados. Requiere conocimiento sobre el software, los objetivos de la prueba y la identificación de casos de prueba adecuados.
- Escribir el código para las pruebas,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



Diseñar la prueba

Implementar el test

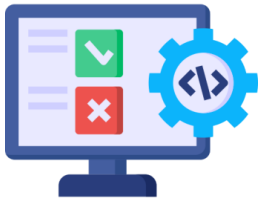


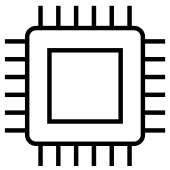
Ejecutar el test

Interpretar resultados del test

- 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.

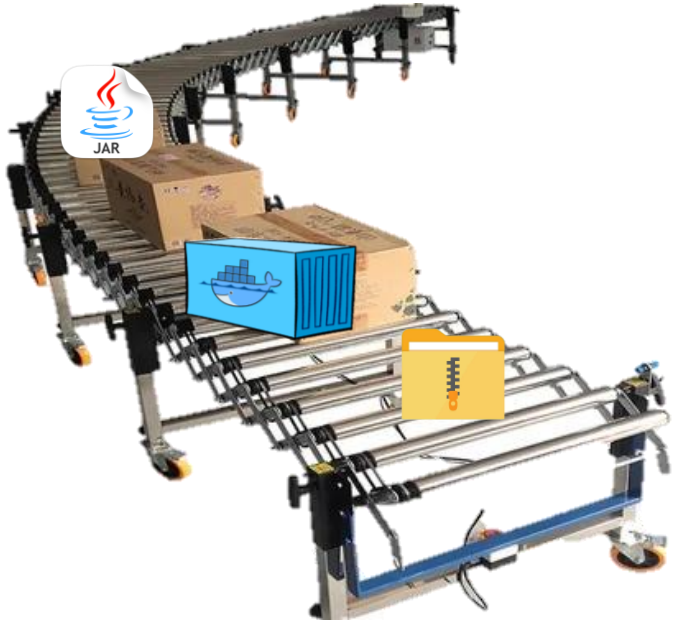
2. Integración continua| testing



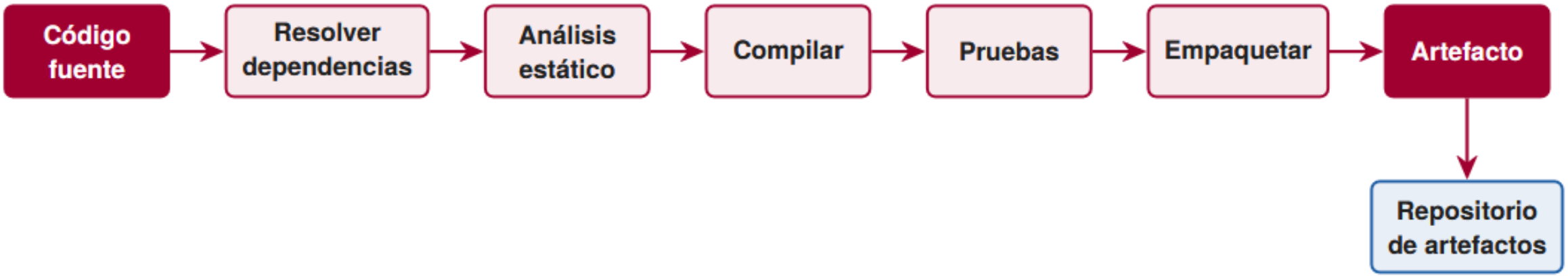


La ejecución de pruebas puede ser automatizada, pero el diseño, la implementación y la interpretación requieren intervención humana para asegurar pruebas efectivas y análisis precisos.

2. Integración continua | empaquetado

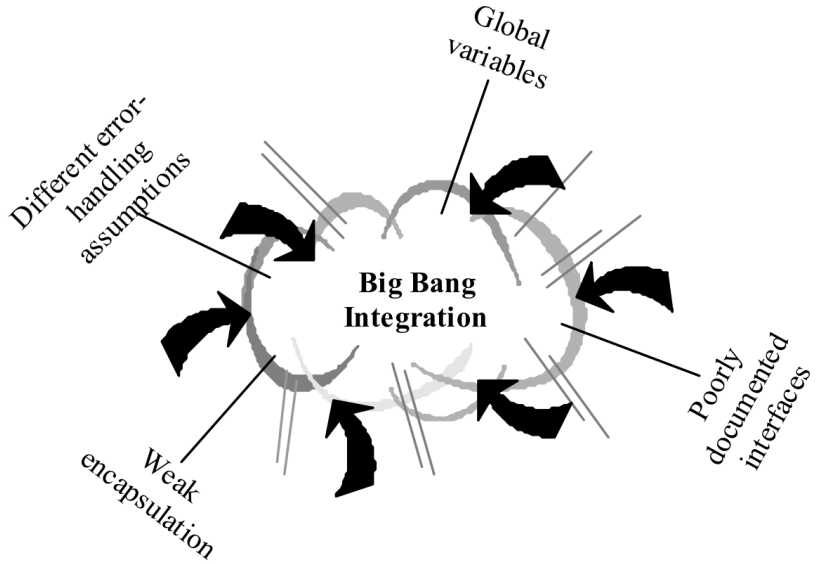


- Definición de Estrategias de Empaquetado
- Configuraciones Personalizadas
- Integración de Recursos
- Generación de Artefactos



2. Integración continua| ideas fundamentales

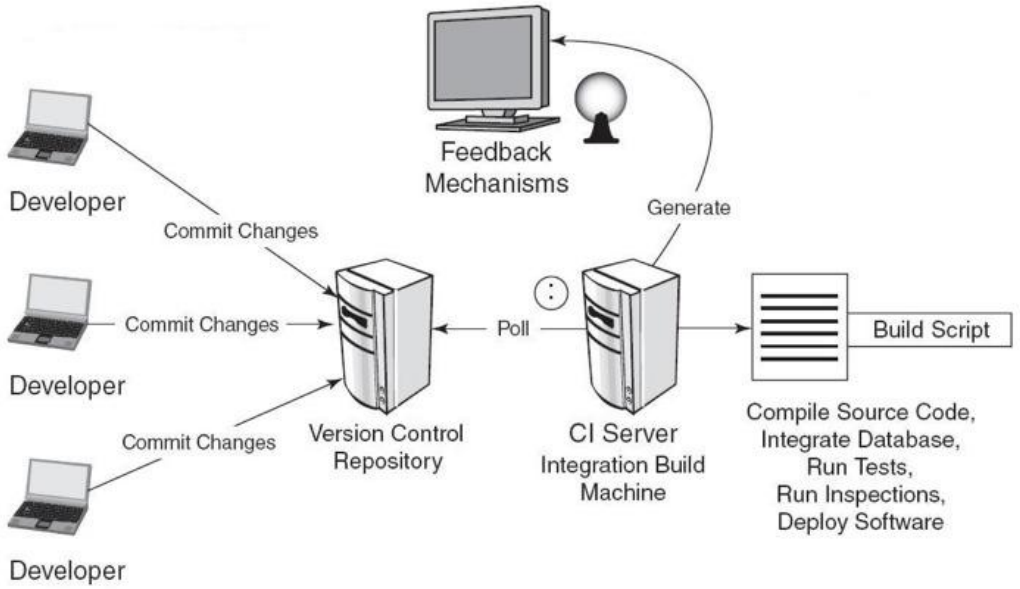
- Evitar el *integration hell* (o *merge hell)*
- Si algo duele, hazlo más a menudo y adelanta el dolor
- Así los problemas aparecerán pronto y cuando todavía son pequeños



PIPELINE

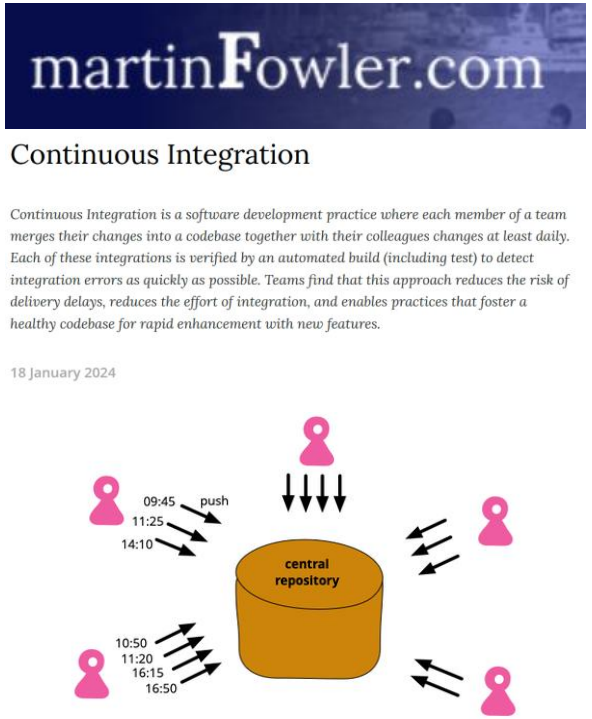
2. Integración continua | ciclo básico

1. Una persona realiza un cambio en el código.
2. Ese cambio se integra en la línea principal compartida del repositorio, normalmente tras una revisión.
3. Un servidor o servicio de CI detecta el cambio.
4. El servidor obtiene una copia limpia del repositorio.
5. Ejecuta automáticamente el proceso de construcción: resuelve dependencias, compila, analiza, pasa las pruebas y, si procede, empaqueta.
6. Publica el resultado del proceso: éxito, fallo, logs, informes de pruebas o artefactos generados.
7. El equipo recibe feedback y actúa en consecuencia.



2. Integración continua| principios prácticos

1. Poner todo en la línea principal de un sistema de control de versiones.
2. Automatizar el build.
3. Hacer que el build sea auto-testeable.
4. Integrar cambios en la línea principal al menos una vez al día, idealmente con más frecuencia.
5. Lanzar una integración automática por cada cambio integrado.
6. Mantener el build rápido.
7. Arreglar un build roto inmediatamente.
8. Esconder el trabajo que no está terminado.
9. Probar en un entorno lo más parecido posible al entorno real.
10. Hacer visible para todo el equipo qué está ocurriendo.



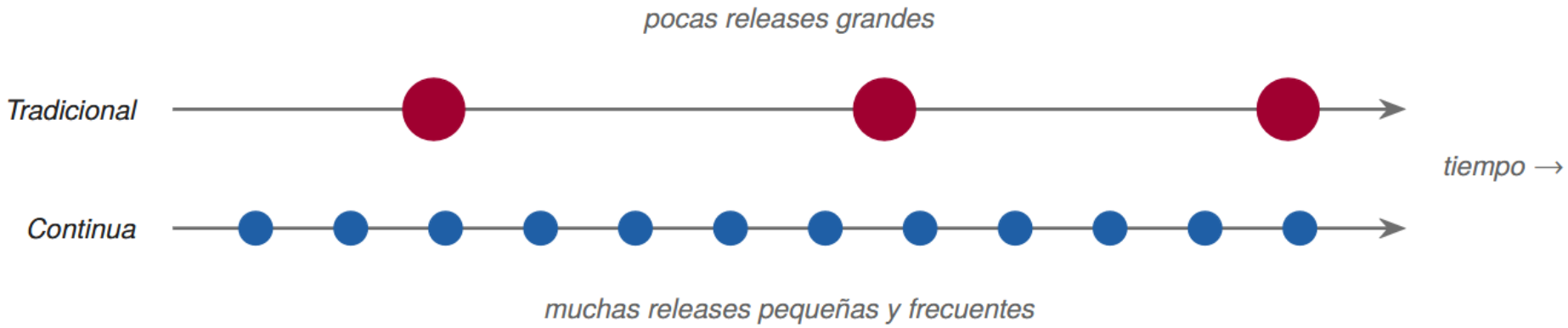
1. Introducción
2. Integración continua
3. Entrega continua
4. Despliegue continuo
5. ¿Y cómo afecta la IA a todo esto?

3. Entrega continua| hacer una *release*

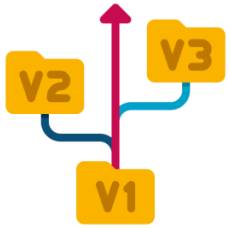
- Tomar un artefacto construido, validarlo y convertirlo en una versión identificada, trazable y potencialmente entregable
- Incluye actividades como:
  - asignar un número de versión
  - etiquetar el código
  - generar notas de versión
  - almacenar el artefacto en un repositorio o canal de distribución
  - dejar registrada la relación entre código fuente, artefacto, versión y entorno

3. Entrega continua| algo de historia

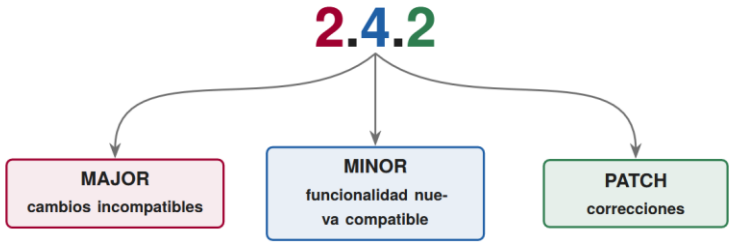
- Históricamente, hacer una *release* era un evento muy importante, planificado y bastante manual.
- Existía la figura de *release manager* o *release captain*.



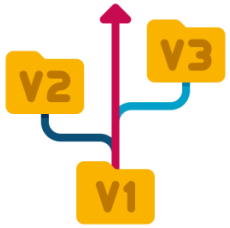
3. Entrega continua| versionado



- 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/



3. Entrega continua| versionado



Release list

- TensorFlow 2.21.0
- TensorFlow 2.21.0-rc1
- TensorFlow 2.21.0-rc0
- TensorFlow 2.19.1
- TensorFlow 2.20.0
- TensorFlow 2.20.0-rc0
- TensorFlow 2.18.1
- TensorFlow 2.19.0
- TensorFlow 2.19.0-rc0
- TensorFlow 2.18.0

‹ Previous Next ›

TensorFlow 2.21.0 Latest

Compare

tensorflow-jenkins released this Mar 6 v2.21.0 a481b10

Release 2.21.0

TensorFlow

Breaking Changes

- Support for Python 3.9 has been removed starting with TF 2.21.
- The TensorBoard (TB) dependency has been removed starting with TF 2.21.

Major Features and Improvements

- `tf.lite`
  - Adds int8 and int16x8 support for SQRT operator.
  - Adds int16x8 support for EQUAL and NOT_EQUAL operators.
  - Adds support for int2 type.

3. Entrega continua | siempre en estado entregable

- *Delivery pipeline* = ruta definida por el equipo para decidir si una versión estaría lista para avanzar a producción (repetible y trazable)
  - toma cada cambio integrado
  - construye un artefacto desplegable
  - lo almacena como *release candidate*
  - lo valida progresivamente
  - determina si puede considerarse liberable

- El software está siempre en un estado entregable.

1. Introducción
2. Integración continua
3. Entrega continua
4. Despliegue continuo
5. ¿Y cómo afecta la IA a todo esto?

4. Despliegue continuo| desplegar

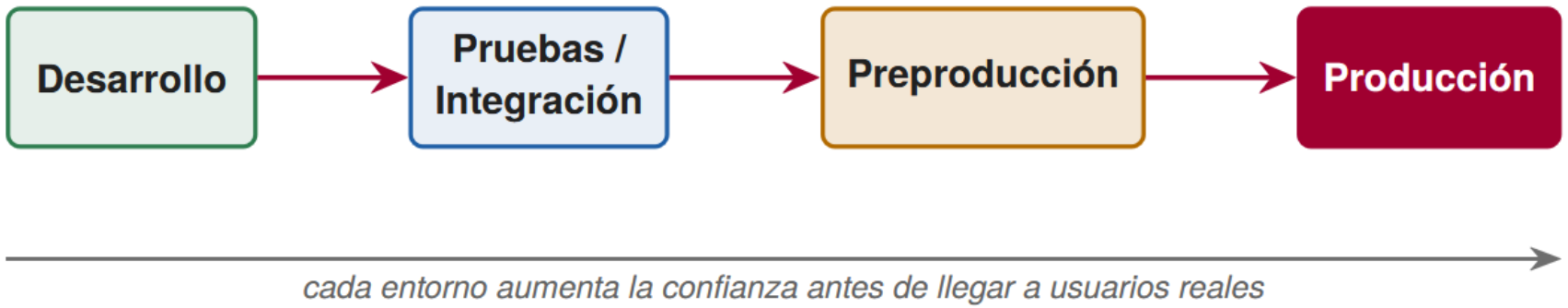
- Llevar una release a un entorno concreto para que pueda ejecutarse allí.

- Ejemplos:
  - instalar un paquete en un servidor
  - arrancar una nueva imagen Docker en un cluster
  - actualizar una función serverless
  - copiar una web estática a un sistema de publicación
  - instalar una nueva versión de un servicio interno
  - ...

- Entregar ≠ desplegar:
  - una release puede existir aunque no se haya desplegado todavía
  - una misma release puede desplegarse en varios entornos distintos

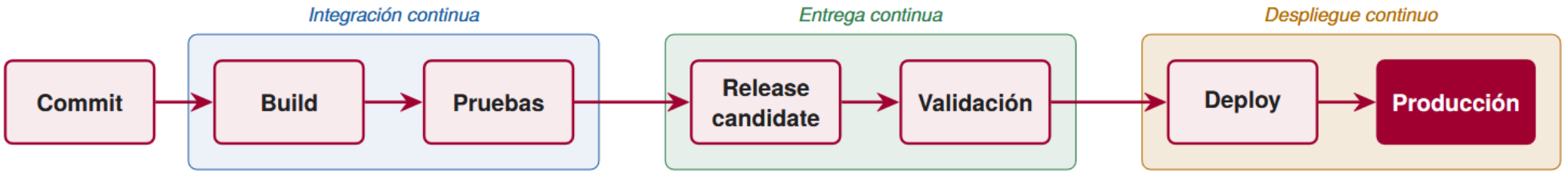
4. Despliegue continuo| entornos

- Una máquina (física o virtual) con su SO, bibliotecas, herramientas…
- En proyectos reales lo habitual es que se tengan varios entornos con objetivos distintos.



4. Despliegue continuo| llevar cada entrega a producción automáticamente

- Si una release supera todas las validaciones necesarias, el sistema la despliega automáticamente en producción.
- Los despliegues están automatizados y documentados para que sean fiables, reproducibles y trazables.
- En muchos tipos de sistemas es habitual que se realicen múltiples despliegues diarios a producción .



4. Despliegue continuo| no siempre encaja

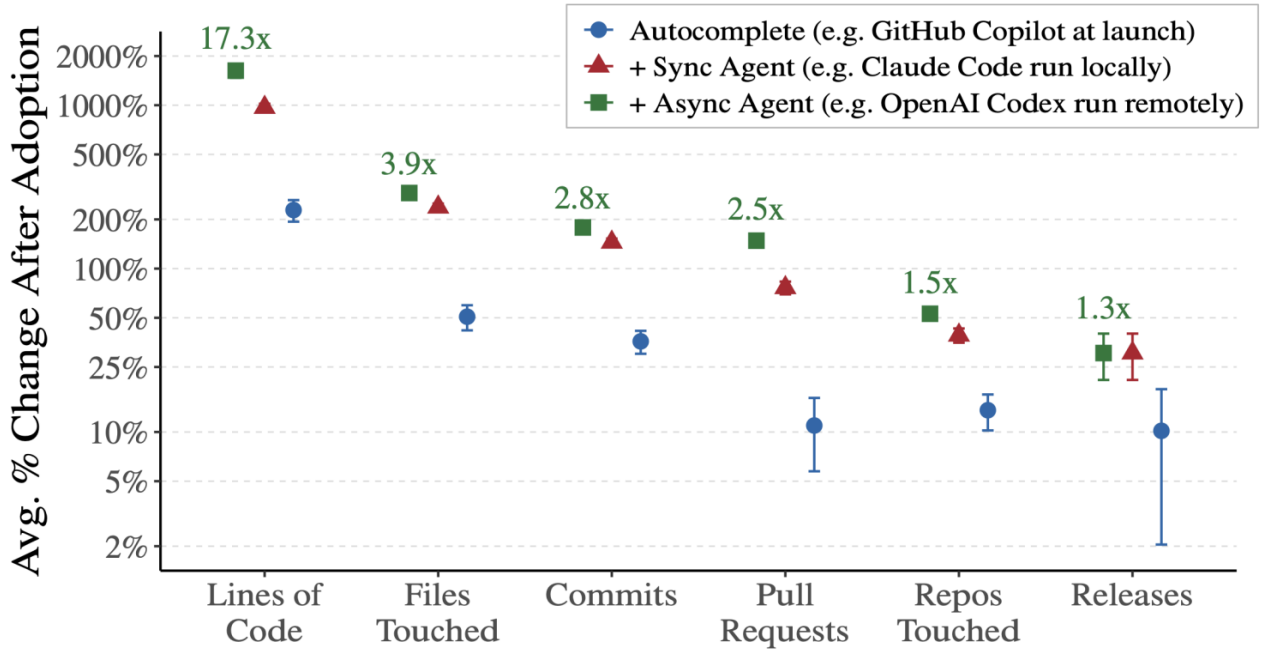
- Servicios web, APIs y sistemas en la nube donde hay control del entorno, se pueden automatizar validaciones y hay mecanismos para detectar problemas.

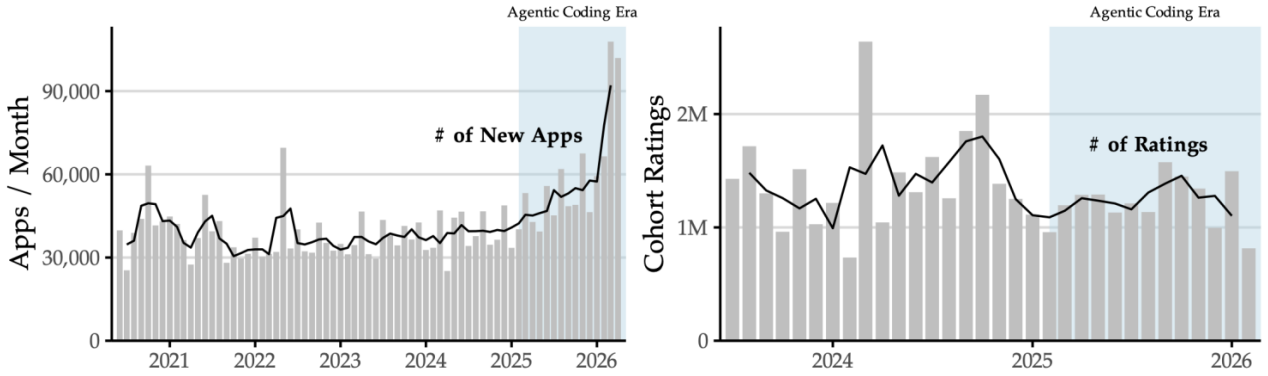
- Cuando no hay tanto control:
  - App móvil, que puede depender de aprobación de una tienda
  - Aplicación de escritorio, que puede requerir instaladores, firma y actualizaciones graduales
  - Firmware, que puede necesitar validaciones muy estrictas

- En esos casos, más razonable quedarse en entrega continua:
  - El producto está siempre preparado para liberarse
  - Pero la publicación o distribución final sigue una política más controlada.

1. Introducción
2. Integración continua
3. Entrega continua
4. Despliegue continuo
5. ¿Y cómo afecta la IA a todo esto?

5. ¿Y cómo afecta la IA a todo esto?





Writing Code vs. Shipping Code: Productivity Effects Across Generations of AI Coding Tools

NBER Working Paper No. w35275

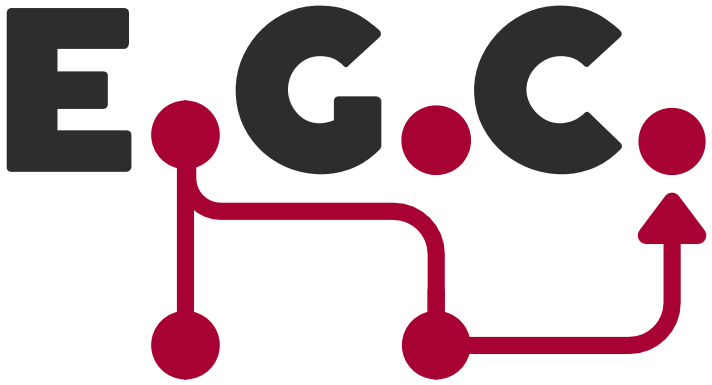
97 Pages • Posted: 3 Jun 2026 • Last revised: 15 Jun 2026

Mert Demirer
Sloan School of Management

Leon Musolff
University of Pennsylvania,

Liyuan Yang
Sloan School of Management

NBER | NATIONAL BUREAU of ECONOMIC RESEARCH



Grado en Ingeniería Informática - Ingeniería del Software

Evolución y Gestión de la Configuración

Universidad de Sevilla 1505

Escuela Técnica Superior de Ingeniería Informática

¡Gracias!