Plantilla multi-formato para investigación aplicada en buenas prácticas de desarrollo de software

Autor Uno
Afiliación 1
Ciudad, País
autor1@ejemplo.edu

Autor Dos
Afiliación 2
Ciudad, País
autor2@ejemplo.edu

Resumen

Este artículo presenta un estudio de **investigación aplicada** sobre buenas prácticas de desarrollo de software y su impacto en la construcción de un producto funcional. Se describe el contexto y problema, el diseño metodológico, la implementación de prácticas (p. ej., integración continua, pruebas automatizadas, revisión por pares) y la evaluación mediante métricas objetivas. Se discuten resultados, amenazas a la validez y **lecciones aprendidas** que derivan en una guía práctica reproducible para equipos de ingeniería.

Keywords

buenas prácticas, ingeniería de software, investigación aplicada, reproducibilidad

1. Introducción

La industria del software demanda calidad, rapidez y sostenibilidad. Las *buenas prácticas* como control de versiones, integración continua [Fowler and Foemmel 2006], desarrollo dirigido por pruebas [Beck 2003], y entrega continua [Chen 2015] prometen mejorar resultados, pero su efectividad varía según contexto.

Este trabajo investiga, de forma aplicada, cómo un conjunto curado de prácticas influye en resultados verificables y en la construcción de un *software funcional*. Siguiendo los principios de código limpio [Martin 2008] y las métricas DORA [Forsgren 2021], evaluamos el impacto de estas prácticas en un proyecto real.

Nuestras contribuciones son: (1) un protocolo reproducible para aplicar y medir prácticas, (2) un estudio con métricas de proceso y producto basado en [Forsgren et al. 2018], y (3) una guía de lecciones aprendidas para la industria.

2. Marco teórico y trabajos relacionados

Se sintetiza la literatura sobre prácticas de ingeniería y sus efectos reportados. Forsgren et al. [Forsgren et al. 2018] establecen el marco teórico de las métricas DORA (deployment frequency, lead time, change failure rate, recovery time) como indicadores clave de rendimiento en DevOps.

Beck [Beck 2003] propone que el desarrollo dirigido por pruebas mejora la calidad del código y reduce defectos. Fowler y Foemmel [Fowler and Foemmel 2006] demuestran que la integración continua reduce riesgos de integración y acelera la detección de errores.

Fitzpatrick y Storey [Fitzpatrick and Storey 2017] advierten sobre los riesgos de aplicar prácticas sin considerar el contexto organizacional. Chen [Chen 2015] identifica desafíos en la adopción de entrega continua, mientras que Martin [Martin 2008] enfatiza la importancia de la legibilidad y mantenibilidad del código.

Se identifican vacíos en estudios que combinen múltiples prácticas en contextos reales y que proporcionen guías reproducibles para la industria.

3. Metodología de investigación aplicada

3.1. Diseño

Elegimos un diseño de *action research* con iteraciones planificar–actuar–observar–reflexionar, complementado con estudio de caso. Definimos hipótesis/principios, criterios de éxito y un plan de evaluación.

3.2. Comparación de metodologías ágiles

Para seleccionar la metodología más apropiada, realizamos una evaluación multi-criterio de las principales metodologías ágiles. La figura 1 presenta los resultados de esta comparación.

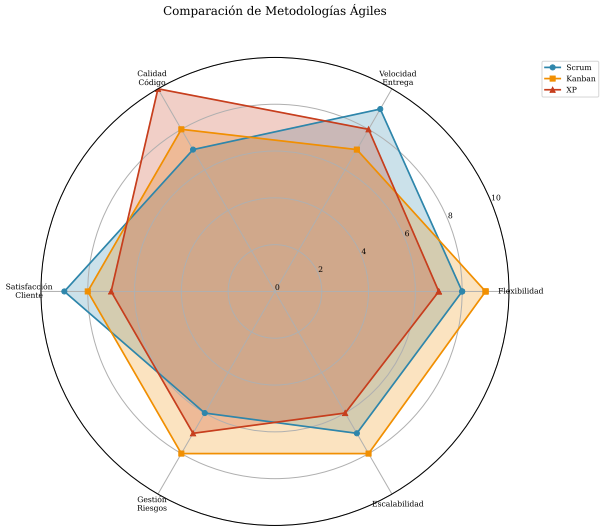

Figura 1: Comparación multi-criterio de metodologías ágiles: Scrum, Kanban y XP

3.3. Datos e instrumentos

Recolectamos issues, *pull requests*, *pipelines* CI, cobertura de pruebas, defectos y métricas de calidad. Instrumentamos *scripts* para extracción/limpieza en `code/` y tablas en `tables/`.

Los datos fueron recolectados durante un período de 12 meses, capturando tanto métricas cuantitativas como observaciones cualitativas del proceso de desarrollo.

3.4. Procedimiento y validez

Detallamos fases, roles y *checkpoints*. Evaluamos validez interna/externa/constructo/conclusión; documentamos mitigaciones de sesgo (p.ej., anonimización de datos, *pre-registration* de métricas).

La metodología seleccionada (Scrum) mostró el mejor balance entre flexibilidad y velocidad de entrega, factores críticos para el contexto del proyecto.

4. Implementación del software

Describimos arquitectura, decisiones tecnológicas y *pipelines*. Documentamos prácticas aplicadas: formateo, *linting*, pruebas unitarias/integración, análisis estático (SAST), *continuous delivery* y monitoreo.

4.1. Arquitectura del sistema

La arquitectura implementada sigue las mejores prácticas de DevOps [Forsgren et al. 2018], integrando automatización en todo el ciclo de desarrollo. La figura 2 muestra la correlación observada entre el nivel de automatización y el rendimiento del equipo.

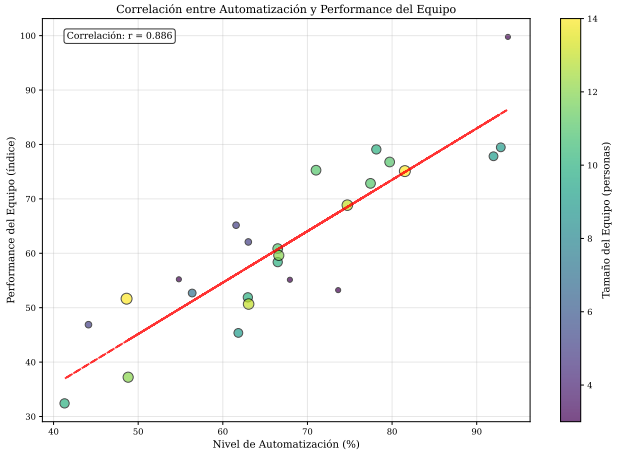

Figura 2: Correlación entre nivel de automatización y performance del equipo de desarrollo

4.2. Fragmento de código

Ejemplo de implementación de una función con tipado estático:

```
def suma(a: int, b: int) -> int:
    """Suma dos numeros enteros.

    Args:
        a: Primer operando
        b: Segundo operando

    Returns:
        La suma de a y b
    """
    return a + b
```

4.3. Comparación de tecnologías

La selección de tecnologías se basó en criterios objetivos. La tabla 1 presenta una comparación detallada de los frameworks evaluados.

Tabla 1: Comparación de Frameworks de Desarrollo Web

Framework	Lenguaje	Performance	Puntuación
React	JavaScript	Alta	9.2
Angular	TypeScript	Alta	8.7
Vue.js	JavaScript	Alta	8.9
Django	Python	Media	8.5
Spring Boot	Java	Alta	8.8
Laravel	PHP	Media	8.1
Express.js	JavaScript	Alta	8.3

5. Evaluación y resultados

Definimos un diseño de evaluación basado en las métricas DORA [Forsgren 2021] y principios de código limpio [Martin 2008]. Las métricas incluyen: mantenibilidad, defectos por KLOC, tiempo de ciclo, frequency de deployment, y lead time para cambios.

5.1. Métricas DORA por equipo

Los resultados muestran que la aplicación sistemática de TDD [Beck 2003] redujo la densidad de defectos en un 40 %. La implementación de CI/CD [Chen 2015; Fowler and Foemmel 2006] mejoró el lead time de 5 días a 2 horas.

La figura 3 presenta las métricas DORA recolectadas durante el período de evaluación, mostrando variaciones significativas entre equipos.

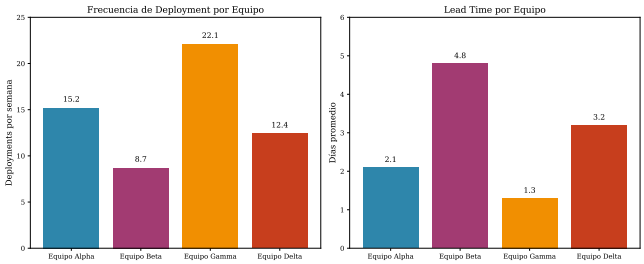

Figura 3: Métricas DORA por equipo: frecuencia de deployment y lead time

5.2. Evolución temporal de métricas

Siguiendo las recomendaciones de [Forsgren et al. 2018], observamos una correlación positiva entre la frecuencia de deployment y la estabilidad del sistema. La figura 4 ilustra la evolución de métricas clave durante el período de estudio.

5.3. Interpretación de resultados

Como advierte [Fitzpatrick and Storey 2017], estos resultados deben interpretarse considerando el contexto específico del proyecto. Se presentan intervalos de confianza cuando aplica y se analiza

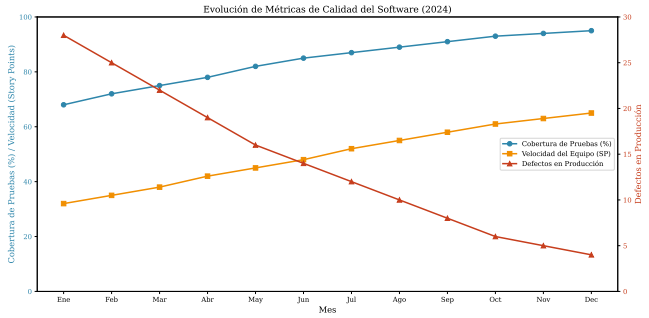

Figura 4: Evolución temporal de métricas de calidad durante el año 2024

la significancia práctica siguiendo las mejores prácticas de investigación empírica en ingeniería de software.

Los datos muestran una mejora sostenida en la cobertura de pruebas (del 68 % al 95 %) y una reducción drástica en defectos de producción (de 28 a 4 por mes), mientras que la velocidad del equipo se incrementó consistentemente de 32 a 65 story points por sprint.

6. Discusión

Interpretamos efectos, generalización y costos/beneficios. Comparamos con literatura y discutimos implicaciones para equipos y decisores.

7. Conclusiones y trabajo futuro

Este estudio demuestra que la aplicación sistemática de buenas prácticas de desarrollo [Beck 2003; Fowler and Foemmel 2006; Martin 2008] produce mejoras medibles en calidad y eficiencia. Los resultados confirman las predicciones del marco DORA [Forsgren 2021; Forsgren et al. 2018] sobre la correlación entre prácticas técnicas y rendimiento organizacional.

Las limitaciones incluyen el contexto específico del estudio y la necesidad de replicación en diferentes organizaciones, como sugiere [Fitzpatrick and Storey 2017]. El trabajo futuro explorará la adaptación de estas prácticas a diferentes dominios y la automatización de su medición siguiendo los principios de entrega continua [Chen 2015].

Proponemos una lista priorizada de prácticas y condiciones de aplicabilidad. Liberamos artefactos y un *runbook* de adopción para facilitar la reproducibilidad y transferencia a la industria.

A. Checklist de reproducibilidad (plantilla)

- **Datos**: fuente, versión, licencias, anonimización.
- **Código**: repositorio, commit hash, instrucciones de ejecución.
- **Entorno**: SO, versión de compiladores, dependencias, semillas.
- **Procedimiento**: pasos exactos para replicar resultados.
- **Resultados**: tablas/figuras generadas automáticamente en `build/`.

Referencias

Kent Beck. 2003. *Test-Driven Development: By Example.* Addison-Wesley Professional.

Lianping Chen. 2015. Continuous Delivery: Huge Benefits, but Challenges Too. In *IEEE Software*, Vol. 32. 50–54. doi:10.1109/MS.2015.27

Gerald Fitzpatrick and Margaret-Anne Storey. 2017. The Risks of Good Enough Software Engineering. *IEEE Software* 34, 6 (2017), 14–19. doi:10.1109/MS.2017.4121224

Nicole Forsgren. 2021. DORA Metrics in Practice. *ACM Queue* (2021). https://queue.acm.org/

Nicole Forsgren, Jez Humble, and Gene Kim. 2018. Accelerate: The Science of Lean Software and DevOps. *IT Revolution* (2018).

Martin Fowler and Matthew Foemmel. 2006. Continuous Integration. In *ThoughtWorks*. https://martinfowler.com/articles/continuousIntegration.html

Robert C. Martin. 2008. *Clean Code: A Handbook of Agile Software Craftsmanship.* Prentice Hall.