UNIVERSIDAD DEL VALLE DE GUATEMALA

Ingeniería de Software – Sección 20

Ing. Pablo Barreno Koch

Tarea de mejora al proceso 5. Principios Lean SW Development.

Osman Emanuel de León García – 23428

Milton Giovanni Polanco Serrano - 23471

Gadiel Amir Ocaña Véliz - 231270

Guatemala, 26 de septiembre de 2025

Tabla Comparativa de Principios Lean

Principio	% Inicial (Abril)	% Actual (Oct)	Explicación	Mejoras o Retrocesos
Eliminar desperdicios	70%	88%	**+18%**. Eliminamos código duplicado (18%→12%) con Factory Pattern. Implementamos CI/CD que redujo deployment de 45min a 4.5min (-90%). ESLint identifica código muerto automáticamente. Documentación actualizada en tiempo real.	- Pipeline CI/CD automatizado - ESLint para código muerto - Duplicación reducida 56% - Buffers 20% para integraciones
Amplificar el aprendizaje	85%	92%	**+7%**. Creamos wiki interna en Notion con lecciones aprendidas. Cobertura de pruebas aumentó 68%→82%. Pair programming rotativo en tareas críticas. Knowledge sharing semanal sobre CI/CD, Redis, seguridad.	- Wiki técnica completa - Cobertura pruebas +14% - Pair programming obligatorio - Knowledge sharing semanal
Tomar decisiones tarde	60%	75%	**+15%**. Implementamos POCs antes de decisiones (Redis, Vercel). Feature flags para reversibilidad. Rollback exitoso en 2min. Matriz de evaluación para comparar opciones técnicas.	- POCs para validación - Feature flags implementados - Matriz de evaluación - Rollback automático
Entregar lo antes posible	90%	95%	**+5%**. CI/CD permite 28 deploys en Sprint #9. Ambientes staging/production separados. Tasa éxito 96% (27/28). Velocidad 100% en Sprints #8-9.	- Continuous Delivery - 1.3 deploys/día - Health checks automáticos - Feedback inmediato
Potenciar el equipo	80%	86%	**+6%**. Planning Poker para estimaciones colaborativas. Autonomía por módulos con ownership claro. Retrospectivas con acciones específicas y	- Autonomía 80% en módulos - Planning Poker - Retrospectivas accionables - Ownership 100%

			fechas. Pair programming distribuye conocimiento.	
Crear integridad	75%	90%	**+15%**. Pipeline completo con testing automatizado. 0 vulnerabilidades críticas (OWASP ZAP). Headers de seguridad configurados. Rate limiting 30 req/min. Cobertura 82%.	- CI/CD con 3 jobs - OWASP ZAP: 0 críticas - XSS protection (DOMPurify) - Rate limiting implementado
Visualizar el conjunto	65%	82%	**+17%**. C4 Model para documentar arquitectura actualizada. Workshops mensuales de alineación. Miro para mapear dependencias. Impact mapping para features. Technical Debt Ratio 2.8%.	- Diagramas C4 actualizados - Impact mapping - Workshops mensuales - Miro para dependencias

Porcentaje General: 75% (Abril) → 87% (Octubre) | **Mejora: +12 puntos (+16%)**

Reflexión sobre Mejoras Conseguidas

Eliminación de Desperdicios

Desperdicios identificados y reducidos:

1. **Código duplicado:** Factory Pattern eliminó 274 líneas duplicadas a 120 reutilizables (-56%). Duplicación total: 18%→12%.
2. **Deploys manuales:** Pipeline CI/CD redujo tiempo 90% (45min→4.5min) con tasa éxito 96%.
3. **Testing ineficiente:** MSW (Mock Service Worker) redujo tiempo de tests 80% (2min 45s→32s), eliminando costos de API.
4. **Documentación tardía:** README y wiki se actualizan en paralelo al desarrollo. Creamos base de conocimiento en Notion.

Prácticas más efectivas:

- CI/CD automatizado detecta defectos antes de producción (evitó 15 bugs críticos)
- Ambientes staging/production separados (0 incidentes críticos en producción)
- Pair programming para código crítico (detectó 3 casos edge en XSS)
- Retrospectivas con acciones específicas y fechas

Amplificación del Aprendizaje

Métodos implementados:

- **Pair programming:** 2-3 veces/semana en tareas críticas (OAuth, XSS, CI/CD)
- **Code reviews:** 100% PRs revisados, límite sugerido 200 líneas/PR
- **Knowledge sharing:** Sesiones semanales (GitHub Actions, Redis, seguridad)
- **Wiki interna:** Notion con guías de Supabase, Redis, CI/CD, patrones
- **Spikes técnicos:** Investigación antes de implementación (Azure AD, Redis)

Cambio en habilidades:

- **Antes:** Conocimiento siloed por especialización, dependencia de expertos
- **Ahora:** Conocimiento distribuido, bus factor reducido, cualquier miembro mantiene cualquier módulo
- **Individuales:** Osman (DevOps/CI/CD), Milton (Performance/IA), Gadiel (Security/Testing)
- **Equipo:** Colaboración efectiva, resiliencia técnica, mentalidad de calidad (82% cobertura)

Tomar Decisiones lo Más Tarde Posible

Decisiones retrasadas con mejores resultados:

1. **Sistema de caché:** Spike técnico evaluó Redis vs Memcached. Redis ganó por persistencia. Resultado: 72% mejora en Tips IA (2.8s→0.78s).
2. **Plataforma deployment:** POC con Netlify/Vercel/Railway. Vercel ganó por integración GitHub. Deployment: 4.5min.
3. **Arquitectura de tests:** Esperamos casos reales para comparar Jest vs Vitest. Jest+MSW redujo tiempo 80%.

Situaciones difíciles:

- Migración a Supabase tarde (Sprint #2) causó costo alto
- Microsoft OAuth sin evaluar alternativas
- **Mejoras Sprint #10:** Matriz de decisión estructurada, timeboxing para decisiones, reversibilidad por diseño

Entrega lo Antes Posible

Reducción tiempo entre entregas:

- **Antes:** Entregas cada 2 semanas con features completas
- **Ahora:** 28 deploys en Sprint #9 (1.3 deploys/día)
- **Deployment time:** 45min→4.5min (-90%)

Beneficios:

- Feedback más rápido (bugs corregidos en 2 horas vs 2 días)
- Menor riesgo por deploy (deploys pequeños y frecuentes)
- Detección temprana con health checks automáticos
- Confianza del equipo (96% tasa éxito)

Feedback que cambió enfoque:

- Homepage móvil: Fix desplegado en 4 horas
- Tips IA: Transparencia con tooltips y rating en Sprint #9
- Performance: Redis cache implementado, mejora 72% validada en 2 días

Potenciar al Equipo

Participación en decisiones:

- Planning Poker para estimaciones colaborativas
- Ownership de módulos con autonomía técnica total
- Retrospectivas con voz equitativa y acciones consensuadas
- Rotación de roles (liderazgo de retrospectivas y demos)

Cambio en autonomía (Julio→Octubre):

- Antes: Dependencia de expertos, comunicación jerárquica
- Ahora: Autonomía 80%, decisiones técnicas descentralizadas, proactividad

Desafíos:

- Carga desigual en tareas complejas → Pair programming obligatorio
- Comunicación limitada con PO → Demos semanales obligatorias
- Silos de conocimiento → Knowledge sharing y buddy system

Crear Integridad

Mejoras en CI/CD:

Métrica	Antes (Sprint #7)	Después (Sprint #9)	Mejora
Deployment time	45 min	4.5 min	-90%
Tasa éxito	78%	96%	+23%
Bugs en production	12/3 sprints	2/3 sprints	-83%
Cobertura pruebas	68%	82%	+14%
Tiempo tests	2min 45s	32s	-80%

Pipeline CI/CD (GitHub Actions):

- Job 1: Testing (ESLint, Jest, Cypress) - 4.5min
- Job 2: Deploy Staging (branch develop) - 2.2min
- Job 3: Deploy Production (branch main) - 2.5min
- Health checks post-deploy automáticos

Seguridad:

- OWASP ZAP: 0 vulnerabilidades críticas
- XSS protection con DOMPurify
- Headers de seguridad (CSP, HSTS, X-Frame-Options)
- Rate limiting: 30 req/min por IP

Dificultades identificadas:

- Testing solo en emuladores → Conseguir dispositivos físicos Sprint #10
- PRs grandes (450 líneas) → Límite 300 líneas/PR
- Sin monitoreo producción → Implementar Sentry Sprint #10
- Documentación API desactualizada → Swagger/OpenAPI Sprint #10

Visualizar Todo el Conjunto

Cambio en visión (Julio→Octubre):

Aspecto	Antes	Ahora
Decisiones	Aisladas por módulo	Considerando impacto global
Documentación	Desactualizada	C4 Model actualizado
Comprensión	Solo expertos	Todo el equipo
Anticipación	Reactiva	Proactiva en planning

Herramientas implementadas:

- **C4 Model:** 4 niveles de arquitectura (Contexto, Contenedores, Componentes, Código)
- **Impact mapping:** Objetivo→Actor→Impacto→Deliverable para cada feature
- **Workshops mensuales:** 2h de alineación técnica/negocio con equipo+PO
- **Miro boards:** Mapeo de dependencias, cuellos de botella, single points of failure
- **ADRs:** Architecture Decision Records documentan decisiones con contexto

Ejemplo de pensamiento holístico:

- Antes de Redis: Evaluamos impacto en costo, deployment, invalidación, monitoreo
- Identificamos que Tips IA dependía de 3 servicios (Gemini, Supabase, Redis), agregamos circuit breaker

Conclusión

El equipo mejoró de **75% a 87% (+12 puntos)** en aplicación de Lean Software Development.

Logros Principales

Mayores mejoras:

1. **Visualizar conjunto:** +17 puntos (C4 Model, impact mapping, workshops)
2. **Eliminar desperdicios:** +18 puntos (CI/CD -90% tiempo, duplicación -56%)
3. **Crear integridad:** +15 puntos (0 vulnerabilidades críticas, cobertura 82%)

Resultados concretos:

- Deployment: 45min→4.5min (-90%), tasa éxito 96%
- Performance: Tips IA +72% (2.8s→0.78s), Homepage LCP +44%
- Seguridad: 0 vulnerabilidades críticas, XSS protection

- Calidad: Technical Debt Ratio 2.8%, cobertura 82%
- Velocidad: 100% en Sprints #8-9 (de 81.8% en Sprint #2)

Desafíos Pendientes Sprint #10

1. **Monitoreo:** Implementar Sentry para error tracking, New Relic para performance
2. **Testing:** Dispositivos físicos, BrowserStack, 10 tests E2E adicionales
3. **Documentación:** Swagger/OpenAPI para API
4. **Refactorización:** 31 puntos de deuda técnica (Design System, TypeScript, SQL optimization)
5. **Seguridad:** Headers HSTS/Referrer-Policy, ocultar X-Powered-By

Proyección

Con mejoras planificadas, proyectamos alcanzar **90-92%** al final del semestre, priorizando:

- Reducir Technical Debt Ratio a <2.5%
- Monitoreo completo en producción
- Documentación API automatizada
- Testing E2E exhaustivo

La aplicación de Lean ha transformado nuestro proceso de reactivo y fragmentado a proactivo e integrado, logrando entregas frecuentes, alta calidad (0 vulnerabilidades críticas) y valor sostenible al cliente.