ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERIA DE SISTEMAS
(ISWD633) CONSTRUCCIÓN Y EVOLUCIÓN DE SOFTWARE
VICENTE EGUEZ

Estudiantes: Javier Quilumba y Jonathan Tipan

Fecha: 22/11/2025

Taller-08-09-10: SCM en el mundo real

Plataforma: HealthCare.gov

Análisis de Caso: Healthcare.gov (2013)

1. Resumen del Caso

Información General

Aspecto	Detalle
Evento	Lanzamiento de Healthcare.gov
Fecha	1 de octubre de 2013
Propósito	Marketplace de seguros médicos del Affordable Care Act (Obamacare)
Alcance	36 estados de EE.UU., millones de usuarios esperados

El Evento

El 1 de octubre de 2013, Estados Unidos lanzó Healthcare.gov, plataforma para inscripción de seguros médicos del Obamacare. El sistema colapsó en 2 horas: solo 6 personas lograron inscribirse el primer día de 250,000 que intentaron. La crisis duró 2 meses hasta que un "Tiger Team" implementó SCM adecuado y recuperó el sistema en 6 semanas.

Fuentes

[1] US Dept. HHS Inspector General Report (2016): https://oig.hhs.gov/oei/reports/oei-06-14-00350.asp
[2] Harvard Business School Case Study: https://d3.harvard.edu/platform-rctom/submission/the-failed-launch-of-www-healthcare-gov/
[3] Dolfing, H. Technical Analysis (2022): https://www.henricodolfing.com/2022/12/case-study-launch-failure-healthcare-gov.html

El Desastre

Métrica	Esperado	Real	Resultado
Usuarios día 1	50,000	250,000	5x más demanda
Inscripciones día 1	Miles	**6 personas**	99.99% fallo
Tasa de éxito	>90%	<1%	Colapso total
Tiempo de carga	<3 seg	8-71 seg	20x más lento
Costo	$93.7M	$1.7B	+1,714%

Causa raíz: 33 contratistas sin integración continua, código con "Lorem Ipsum" en producción, sin pruebas de carga, ausencia total de SCM centralizado.

Recuperación: Tiger Team de Silicon Valley + metodología Scrum. En 6 semanas: 400 bugs corregidos, capacidad 5x, tiempo 1s. Resultado: 1.2M inscripciones dic vs 26K oct.

2. Clasificación del Mantenimiento

Tipo: Mantenimiento Correctivo (85%)

Sistema con defectos críticos desde día 1 que impedían función básica. Esfuerzo reactivo para corregir sistema roto con deadline legal 23 dic 2013.

Distribución: Correctivo 85% (400 bugs críticos), Perfectivo 10% (optimización 71s→1s), Preventivo 3% (US Digital Service), Adaptativo 2% (ajustes integración).

Justificación: Defectos funcionales severos + bugs producción + naturaleza reactiva + urgencia extrema = Correctivo puro.

3. Procesos SCM Involucrados

3.1 Control de Versiones (VCS)

Problemas Pre-Lanzamiento (Ausencia de VCS Adecuado)

- **Sin integración continua:** 33 contratistas desarrollaban componentes aislados sin integrarlos hasta últimas semanas. No había pipeline CI/CD detectando conflictos tempranamente. Componentes se probaban individualmente pero nunca como sistema integrado.

Impacto SCM: Cuando intentaron integrar, descubrieron incompatibilidades masivas, APIs inconsistentes, dependencias circulares. Demasiado tarde para rediseñar.

- **Confusión de roles:** CGI Federal creía que no era integrador de sistemas, CMS pensaba que sí. Ningún "gatekeeper" revisando merges o verificando impacto de cambios en otros sistemas.

Impacto SCM: Sin ownership claro de branches, cada equipo hacía push sin verificar consecuencias. No había proceso de code review centralizado.

- **Cambios last-minute sin control:** Eliminaron "navegar sin login" última semana septiembre sin ajustar capacidad del sistema. No usaban feature branches ni staging para probar impacto.

Impacto SCM: Cambios críticos entraban directo a producción. Sin rollback posible porque no mantenían versiones estables anteriores ni tags de releases.

- **Código incompleto en producción:** "Lorem Ipsum" en JavaScript evidencia que no usaban feature branches. Código "work in progress" mezclado con código producción.

Impacto SCM: Imposible distinguir código completo de incompleto. Sin releases formales, sin forma de identificar qué era production-ready.

Soluciones Durante Recuperación (VCS Correcto)

Branching strategy clara:

- Main branch: protegido, solo código estable
- Development: integración continua
- Feature branches: cada fix en rama propia (fix/login-bottleneck-247)
- Hotfix branches: correcciones urgentes
- **Commits rastreables:** Cada commit vinculado a ticket ([TICKET-ID] descripción). Commits atómicos (un problema, un commit). Historial legible permitió bisect rápido para identificar bugs.
- **Code reviews obligatorios:** Mínimo 2 revisores antes de merge. Pull requests con descripción, solución, evidencia de testing. Calidad mejoró drásticamente.
- **Integración continua:** Cada push disparaba build, tests unitarios, tests integración, análisis estático. Build "rojo" = no merge. Detección inmediata de código que rompía sistema.
- **Cultura "badgeless":** Todos en mismo repositorio, mismas prácticas. Eliminó silos de conocimiento entre gobierno y contratistas.

3.2 Gestión de Cambios (Change Management)

Problemas Pre-Lanzamiento (Sin Change Management)

- **Sin proceso formal de aprobación:** Cambios arquitectónicos se implementaban sin evaluación de impacto técnico. Decisión de eliminar "navegar sin login" fue política, sin consultar equipo técnico sobre capacidad del sistema.

Impacto SCM: Sin change request forms documentando justificación, sistemas afectados, plan rollback, testing requerido. Cambio fue directo a producción sin análisis de riesgo.

- **Ignorar señales de alerta:** 18 advertencias escritas documentadas. Reporte McKinsey (abril 2013) sobre "alto riesgo" nunca compartido con líderes técnicos. Sin proceso Go/No-Go basado en criterios objetivos.

Impacto SCM: En SCM maduro existen "quality gates" (cobertura tests >80%, performance tests passed, security audit). Healthcare.gov no tenía ninguno. Lanzamiento por mandato legal, no porque sistema estuviera listo.

Soluciones Durante Recuperación (Change Management Correcto)

- **Scrums diarios obligatorios:** Revisión progreso, identificar bloqueos, ajustar prioridades. Jeffrey Zients autoridad final sobre todos los cambios significativos.
- **Proceso Go/No-Go basado en métricas:** Antes de cada deployment: ¿tests pasaron? ¿performance aceptable? ¿aprobación security? ¿plan rollback listo?

- **Testing riguroso pre-deployment:** Cada fix pasaba staging → QA → load testing → security review → producción. Tests de regresión aseguraban que fixes no rompían otras funcionalidades.

3.3 Gestión de Configuración de Ambiente

Problemas Detectados

Configuración inconsistente: Versiones mixtas vSphere (v4.1 y v5.1), 48 VMs en 12 servidores con configs diferentes. Base de datos MarkLogic con middleware auto-generado que no escalaba. Infraestructura al 25% capacidad (1 Gb/s de 4 Gb/s).

Impacto SCM: Sin "infrastructure as code" (IaC). Cada servidor configurado manualmente = configuraciones drifteadas. Imposible replicar ambiente producción en staging para testing.

Soluciones Implementadas

Estandarización: Todos los servidores migrados a misma versión vSphere. Configuraciones documentadas y replicables. Optimización red a capacidad completa. CDN (Akamai) para contenido estático reduciendo carga servidores principales.

Resultado: Ambientes predecibles. Testing en staging reflejaba comportamiento real de producción.

4. Impacto en el SDLC

Análisis por Fase

Fase	Pre-Lanzamiento	Recuperación	Impacto
Planning	Sin líder claro, 33 contratistas, timeline 7 meses irreal	Jeffrey Zients líder, sprints 2 semanas	Crítico
Requirements	Cambios constantes, scope creep sin evaluación	Features congelados, priorización core	Crítico
Design	55 equipos aislados, sin arquitecto, decisión fatal login	Rediseño coordinado de flujos críticos	Crítico
Development	Lorem Ipsum en prod, sin estándares código	Code reviews + pair programming	Alto
Testing	Meses→semanas, sin pruebas carga, sin end-to-end	Testing continuo, pruebas incrementales	**MÁS CRÍTICO**
Deployment	Big Bang sin rollback, fecha política inamovible	Incremental, staging primero	Crítico
Maintenance	Fase dominante 2 meses (crisis 24/7)	400 defectos corregidos organizadamente	Crítico

Testing: La Fase Más Comprometida

El testing fue la fase más afectada y la causa directa del colapso. Plan original contemplaba meses, se ejecutó en semanas. Nunca se hicieron pruebas end-to-end del sistema completo ni pruebas de carga con 250,000 usuarios. QSSI (contratista de testing) encontró problemas críticos pero no pudo detener lanzamiento por mandato legal.

Consecuencia: Sistema lanzado sin saber si funcionaba bajo carga real. Resultado: <1% tasa de éxito.

Lección SDLC: Comprimir testing para "ahorrar" 6 meses resultó en $1.6B en sobrecostos + 2 meses inoperatividad + daño reputacional masivo. ROI negativo: ~$267M perdidos por cada mes "ahorrado".

5. Beneficios del SCM

La Prueba Definitiva: Mismo Sistema, Diferente SCM

Métrica	Oct 2013 (Sin SCM)	Dic 2013 (Con SCM)	Mejora
Inscripciones/mes	26,000	975,000	**+3,650%**
Tasa de éxito	<1%	>90%	**+8,900%**
Tiempo de carga	71 seg	1 seg	**-98%**
Usuarios concurrentes	~5,000	25,000	**+400%**
Defectos conocidos	Sin tracking	400 identificados y corregidos	Visibilidad total

1. Trazabilidad Total: Cada commit rastreado a ticket, autor, fecha, razón. Tiger Team usó git blame y git bisect para encontrar 400 bugs rápidamente. Auditoría completa de decisiones técnicas.
2. Control de Calidad: Code reviews obligatorios (mínimo 2 revisores). CI/CD bloqueaba merges si tests fallaban. Solo código completo y revisado llegaba a producción.
3. Gestión de Riesgos: Sistema de tracking por criticidad (P0-P4). Proceso Go/No-Go basado en métricas objetivas. Visibilidad total de riesgos a todos niveles.
4. Colaboración Efectiva: Cultura "badgeless" (un solo equipo). Repositorio centralizado con visibilidad completa. Pull requests = mecanismo de comunicación técnica. Pair programming = transferencia conocimiento.
5. Velocidad de Recuperació: 6 semanas para corregir 400 defectos organizadamente. Rollback rápido de cambios problemáticos. Desarrollo paralelo sin conflictos usando branches.

Costo de Ignorar SCM

Estimación conservadora: Con SCM desde inicio = $200-400M. Real: $1.7B. **Ahorro potencial: $1.3-1.5B.**

Conclusiones

1. **SCM no es opcional:** Implementarlo en crisis cuesta 18x más ($1.7B vs $93.7M)
2. **Testing no es negociable:** Comprimirlo = <1% tasa éxito, sistema inutilizable
3. **Big Bang Deployment = Alto riesgo:** Sin rollback plan = crisis nacional
4. **Integración continua esencial:** 33 equipos sin CI = incompatibilidades masivas
5. **Liderazgo técnico crítico:** Sin integrador sistemas claro = confusión total
6. **SCM acelera recuperación:** Con SCM, Tiger Team arregló en 6 semanas lo que tomó 2 meses sin SCM

Reflexión Final

Healthcare.gov demuestra que el SCM no es burocracia, es la columna vertebral del desarrollo de software moderno.

Fracaso:

```
Planning deficiente + Testing comprimido + Big Bang Deploy
+ Sin SCM + Deadline político = $1.7B perdidos
```

Éxito:

```
Líder técnico fuerte + SCM riguroso + CI/CD + Testing continuo
+ Deploys incrementales = Sistema funcional en 6 semanas
```

Referencias

[1] U.S. Department of Health and Human Services, Office of Inspector General. *"HealthCare.gov: CMS Management of the Federal Marketplace - A Case Study"* (2016)

[2] Harvard Business School - Technology and Operations Management. *"The Failed Launch Of www.HealthCare.gov"* (2016)

[3] Dolfing, H. *"Case Study: The Disastrous Launch of Healthcare.gov"* (2022)