Documento de Control de Configuración (CM)

Web Scraper para Plataforma X

Estándar: IEEE 828-2012

Versión: 1.0

Fecha: 14 de mayo de 2025

Autores

- 👩‍💻 **Leonardo Valdés Palafox**
- 👩‍💻 **Andrea Marlene Ortega Almendares**

Control de Documento

Versión	Fecha	Autor(es)	Descripción de Cambios
0.1	10/05/2025	Leonardo Valdés	Versión inicial del documento
0.2	12/05/2025	Andrea Ortega	Revisión y adiciones de política de ramas
1.0	14/05/2025	Leonardo Valdés, Andrea Ortega	Versión final aprobada

---

Índice

---

1. Introducción

1.1 Propósito

Este documento describe el plan de Control de Configuración para el desarrollo y mantenimiento del proyecto "Web Scraper para plataforma X", una herramienta desarrollada en Python y Selenium diseñada para extraer tweets, métricas y metadatos de la plataforma X, almacenándolos en formato CSV para su posterior análisis. El objetivo de este plan es establecer y mantener la integridad de los productos del proyecto a lo largo de su ciclo de vida.

1.2 Alcance

Este plan de CM aplica a todos los elementos de configuración del proyecto "Web Scraper para plataforma X", incluyendo:

- Código fuente
- Documentación
- Pruebas
- Scripts de automatización
- Configuraciones de entorno
- Datos de prueba

El plan establece las políticas, procedimientos, roles y responsabilidades para asegurar la integridad y trazabilidad de los cambios en el proyecto, desde su inicio hasta su liberación y mantenimiento.

1.3 Definiciones y Acrónimos

Término	Descripción
CM	Control de Configuración (Configuration Management)
CI	Integración Continua (Continuous Integration)
EC	Elemento de Configuración
PR	Solicitud de Integración (Pull Request)
CCB	Consejo de Control de Cambios (Change Control Board)
SRS	Especificación de Requisitos de Software
API	Interfaz de Programación de Aplicaciones
Web Scraping	Técnica de extracción automatizada de información de sitios web

1.4 Referencias

- IEEE 828-2012 - Estándar para Planes de Control de Configuración de Software
- Documentación de Git y GitHub
- Matriz de Productos Documentales del Proyecto
- Especificación de Proyecto: Sistema de Análisis de Información para Restaurantes Estandarizado (IEEE 830-1998)
- Metodología SCRUM implementada en el proyecto

2. Gestión de Control de Configuración

2.1 Organización y Responsabilidades

El proceso de Control de Configuración será supervisado por el equipo de desarrollo conforme a la estructura organizativa de SCRUM establecida en la especificación del proyecto. La integración entre los roles SCRUM y los roles CM se detalla a continuación.

2.2 Roles CM

Rol	Responsabilidades	Asignado a
CM Manager	Supervisión general del proceso CM, resolución de conflictos, aprobación final de cambios importantes	Product Owner
CM Administrator	Configuración y mantenimiento de herramientas CM, garantizar integridad del repositorio	Scrum Master
Desarrolladores CM	Creación y entrega de cambios siguiendo los procedimientos CM	Equipo de Desarrollo
Auditor CM	Verificación de cumplimiento de procesos CM	Scrum Master y Product Owner

2.3 Herramientas CM y Entorno

Herramienta	Propósito	Versión
Git	Sistema de control de versiones distribuido	2.42.0
GitHub	Plataforma de hospedaje de código y colaboración	N/A
GitHub Actions	Integración continua y automatización	N/A
Jira	Gestión de proyectos y seguimiento de problemas	Última disponible
Planning Poker for Jira	Estimación de tareas	Última disponible

3. Actividades de CM

3.1 Identificación de Configuración

3.1.1 Elementos de Configuración

Los siguientes elementos serán identificados y gestionados como Elementos de Configuración (EC):

Tipo de EC	Descripción	Convención de Nomenclatura
Código Fuente	Archivos Python, scripts de Selenium	`[módulo]_[funcionalidad].py`
Documentación	Especificaciones, manuales, guías	`[tipo]_[descripción].md`
Datos de Configuración	Archivos de configuración	`config_[entorno].yaml`
Tests	Pruebas unitarias e integración	`test_[funcionalidad].py`
Datos de Prueba	Conjuntos de datos para testing	`sample_[tipo]_data.csv`
Builds	Paquetes compilados	`scraper-x-[versión].tar.gz`

3.1.2 Estructura de Versionado

El proyecto utilizará Versionado Semántico (SemVer) con el formato X.Y.Z:

- **X**: Versión mayor (cambios incompatibles con versiones anteriores)
- **Y**: Versión menor (nuevas funcionalidades manteniendo compatibilidad)
- **Z**: Versión de parche (correcciones de errores manteniendo compatibilidad)

Prefijos adicionales para versiones en desarrollo:

- **alpha**: Primeras etapas de desarrollo (inestable)
- **beta**: Funcionalidad completa pero en pruebas
- **rc**: Candidato a liberación

Ejemplo: `1.2.0-beta.1`

3.2 Control de Cambios

3.2.1 Solicitud de Cambios

Las solicitudes de cambios se gestionarán a través de issues en GitHub y se integrarán con las historias de usuario en Jira. Cada solicitud incluirá:

- Título descriptivo
- Descripción detallada
- Justificación del cambio
- Prioridad y severidad
- Referencias a requisitos relacionados
- Estimación inicial (puntos de Planning Poker)

3.2.2 Evaluación de Cambios

Los cambios serán evaluados durante las reuniones de Sprint Planning, utilizando Planning Poker para su estimación. Los criterios de evaluación incluirán:

- Impacto en la funcionalidad existente
- Alineación con los objetivos del proyecto
- Esfuerzo requerido (estimado en puntos)
- Riesgos asociados
- Dependencias con otros cambios

3.2.3 Aprobación de Cambios

La aprobación seguirá un proceso basado en la complejidad del cambio:

- **Cambios menores (1-3 puntos)**: Aprobación del Scrum Master
- **Cambios moderados (5-8 puntos)**: Aprobación del Product Owner
- **Cambios complejos (13+ puntos)**: Revisión y aprobación por CCB (Product Owner + Scrum Master + Representante técnico)

3.3 Registro y Reporte de Estado

3.3.1 Registro de Estado

El estado de los elementos de configuración se mantendrá actualizado en:

- GitHub para el código fuente (mediante commits, branches, tags)
- Jira para el seguimiento de tareas e issues
- Documentación de estado generada automáticamente tras cada sprint

3.3.2 Informes de Estado CM

Se generarán los siguientes informes:

Informe	Frecuencia	Audiencia	Contenido
Resumen de Cambios	Final de cada Sprint	Equipo y Stakeholders	Cambios implementados, pruebas realizadas
Informe de Issues	Semanal	Equipo de Desarrollo	Issues abiertos, cerrados, en progreso
Informe de Calidad	Bi-semanal	Product Owner, Scrum Master	Métricas de código, cobertura de pruebas
Estado de Integración	Diario	Equipo de Desarrollo	Resultados de CI/CD, builds fallidos

3.4 Auditoría y Revisión

3.4.1 Auditorías CM

Se realizarán auditorías formales para verificar:

- Cumplimiento de procedimientos CM
- Integridad de la línea base
- Trazabilidad entre requisitos y elementos implementados

Las auditorías se realizarán:

- Al final de cada release mayor

- Antes de despliegues en producción
- Cuando se identifiquen anomalías en el proceso

3.4.2 Revisiones Técnicas

Las revisiones técnicas son parte integral del proceso de Pull Request y seguirán estos criterios:

- Cumplimiento con estándares de codificación
- Adecuación de tests
- Documentación apropiada
- Rendimiento y escalabilidad
- Seguridad

4. Planificación de Entregas

4.1 Estrategia de Versionado

4.1.1 Líneas Base

Las líneas base se establecerán:

- Al final de cada sprint (línea base interna)
- Al final de cada release (línea base externa)
- En puntos específicos del roadmap (milestone releases)

4.1.2 Etiquetado de Versiones

Las versiones se etiquetarán en Git con el prefijo "v" seguido del número de versión:

```
v1.0.0
v1.1.0
v1.1.1
```

4.2 Ciclo de Vida de Versiones

Las versiones seguirán el siguiente ciclo de vida:

1. **Desarrollo**: Implementación activa en ramas de características
2. **Integración**: Merge a `develop` y pruebas de integración
3. **Release Candidate**: Creación de rama `release/vX.Y.Z` para pruebas finales
4. **Producción**: Merge a `main` y etiquetado

5. **Mantenimiento**: Corrección de bugs en ramas `hotfix`
6. **EOL (End of Life)**: Versiones descontinuadas

5. Procedimientos CM

5.1 Control de Código Fuente

Todo el código fuente será gestionado en GitHub siguiendo estas pautas:

- Los commits deben ser atómicos (una sola funcionalidad o corrección)
- Los mensajes de commit seguirán el formato:

```
[tipo]: Descripción corta (máx. 50 caracteres)

Descripción detallada (opcional, separada por línea en blanco)

Referencia a issues (#número)
```

- Tipos de commit:
  - `feat`: Nueva funcionalidad
  - `fix`: Corrección de error
  - `docs`: Cambios en documentación
  - `style`: Cambios de formato, estilo
  - `refactor`: Refactorización del código
  - `test`: Adición o modificación de tests
  - `chore`: Cambios en el proceso de build, herramientas, etc.

5.2 Política de Ramas

Se implementará un modelo de branching basado en GitFlow adaptado a SCRUM:

5.2.1 Ramas Principales

- **`main`**: Código en producción
- **`develop`**: Código integrado para la próxima entrega

5.2.2 Ramas de Soporte

- **`feature/[id-jira]-descripción`**: Nuevas funcionalidades
- **`bugfix/[id-jira]-descripción`**: Correcciones de errores

- `release/vX.Y.Z`: Preparación para release
- `hotfix/[id-jira]-descripción`: Correcciones urgentes en producción
- `docs/[descripción]`: Actualización de documentación

5.2.3 Flujo de Trabajo

1. Desarrollo en ramas `feature` o `bugfix` a partir de `develop`
2. Pull Request a `develop` cuando la funcionalidad está completa
3. Creación de rama `release` desde `develop` para preparar versión
4. Después de pruebas en `release`, merge a `main` y `develop`
5. Etiquetado en `main` con la versión

5.3 Proceso de Solicitud de Cambios

5.3.1 Pull Requests

Cada Pull Request deberá:

1. Referenciar el issue o historia de usuario correspondiente
2. Incluir una descripción clara de los cambios realizados
3. Superar los checks automáticos (linting, tests, build)
4. Ser revisado por al menos un miembro del equipo
5. Recibir aprobación explícita antes del merge

5.3.2 Criterios de Revisión

- Cumplimiento de estándares de código
- Presencia de tests adecuados
- Documentación actualizada
- No regresiones en funcionalidad existente
- Rendimiento aceptable

5.4 Creación de Builds

5.4.1 Entornos de Build

Entorno	Propósito	Trigger
Desarrollo	Testing local	Manual
Integración	Pruebas de integración	Automático (en merge a `develop`)
Staging	UAT, pruebas de rendimiento	Automático (en merge a `release`)
Producción	Entorno real	Manual (después de aprobación)

5.4.2 Proceso de Build

1. Checkout del código desde el repositorio Git
2. Instalación de dependencias desde `requirements.txt`
3. Ejecución de linters y analyzers
4. Ejecución de tests unitarios
5. Ejecución de tests de integración
6. Empaquetado del código
7. Etiquetado del build con metadatos

5.5 Despliegue

5.5.1 Proceso de Despliegue

1. Selección del build verificado
2. Aprobación por el Product Owner
3. Despliegue en el entorno target
4. Verificación post-despliegue
5. Notificación a stakeholders

5.5.2 Rollback

En caso de fallo durante el despliegue:

1. Activación inmediata del plan de rollback
2. Restauración de la versión anterior estable
3. Notificación al equipo de desarrollo
4. Análisis de causa raíz
5. Documentación del incidente

6. Métricas CM

Las siguientes métricas se recopilarán para evaluar la efectividad del proceso CM:

Métrica	Descripción	Meta	Frecuencia
Tiempo de Integración	Tiempo desde PR hasta merge	< 1 día	Por PR
Tasa de Éxito de Build	% de builds exitosos	> 95%	Diario
Tiempo de Resolución de Defectos	Tiempo promedio para resolver issues	< 3 días	Semanal
Cobertura de Código	% de código cubierto por tests	> 80%	Por build
Deuda Técnica	Medida por herramientas de análisis estático	Decreciente	Semanal
Defectos por Release	Número de defectos encontrados post-release	< 5	Por release

7. Apéndices

7.1 Estructura del Repositorio

```
/
├── .github/
│   ├── workflows/         # GitHub Actions CI/CD
│   └── PULL_REQUEST_TEMPLATE.md
├── docs/
│   ├── api/               # Documentación de API
│   ├── user/              # Guías de usuario
│   └── development/       # Guías para desarrolladores
├── src/
│   ├── scraper/           # Módulo principal de scraping
│   ├── processors/        # Procesamiento de datos
│   ├── analytics/         # Análisis de datos
│   └── utils/             # Utilidades comunes
├── tests/
│   ├── unit/              # Tests unitarios
│   ├── integration/       # Tests de integración
│   └── fixtures/          # Datos de prueba
├── scripts/               # Scripts de automatización
├── .gitignore
├── LICENSE
├── README.md
├── requirements.txt       # Dependencias de producción
└── requirements-dev.txt   # Dependencias de desarrollo
```

7.2 Plantillas

7.2.1 Plantilla de Issue

markdown

```
## Descripción
[Descripción clara y concisa del issue]

## Comportamiento Esperado
[Descripción de lo que debería ocurrir]

## Comportamiento Actual
[Descripción de lo que está ocurriendo actualmente]

## Pasos para Reproducir
1. [Primer paso]
2. [Segundo paso]
3. [Y así sucesivamente...]

## Contexto Adicional
[Cualquier otra información que pueda ser relevante]

## Criterios de Aceptación
- [ ] [Criterio 1]
- [ ] [Criterio 2]
- [ ] [Criterio 3]
```

7.2.2 Plantilla de Pull Request

```markdown
## Descripción
[Descripción clara de los cambios y su propósito]

## Issue Relacionado
Closes #[número-de-issue]

## Tipo de Cambio
- [ ] Funcionalidad nueva
- [ ] Corrección de error
- [ ] Mejora de rendimiento
- [ ] Refactorización
- [ ] Actualización de documentación
- [ ] Otro (especificar)

## Checklist
- [ ] He añadido tests que prueban la funcionalidad
- [ ] Los tests existentes siguen pasando
- [ ] He actualizado la documentación correspondiente
- [ ] He seguido las convenciones de código del proyecto
- [ ] He realizado una auto-revisión de mi código

## Screenshots (si aplica)
[Capturas de pantalla que muestran los cambios]
```

Documento Aprobado por:

Leonardo Valdés Palafox - *Fecha: 14/05/2025*
Andrea Marlene Ortega Almendares - *Fecha: 14/05/2025*

Repositorio del Proyecto:
https://github.com/PrivateerDev/PrivateerDev-Web-Scraper-para-plataforma-X