

# Ingeniería de Software

## Trabajo Práctico 1

### Plan de Gestión de Configuraciones v1.1.1

Alumnos:  
Blanco, Lucas  
Murillo, Sebastian

Docente:  
Miceli, Martin

FCEFYN – UNC  
2019

## Introducción:

El presente documento cubre el plan de configuration management para la implementación de un simulador de poblaciones de vida salvaje, especificando las herramientas, el esquema de directorios, el sistema de nombramiento y versionado de archivos, el sistema de branching, las políticas de testing, release y gestión de defectos a utilizar. Así también como la gestión de cambios y la definición del rol de cada integrante del grupo.

### Referencias y acrónimos utilizados

IDE	integrated development enviroment
UML	unified modeling language
CCB	change control board
SPVS	Simulador de Poblaciones de Vida Salvaje
PM	Proyect Manager

## Configuration Management Tools:

A continuación se especifican las herramientas de software a utilizar durante el desarrollo del proyecto:

Herramienta	Descripción
git	sistema de versionado
Travis CI	sistema de desarrollo por integración continua
ant	sistema de build
eclipse	IDE
Java	lenguaje principal
Star UML	creación de diagramas UML

### Sistema de Versionado

El sistema de versionado a utilizar sera github, mas específicamente <https://github.com/seba-murillo/SPVS>. La aceptación de los commits por parte de los desarrolladores estará a cargo del PM.

### Herramienta de Integración Continua:

La herramienta a utilizar sera Travis CI, su uso estará restringido al PM, quien tendrá la responsabilidad de integrar los distritos commits y posee la palabra final sobre la aceptación de los mismos.

Todo código a ser integrado debe ser exitoso en el %100 de los tests **antes** de ser integrado.

La herramienta esta ubicada en <https://travis-ci.org/seba-murillo/SPVS>.

### Herramienta de Gestión de Defectos:

Los defectos del software serán administrados por el sistema de 'issues' incluido en github (<https://github.com/seba-murillo/SPVS/issues>). Los desarrolladores asignaran una prioridad a cada defecto enviado por este medio, y serán resueltos en las próximas integraciones, dependiendo de la gravedad del defecto.

### Nombramiento de Archivos:

Los archivos deberán tener el siguiente formato:

**[filename]\_X.Y.Z.ext**

filename: nombre de archivo, este debe ser intuitivo, describiendo en una o dos palabras las funciones del código. en el mismo.

- X.Y.Z: versión del archivo donde:
  - X - versión del release
  - Y - versión del patch
  - Z - subversión del patch
- ext: extension del archivo

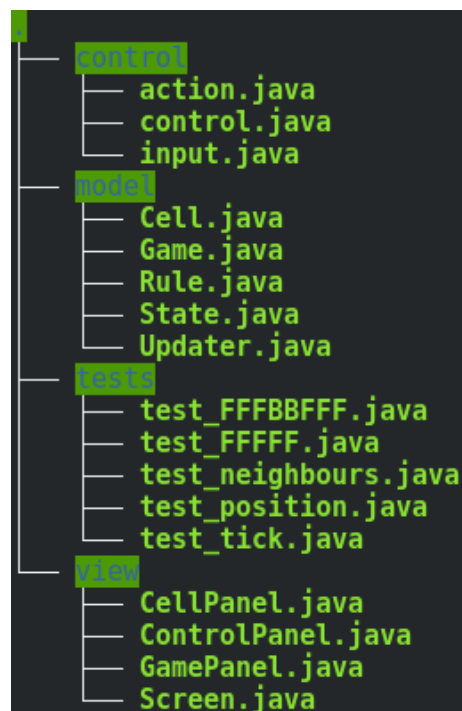
Todos los archivos deberán estar contenido en su respectivo directorio descrito en el esquema de directorios

### Esquema de Directorios:

Se utilizaran 4 directorios principales:

- model: se incluirá todo el código. fuente relacionado con la lógica del sistema.
- view: se incluirá todo el código. fuente relacionado con vistas, pantalla y GUI.
- tests: se incluirá todo el código. fuente relacionado con los tests de todo el código. del software
- controller: e incluirá todo el código. fuente relacionado a el input del usuario.

ejemplo:



Los archivos que no cumplan ninguna de las características anteriores simplemente estarán contenidos en el directorio principal del proyecto.

### Branching:

Se empleara el sistema de branching GitHubFlow (<https://guides.github.com/introduction/flow/>), en el cual solo se incluirán versiones prontas para release en la rama 'master'. Para añadir nuevas características y/o tests al software se deberán crear branches para cada desarrollador y cada implementación después de obtener la aprobación del PM.

## Políticas de Testing:

Los tests unitarios deberán ser implementados por los desarrolladores de sus respectivos módulos. Estos test deberán tener una cobertura del %50 **como mínimo** (recomendado +%90).

Los tests deberán ser implementados con la intencionalidad de hacer fallar el software en **TODOS** los casos de uso del modulo en particular.

Los test de integración y de sistema estarán a cargo del PM, estos deberán chequear la consistencia de los módulos entre si. En la eventualidad de que, al incluir un nuevo test de integración y/o sistema, falle alguno de los módulos ya integrado, el desarrollador de dicho modulo sera el responsable por adaptar el modulo a los nuevos test. Esta ultima tarea deberá ser de máxima prioridad para el desarrollador para así facilitar la integración continua de nuevo código.

## Políticas de Construcción:

Se utilizara ant, por su simpleza.

## Gestión de Cambios:

Cambios en el software pueden ser sugeridos tanto por el cliente como por sus desarrolladores. Será responsabilidad de los desarrolladores en conjunto decidir de aplicar o no cambios al software teniendo en cuenta los costos, beneficios y efectos del cambio en cuestión.

## Gestión de Entrega:

Considerando el soport multiplataforma de Java, la entrega consistirá en un ejecutable .jar (ejemplpo: SPVS\_1.0.0.jar), junto a una instrucción personal breve (5 min) de su uso. También se proveerá al cliente con acceso al sistema de defectos 'issues', en caso de existir errores en el software.

## Historial del Documento:

### v1.0.0

- version inicial

### v1.1.0

- añadida gestion de entrega
- editada gestión de defectos
- añadido control de cambios
- cambio de la herramienta de integración continua
  - (jenkins → travis CI)
- leves cambios de formato al documento

### v1.1.1

- se cambio gradle por ant