|  |  |
| --- | --- |
|  | Unidad de Gestión e Innovación Tecnológica  Sede Interuniversitaria de Alajuela |

API Público: Versionamiento Semántico

UGIT.API.VS-1.0.0

**DOCUMENTO:**

|  |  |
| --- | --- |
| Título: | API Público: Versionamiento Semántico |
| Código: | UGIT.API.VS-1.0.0 |
| Fecha: | 19 de julio 2019 |
| Versión: | 1.0.0 |
| Ubicación: | https://ugit.siua.ac.cr/?page\_id=756 |

**REGISTRO DE CAMBIOS:**

|  |  |  |
| --- | --- | --- |
| Versión | Fecha | Motivo de Cambios |
| 1.0.0 | 19.07.2019 | Primera versión entregada la Administración CONARE para revisión |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**APROBACIÓN DEL DOCUMENTO:**

|  |  |  |
| --- | --- | --- |
| Nombre | Universidad | Firma |
| Noelia Soto González | CONARE |  |
|  | UCR |  |
|  | TEC |  |
|  | UNA |  |
|  | UNED |  |
|  | UTN |  |
| Gustavo Matamoros González | UGIT |  |

**DOCUMENTO GENERADO:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ELABORADO: | **MATAMOROS GONZÁLEZ GUSTAVO** | **MORA ROJAS RAQUEL** | **MONGE MENA JUAN JOSÉ** | **SOLÍS SOLORZANO RAMSES** |
| Unidad de Gestión e Innovación Tecnológica |  |  |  |  |

**Índice**

# **Terminología usa en este documento**

* **TICs:** Tecnologías de Información y Comunicación
* **SIUA:** Sede Interuniversitaria de Alajuela
* **UGIT:** Unidad de Gestión e Innovación Tecnología
* **UGIT.API.VS :** UGIT.API Versionamiento Semántico
* **TI:** Tecnologías de Información
* **CONARE:** Consejo Nacional de Rectores

# **Objetivo:**

Establecer un marco de control de sobre el versionamiento semántico de aplicaciones desarrolladas o implementadas por el personal de la Unidad de Gestión e Innovación Tecnológica (UGIT) o personal externo dentro de la Sede Interuniversitaria de Alajuela.

# **Alcance**

Este documento es aplicable y de conocimiento de todo el personal de la Unidad de Gestión e Innovación Tecnología (de ahora en adelante mencionadas en este documento como UGIT) de la Sede Interuniversitaria de Alajuela (de ahora en adelante mencionadas en este documento como SIUA) y todo el personal interno y externo que desarrolle o implemente una aplicación tecnológica dentro de la SIUA.

# **Responsabilidades**

El cumplimiento de este documento es de carácter obligatorio para todo el personal de la UGIT, así como para usuarios internos y externos que desarrollen o implementen una aplicación tecnológica dentro de la SIUA y que sea mantenida por el personal de la UGIT.

# **Principios Generales**

## **Presentación**

La presente API pública de Versionamiento Semántico (de ahora en adelante conocida como UGIT.API.VS) son un conjunto de especificaciones de versionamiento semántico que establecen el estándar oficial para el versionamiento de sistemas desarrollados o implementados en la SIUA y que el departamento de la UGIT será la encargada de darle soporte a dicha aplicación.

# **Estructura y Contenido**

El presente documento está organizado siguiendo la siguiente estructura:

1. Capítulo I: Introducción
2. Capítulo II: Especificación de Versionamiento Semántico UGIT-SIUA

# **Capítulo I: Introducción**

## **¿Por qué utilizar Versionamiento Semántico?**

Cuando decides utilizar un código que no es tuyo como puede ser una framework o una librería, una de las cosas que debes tomar en cuenta es utilizar la última versión de mismo, no solo porque está puede incluir nuevas funcionalidades que te puedan interesar, sino también porque puede que esta última versión en la cual, acabas de desarrollar todo tu código no sea compatible con versiones anteriores, por tanto, aquí nacen dos problemas del mundo del software. Uno, para el consumidor de ese framework o librería es importante conocer si esta última versión modifica totalmente el comportamiento y funcionamiento del mismo o solo arregla pequeños errores u agrega nuevas funcionalidades. Y dos, para el creador del framework o librería necesita de una manera de comunicar a sus usuarios la naturaleza y alcance de la última versión de su código. Por esta razón nace lo que hoy día se conoce como "Versionamiento Semántico (Semantic Versioning)".

## **Tipo de versiones**

El versionamiento semántico es un convenio o estándar a la hora de definir la versión de tu código, dependiendo de la naturaleza del cambio o cambios que estás introduciendo. De tal forma, se identifican 3 tipos de cambios:

* **Mayor:** Cambio drástico en el código. No es compatible con el código de versiones anteriores.
* **Menor:** Cambio que añade alguna o algunas características nuevas al código o modifica alguna o varias ya existentes, pero que sigue siendo compatible con código existente. También cuando marcamos algo como obsoleto que no modifique el funcionamiento principal.
* **Revisión:** Cuando arreglamos un error siendo el cambio compatible con la versión actual o un aspecto de diseño.

De esta forma, tenemos un lenguaje común entre desarrolladores y consumidores a la hora de hablar de versiones.

## **¿Cómo se marca una versión como mayor, menor o revisión?**

Cada vez que enviemos código al repositorio crearemos un nuevo **“tag”** siguiendo el convenio semántico, dependiendo de los cambios introducidos. El tag contendrá la información, separando las versiones de cada tipo por puntos, de la forma: **“mayor.menor.revision** y este no puede ser cambiado jamás, para que si alguien depende de esa versión en particular, pueda seguir haciéndolo sin problemas.

Si el framework o librería utiliza el versionamiento semántico de forma correcta, cualquier usuario que lo utilice puede actualizar sin peligro a todas las versiones marcadas como **revisión** o **menor** porque no romperán su código. Por tanto, podría actualizar de la versión 5.3.3 a la 5.3.4 o incluso 5.4.0, porque solo estarían incrementando las versiones **menor** y **revisión**, que son compatibles con código existente.

Si por el contrario pasa de la versión 1.4 a la versión 2.0, tendría que tener cuidado porque seguramente habría muchas cosas que dejarían de funcionar ya que se ha cambiado la **mayor**.

## **Identificadores de estabilidad**

Además de poder definir los cambios en tu código como **mayor**, **menor** o **revisión**, existen además unos identificadores que ayudan a marcar versiones especificas que quieras diferenciar, ya que se utilizan para indicar la “estabilidad” de esa versión.

Por ejemplo, si tenemos la versión 1.4.6 de una aplicación y decides comenzar el desarrollo de la versión 2.0.0 (incluye cambios no compatibles con las versiones anteriores), pero apenas estas comenzando, por tanto; debes indicarles a tus usuarios la estabilidad de tu aplicación para que sepan que todavía estas en la etapa de desarrollo para esto se utilizan estos identificadores:

* **alpha:** se encuentra en etapa de desarrollo no es estable. Ejemplo: 2.0.0-alpha
* **beta:** se encuentra como una posible candidata a ser liberada. Ejemplo: 2.0.0-beta
* **rc (Release Candidate):** candidata a liberación el cual lleva un número de versión, mientras es liberada. Ejemplo 2.0.0-rc2 (versión 2).

## **Opciones adicionales**

Además, en de todo lo que se ha mencionado en el desarrollo de software de grandes aplicaciones como video juegos y sistemas operativos es importante considerar el en versionamiento, los parches o fechas de lanzamientos, dependiendo del caso. Para estos podemos agregar un número más el versionamiento, por ejemplo:

#### Versión de parche:

Si considera importante el manejo y control de los parches puede agregar un número adicional a la estructura pasando de X.Y.Z a X.Y.Z.P donde “P” sería el número de parche, además puede utilizar una letra “p” para indicar que es un parche. Por ejemplo: 1.2.5.2 / 02.03.03.01 / 1.25.4.p2

#### Versión por fecha

Si considera importante el manejo y control de las fechas de liberación de su aplicación podemos adicionar la fecha al versionamiento. En este caso el formato y el orden puede variar dependiendo de las necesidades, por ejemplo puede querer utilizar los cuatro dígitos del año (2019, 2010, …) o solo dos dígitos (19, 10, …), puede querer utilizar el año de primero, luego el mes y día ó día, mes y año, pero todo va depender de sus necesidades, como adicional el desarrollo de un software siempre requiere bastante tiempo invertido por lo que si es necesario utilizar las fecha dentro del versionamiento le recomendamos utilizar el año de primero.

# **Capítulo II: Especificación de Versionamiento Semántico UGIT-SIUA**

1. Cualquier software desarrollado o implementado dentro de la SIUA, por el personal de la UGIT, personal interno o externo a la SIUA debe señalar el uso de este documento como estándar oficial de versionamiento a través de los mecanismos:
   1. Enlace público https://ugit.siua.ac.cr/?page\_id=756
   2. O plasmarlo directamente dentro del código
2. El versionamiento de una versión de software será identificada por tres dígitos enteros no negativos, siguiendo el patrón X.Y.Z, donde:

|  |  |  |  |
| --- | --- | --- | --- |
| Clave | Nombre | ¿Cuándo? | Ejemplos |
| X | **Mayor:** representa un cambio grande en el software. | La versión mayor X (X.y.z | X > 0) DEBE ser incrementada si cualquier cambio no compatible con la versión anterior es introducido a la aplicación. PUEDE incluir cambios de nivel menor y/o revisión. Las versiones revisión y menor DEBEN ser reseteadas a 0 cuando se incrementa la versión mayor. | **i.** Se eliminan funcionalidades.  **ii.** Se modifica el funcionamiento de funciones que hacen incompatibles con las versiones anteriores.  **iii.** Se marca como obsoleto funcionalidades principales. |
| Y | **Menor:** representa un cambio menor compatible con versiones anteriores | La versión menor Y (x.Y.z | x > 0) DEBE ser incrementada si se introduce una nueva funcionalidad compatible con la versión anterior. Se DEBE incrementar si cualquier funcionalidad de la aplicación es marcada como obsoleta. PUEDE ser incrementada si se agrega funcionalidad o arreglos considerables al código. Puede incluir cambios de nivel revisión. La versión revisión DEBE ser reseteada a 0 cuando la versión menor es incrementada. | **i.** Se agregan nuevas funcionalidades con compatibilidad a la versión actual  **ii.** Se marca como obsoleto funciones que no afectan a la aplicación principal. |
| Z | **Revisión:** representa un cambio de aspecto o corrección de un error pequeño. | La versión revisión Z (x.y.Z | x > 0) DEBE incrementarse cuando se introducen solo arreglos compatibles con la versión anterior. Un arreglo de error se define como un cambio interno que corrige un comportamiento erróneo. | **i.** Se corrigen fallas que no afectan el funcionamiento principal.  **ii.** Modificación de aspectos estéticos.  Se corrigen fallas que no afectan el funcionamiento principal.  **ii.** Modificación de aspectos estéticos. |

1. Cada elemento del versionamiento se debe incrementar numéricamente en incrementos de uno (1).
2. Una vez que un paquete versionado ha sido liberado (release), los contenidos de esa versión NO DEBEN ser modificadas. Cualquier modificación DEBE ser liberada como una nueva versión.
3. La versión mayor en cero (0.Y.Z) es para desarrollo inicial por lo que no debiera ser considerada estable.
4. La versión 1.0.0 define la primera versión estable de la aplicación.
5. La versión revisión Z (x.y.Z | x > 0) DEBE incrementarse cuando se introducen solo arreglos compatibles con la versión anterior. Un arreglo de error se define como un cambio interno que corrige un comportamiento erróneo.
6. La versión menor Y (x.Y.z | x > 0) DEBE ser incrementada si se introduce una nueva funcionalidad compatible con la versión anterior. Se DEBE incrementar si cualquier funcionalidad de la aplicación es marcada como obsoleta. PUEDE ser incrementada si se agrega funcionalidad o arreglos considerables al código. Puede incluir cambios de nivel revisión. La versión revisión DEBE ser reseteada a 0 cuando la versión menor es incrementada.
7. La versión mayor X (X.y.z | X > 0) DEBE ser incrementada si cualquier cambio no compatible con la versión anterior es introducido a la aplicación. PUEDE incluir cambios de nivel menor y/o revisión. Las versiones revisión y menor DEBEN ser reseteadas a 0 cuando se incrementa la versión mayor.
8. Se establecen la utilización de los siguientes identificadores de estabilidad, lo cuales serán escritos en letras minúsculas y separados del versionamiento por un guion medio (“-”):

|  |  |  |
| --- | --- | --- |
| Nombre | Estado de estabilidad | Ejemplo |
| **alpha** | La aplicación se encuentra en el desarrollo, todavía no es una versión estable. | 2.0.0-alpha |
| **beta** | La aplicación se encuentra en un desarrollo final para ser probada para solicitar ser candidata a liberación. | 2.1.3-beta |
| **rc** | La aplicación esta lista para ser liberada como una versión candidata, para pasar a pruebas de usuario. Además, es necesario que se establezca un número de versión para este tipo, donde es un número entero incremental en uno (1), por ejemplo: rc01, rc02, …, rc10, rcN, si el número es menor 10 (N<10), se debe agregar un cero (“0”) delante del número. Ejemplo: rc01, rc02, …, rc09. | 2.4.5-rc01  2.4.5-rc02 |