

IC-6831 Aseguramiento de la Calidad de Software

*Tarea 3: Administración de la
configuración del software*

Profesor: M. Sc. Saúl Calderón Ramírez

Estudiantes:

Carlos M. Girón Alas - 2014113159

Julián J. Méndez Oconitrillo - 2014121700

Daniel A. Troyo Garro - 2014073396



24 de setiembre, 2016.

Índice

1	Introducción	1
2	Lista de ítems de configuración del software	2
2.1	Relaciones entre los ítems de administración de la configuración del software	6
2.2	Copia local del repositorio	6
2.3	Versionamiento Semántico	7
2.3.1	<i>Branch</i> del sprint	7
2.3.2	Actividades finales de un sprint	8
2.3.3	<i>Tags</i> para cada versión interna del sprint	9
3	Control de la configuración	12
3.1	Identificación y documentación de la necesidad de un cambio	12
3.1.1	Ítems de requerimientos	12
3.1.2	Ítems de diseño	12
3.1.3	Ítems de implementación (código fuente)	13
3.2	Análisis, evaluación y aprobación del requerimiento de cambio	13
3.2.1	Ítems de requerimientos	14
3.2.2	Ítems de diseño	14
3.2.3	Ítems de implementación (código fuente)	14
3.2.4	Revisión de cambios	15
3.3	Implementación de los cambios	15
3.3.1	Ítems de requerimientos	15
3.3.2	Ítems de diseño	15
3.3.3	Ítems de implementación (código fuente)	16
3.3.4	Documentación de la implementación	16
4	Conclusión	18

1. Introducción

En el presente trabajo se presentará una explicación de lo que es la implementación de partes de un proceso de administración de configuración de software en el proyecto de segmentación temporal automática de videos de partidos de fútbol. Se presentará el desarrollo parcial de un plan de administración siguiendo el estándar IEEE-828-1998 en el proyecto de segmentación temporal el cual se encuentra localizado en el repositorio git en la siguiente dirección https://github.com/menocsk27/ACS_SegmentTemp.git.

Se mostrarán distintos ejemplos de cómo manejar el versionamiento de cambios en el proyecto y sobre cómo se trabajará el manejo de cambios en los ítems de configuración de software, concepto que será explicado más adelante en el documento. Finalmente, se denotará un plan de administración de la configuración del proyecto que explica paso a paso el manejo de un cambio en cualquier archivo del proyecto y la cascada de cambios que esto implicaría.

Puede decirse que el proceso de administración de la configuración del proyecto de software es la etapa más fundamental para asegurar el éxito del producto que se desarrolla. La capacidad del mantener respaldos, versiones, mejoras, cambios, en un sólo lugar al que se puede acceder de forma automatizada es un método excelente para facilitar todo el proceso de desarrollo y la calidad del mismo. Por otra parte, la definición de todos los elementos involucrados, o ítems, como se les conoce en el estándar IEEE-828-1998 en el que se basa esta tarea, da una perspectiva clara de la forma en que se desarrolla el trabajo y las herramientas que se involucraron en el mismo, asegurando que sea reproducible fácilmente, incluso por personas que no están involucradas en el proceso original de desarrollo.

2. Lista de ítems de configuración del software

La administración de la configuración de software permite la identificación y el manejo apropiado de todos los cambios realizados en el proyecto que pueden tener un impacto significativo sobre el rumbo del mismo. Esta administración conlleva al establecimiento de un conjunto de ítems de administración de la configuración, que es reconocido al tener un identificado apropiado y puede ser tan pequeño como un archivo de código fuente o un diagrama hasta tan grande como un diseño completo. En este trabajo se enlistarán a continuación los ítems de la administración de la configuración del software encontrados, estando algunos como lo son los archivos de código fuente agrupados para reducir la complejidad innecesaria.

Número de ítem	Tipo de ítem	Detalle de ítem	Nombre de ítem	Nombre y versión de herramienta editora	Volatilidad
1	Implementación - Herramienta o software utilizado	Framework compuesto de funciones y procedimientos relacionados a computervision y procesamiento de video especial para el lenguaje Java	opencv-2.4.13. Se puede acceder mediante el link en el archivo herramientas.txt en la carpeta de Herramientas. Versión 2.4.13.	No aplica.	Baja
2	Implementación - Herramienta o software utilizado	Enterprise Architect. Programa CASE de diagramación y modelado UML	SparEntAr121127.rar. Se puede acceder mediante el link en el archivo herramientas.txt en la carpeta de Herramientas. Versión 12.1.	No aplica.	Baja
3	Implementación - Herramienta o software utilizado	Eclipse Java Neon - Entorno de desarrollo integrado para lenguaje Java. El instalador permite instalar la versión deseada.	eclipse-inst-win64.exe Se puede acceder mediante el link en el archivo herramientas.txt en la carpeta de Herramientas.	No aplica.	Media

4	Implementación - Herramienta o software utilizado	Github Desktop- Entorno gráfico o en consola que comunica la versión local del proyecto con un repositorio para versionamiento y recuperación apropiados de los componentes del proyecto.	GitHubSetup.exe. Se puede acceder mediante el link en el archivo herramientas.txt en la carpeta de Herramientas. Versión 3.3.0.0.	No aplica.	Media
5	Implementación - Herramienta o software utilizado	WAMP Server- Aplicación para el levantamiento de servidores locales.	wampserver3.0.6_x64_apache2.4.23_mysql5.7.14_php5.6.25-7.0.10. Se puede acceder mediante el link en el archivo herramientas.txt en la carpeta de Herramientas. Versión 3.0.6..	No aplica.	Baja
6	Implementación - Herramienta o software utilizado	Sublime Text 3 - Entorno de edición de archivos de texto con plugins especializado para programación.	SublimeTextBuild3124x64Setup.exe. Se puede acceder mediante el link en el archivo herramientas.txt en la carpeta de Herramientas. Build 3103	No aplica.	Media
7	Implementación - Herramienta o software utilizado	Plugin de Eclipse para la validación de estándares de codificación en Java.	net.sf.eclipsecs-updatesite_6.19.1.201607051943.rar. Se puede acceder mediante el link en el archivo herramientas.txt en la carpeta de Herramientas. Versión 6.19.	Eclipse Maven Market.	Baja

8	Implementación - Herramienta o software utilizado	Página web para el análisis de estándar de codificación en JavaScript determinado.	JSHint. Accesible mediante link en el archivo herramientas.txt en la carpeta de Herramientas.	Navegador web estándar.	Baja
9	Requerimientos- Documento de Requerimientos	Especificación de requerimientos SyRS, que documenta los requerimientos del sistema.	DocumentoSyRS.pdf	Adobe Acrobat Reader	Media
10	Requerimientos- Documento de Requerimientos	Enunciado inicial del proyecto en el que se especifican los requerimientos funcionales iniciales así como ciertas condiciones de trabajo del proyecto. Además define restricciones técnicas del proyecto como bibliotecas a utilizar y el lenguaje de programación a usar.	Proyecto2.pdf	Adobe Acrobat Reader	Baja
11	Requerimientos- Documento de Requerimientos	Enunciado de las funcionalidades deseadas en la prueba de concepto.	POCspec.pdf	Adobe Acrobat Reader	Baja
12	Requerimientos- Documento de Requerimientos	Enunciado de los ítems deseados en el avance 1 del proyecto, así como los pasos y documentos a presentar.	Avance1.pdf	Adobe Acrobat Reader	Baja
13	Diseño	Archivo de diagramas UML- Diagrama de clases.	diagramas.eap	Enterprise Architect	Alta
14	Diseño	Archivo de diagramas UML - Diagrama de componentes	diagramas.eap	Enterprise Architect	Media

15	Diseño	Archivo de diagramas UML - Diagrama de actividad.	diagramas.eap	Enterprise Architect	Baja
16	Diseño	Archivo de diagramas UML - Diagrama de estados.	diagramas.eap	Enterprise Architect	Media
17	Diseño	Archivo de diagramas UML - Diagrama de usos.	diagramas.eap	Enterprise Architect	Media
18	Implementación (Código fuente)	Archivos de código fuente para la lógica relacionada a la vista o interfaz de usuario.	Aplicación web	Sublime 3	Media
19	Implementación (Código fuente)	Framework para desarrollo de aplicaciones web dinámicas.	AngularJS versión 1.3.15	Sublime 3	Baja (Casi nula)
20	Implementación (Código fuente)	Aplicación de java que se conecta con la principal para permitir la comunicación entre la aplicación web y el Segmentador temporal de video.	Servidor	Eclipse IDE for Java (Neon)	Baja
21	Implementación (Código fuente)	El programa principal, hecho en Java, que corre en el servidor. Se encarga de generar histogramas de los frames de un video para analizarlos y separar las escenas del video.	Segmentador temporal de video	Eclipse IDE for Java (Neon)	Alta

2.1. Relaciones entre los ítems de administración de la configuración del software

Tras obtener la lista de ítems de configuración del proyecto, se procede a enlistar las relaciones que hay entre ellos. Este paso es importante ya que permite establecer la cadena de efectos que conllevaría el cambio en un ítem y el impacto que esto tendría sobre la estructura general del proyecto. A continuación se muestra un diagrama UML generalizado que muestra las relaciones de dependencia y composición de los ítems de configuración de software.

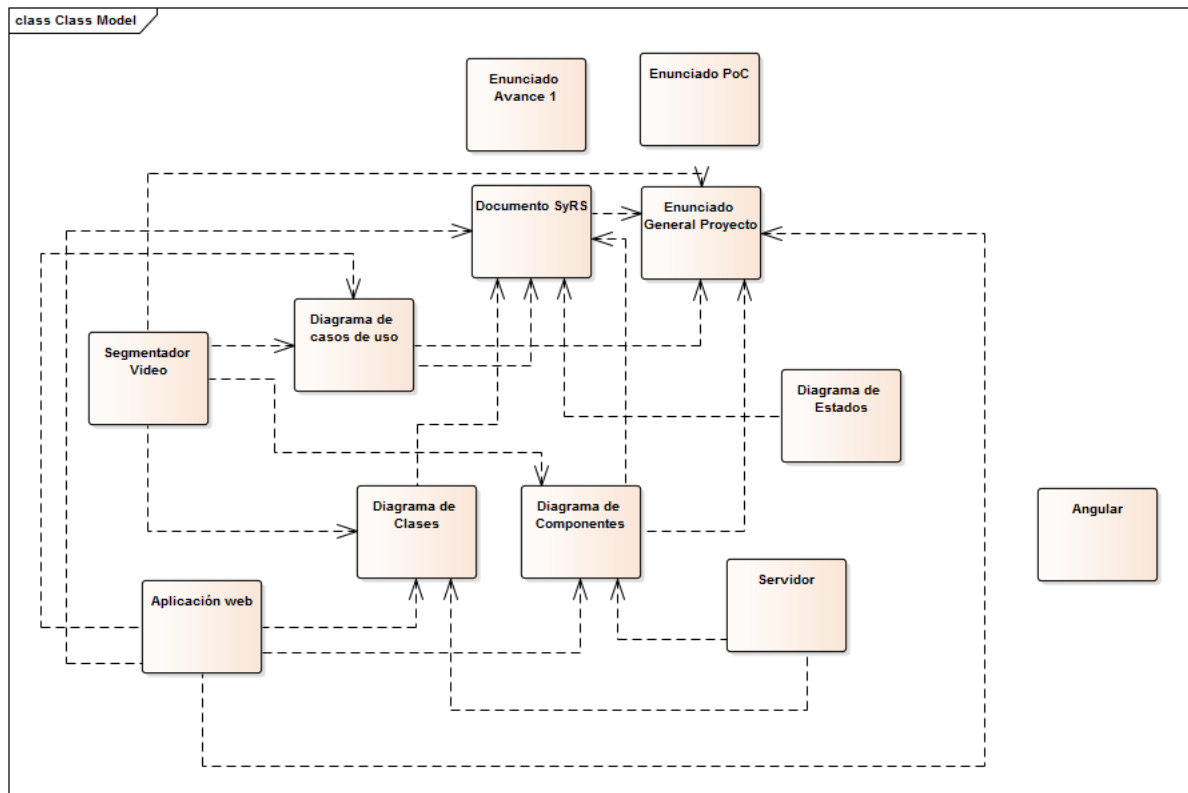


Figura 1. Relaciones entre los ítems de configuración

2.2. Copia local del repositorio

Como se explicó en la introducción de este trabajo, a lo largo del mismo se irán presentando ejemplos de procedimientos para la correcta utilización de la herramienta de versionamiento Git en el proyecto actual. El primero de estos procedimientos es la generación de una rama o *branch* local de trabajo generada a partir del repositorio principal. Para esto necesitamos iniciar la herramienta de Git, la cual es una aplicación CLI (*Command Line Interface*) que también suele ser usada encapsulada tras una aplicación gráfica como lo es Github Desktop, que es la herramienta utilizada por los miembros del equipo de desarrollo de este proyecto.

```
1  cd copia
2  git clone https://github.com/menocsk27/ACS.SegmentTemp.git
3  cd ACS.SegmentTemp
4  git branch branchLocal
5  git branch
6  git checkout branchLocal
```

Los comandos anteriores permiten el clonado del repositorio y de su estructura general, la cual está ordenada de acuerdo al tipo de ítem de configuración de software albergados en cada

carpeta. El primer comando establece la carpeta en la cual se desea alberga el repositorio, la cual llamaremos *CarpetaCopia*. Para este procedimiento, **se asume que el usuario ya creó dicha carpeta**. Una vez dentro de la carpeta, procedemos a clonar el repositorio mediante el comando *git clone* e insertando como parámetro la URL del repositorio. Este comando creará una carpeta con la estructura del repositorio tal y como se encuentra en la rama principal, la cual es llamada *master* en Git.

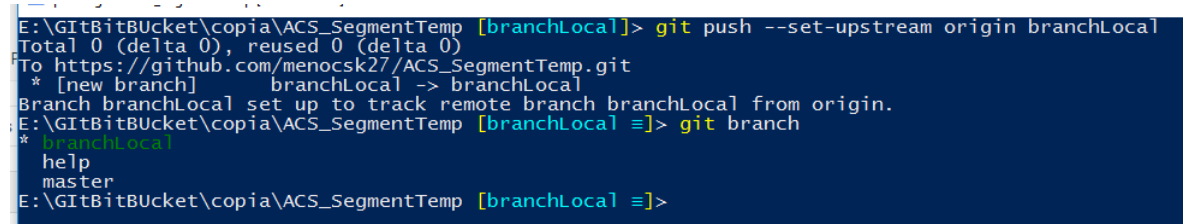
Una rama o *branch* es una derivación o versión de una parte del proyecto. Es utilizada para que los procesos de desarrollo de nuevas características no afecten a la versión base del proyecto, por lo que se suelen crear *branches* nuevas para cada feature nuevo en desarrollo. En nuestro caso, por cada sprint se generará un nuevo branch en donde los cambios se unirán con los del branch *master* hasta que estén completamente implementados de manera estable, tras haber pasado por pruebas y la evaluación del equipo de desarrollo. Los comandos siguientes denotan la creación **local** de un branch. Este branch no existe todavía en el repositorio principal y sólo se puede acceder a él en la terminal en la cuál se creó. Para poder subir o mostrar este branch en el repositorio principal, se tiene que ejecutar el comando **git push** tras haber realizado un cambio guardado, el cual se logra mediante un *commit* en git.

2.3. Versionamiento Semántico

Una línea base o *baseline* es una versión del proyecto estable sobre la cual se puede comenzar un nuevo sprint o ronda de cambios. En este proyecto se establecen dos líneas base identificables: el fin de la prueba del concepto y el fin del avance 1, suponiendo cada uno las condiciones apropiadas para el inicio de un nuevo sprint. Para cada línea base y dado la mínima cantidad de cambios sobre la funcionalidad y los ítems de administración de la configuración de software modificados, cada sprint realizado es considerado como un parche para el versionamiento semántico, estándar de etiquetado de versiones utilizado en este proyecto.

2.3.1. Branch del sprint

Anteriormente se explicó cómo crear un branch **local** a partir de la rama maestra del repositorio. Ahora se enseñará a crear dicho branch de manera sincronizada con el repositorio. La siguiente serie de comandos deben ser ejecutados tras ejecutar los pasos de la sección 2.2 de este documento.



```
E:\GITBitBUCKET\copia\ACS_SegmentTemp [branchLocal]> git push --set-upstream origin branchLocal
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/menocsk27/ACS_SegmentTemp.git
 * [new branch]      branchLocal -> branchLocal
Branch branchLocal set up to track remote branch branchLocal from origin.
E:\GITBitBUCKET\copia\ACS_SegmentTemp [branchLocal =>]> git branch
 * branchLocal
  help
  master
E:\GITBitBUCKET\copia\ACS_SegmentTemp [branchLocal =>]>
```

Figura 2. Subida del branch local al repositorio.

```
1 git push --set-upstream origin branchLocal
2 git branch
```

Los comandos anteriores permiten subir el branch creado anteriormente al repositorio. Así, una vez todos los demás miembros del equipo de desarrollo sincronicen sus copias locales con el repositorio, podrán acceder a dicho branch y poder ver los cambios realizados en él.

2.3.2. Actividades finales de un sprint

Al finalizar un sprint se realizarán las actividades de verificación y validación de los cambios y funciones realizadas en dicho sprint. Estas actividades incluyen desde la verificación de la utilización correcta de estándares de codificación hasta el manejo apropiado de la herramienta de versionamiento. A continuación se enlista tales actividades:

- **Revisión de la conformidad del código javascript respecto al estándar de JSHint. Herramientas:** Se utilizará la herramienta de JSHint para evaluar cada archivo de código fuente javascript. Los pasos a seguir son los siguientes:
 1. Introducir el código fuente javascript a la entrada del sitio web jshint.com
 2. Realizar las modificaciones recomendadas por la aplicación para cumplir el estándar.
 3. Reemplazar el código fuente en el archivo oficial con el modificado de la aplicación web.
 4. Registrar los cambios en la rama de trabajo, tal y como se explica más adelante en Control de la configuración.
- **Revisión de la conformidad del código java respecto al estándar de Google Java Style. Herramientas:** Se utilizará la herramienta de CheckStyle y Eclipse Java Neon para evaluar cada archivo de código fuente java. Los pasos a seguir son los siguientes:
 1. Apertura del proyecto de java en Eclipse Java Neon.
 2. Click derecho sobre el proyecto
 3. Se selecciona cada archivo fuente con click Derecho.
 4. Se selecciona CheckStyle > Validate CheckStyle fixes.
 5. Se corrigen las líneas señaladas.
 6. Registrar los cambios en la rama de trabajo, tal y como se explica más adelante en Control de la configuración.
- **Revisión de la conformidad del código respecto al diseño. Herramientas:** Se utilizará la herramienta de Enterprise Architect para poder generar una plantilla. Los pasos a seguir son los siguientes:
 1. Apertura del ítem de configuración que almacena los diagramas con el programa Enterprise Architect.
 2. Se genera el código fuente a partir de los diagramas principales de clases para que cumplan con el diseño.
 3. Se compara dicho código generado con los archivos de código fuente oficiales.
 4. Si hay diferencias, asegurarse de que la modificación de los archivos de código fuente oficiales sean los modificados para que la comparación sea igual.
 5. Registrar los cambios en la rama de trabajo, tal y como se explica más adelante en Control de la configuración.
- **Revisión de la conformidad del diseño respecto a los requerimientos. Herramientas:** Se utilizará la herramienta de Enterprise Architect para la visualización del diagrama. Los pasos a seguir son los siguientes:
 1. Se obtiene la lista de requerimientos que abarcan el sprint actual.
 2. Se verifica que cada uno esté cubierto por alguna sección del diseño.
- **Realización de pruebas unitarias por cada función del proyecto. Herramientas:** Se utilizará la herramienta JUnit para la realización y validación de pruebas del lado del servidor, así como Eclipse Java Neon como ambiente de desarrollo. Los pasos a seguir son los siguientes:

1. Desarrollo de las pruebas unitarias que validen cada posible caso para las nuevas funciones implementadas.
 2. Ejecución de todas las pruebas unitarias existentes hasta el momento.
 3. Validación del resultado de las pruebas.
 4. Si hay un error, corregir el error en la implementación de la función y volver a validar las pruebas.
- Realización de pruebas de integración. **Herramientas:** Se utilizará la herramienta JUnit para la realización y validación de pruebas de integración entre componentes del lado del servidor, así como Eclipse Java Neon como ambiente de desarrollo. Los pasos a seguir son los siguientes:
 1. Desarrollo de las pruebas de integración que validen cada posible caso para los nuevos componentes adjuntos a los componentes antiguos.
 2. Ejecución de todas las pruebas de integración existentes hasta el momento.
 3. Validación del resultado de las pruebas.
 4. Si hay un error, corregir el error en la implementación de los componentes involucrados y volver a validar las pruebas.
 - Realización de pruebas de aceptación. **Herramientas:** No se hará uso de herramientas importantes en esta etapa, adicional a una encuesta al cliente mediante servicios de formularios en línea, como Google Forms. Los pasos a seguir son los siguientes:
 1. Reunión con el cliente.
 2. Se le indica al cliente que ejecute las funciones agregadas en el sprint.
 3. Se le pide al cliente responder una encuesta que validará el pase final de las funciones agregadas en el sprint.
 - Revisión de la estructura general de la rama maestra del repositorio y sincronización general. **Herramientas:** Se utilizará el software de versionamiento Git, disponible mediante la herramienta GitHub Desktop. Los pasos a seguir son los siguientes:
 1. Nombramiento de la versión interna de la última modificación siguiendo el estándar de versionamiento semántico mediante el uso de *tags* en Git.
 2. Validación de que las copias locales de cada terminal de trabajo esté sincronizada con el repositorio.
 3. Apertura de la nueva rama a partir de la rama maestra para el siguiente sprint.

2.3.3. Tags para cada versión interna del sprint

Las etiquetas o *tags* son nombres que se le pueden dar a puntos específicos de la historia del proyecto para poder establecer un marcador de algún acontecimiento importante que ocurrió. En Git, se le suelen enlazar a *commits*, que es el término con el que se refiere a un conjunto de cambios realizados por un miembro contribuidor del repositorio. Las etiquetas son ideales para poder llevar un control del versionamiento y de las líneas base de un proyecto, así se puede añadir una etiqueta al *commit* que estableció una línea base y se puede regresar a esta línea base para poder visualizar el estado de ella o simplemente generar una versión alternativa a partir de este punto. Para poder ver las etiquetas de un repositorio de git, se ejecuta el siguiente comando:

```
1 git tag
```

```
E:\GitBitBUcket\copia\ACS_SegmentTemp [master =>] > git tag
v0.0.0
v0.0.1
v0.0.2
E:\GitBitBUcket\copia\ACS_SegmentTemp [master =>] > _
```

Figura 3. Etiquetas existentes en el proyecto.

Cada uno de esas etiquetas fue asignada a un commit específico en la historia del proyecto que definió el final de un sprint, por lo que se podría definir que estableció una nueva línea base para poder iniciar un nuevo sprint. Debido a que los cambios en cada sprint no fueron tan significativos, sólo se realizó un aumento el número del parche, ni siquiera llegando a lo que se considera una versión menor. Realizaremos una simulación utilizando la rama **branchLocal** creada anteriormente, asumiendo que esta contiene los cambios del sprint y realizaremos una línea base a partir de esta. Primero, volveremos a la rama mediante el comando `git checkout branchLocal` y una vez allí, realizaremos un cambio que consistirá en la añadidura de un archivo de texto vacío llamado **ejemplo.txt**.

Name	Date modified	Type	Size
Asignacion_Responsabilidades	24/9/2016 8:49 p. m.	File folder	
Avance1_Miscelaneo	24/9/2016 8:49 p. m.	File folder	
Diseño	24/9/2016 8:49 p. m.	File folder	
Herramientas	24/9/2016 8:49 p. m.	File folder	
Implementación	24/9/2016 8:49 p. m.	File folder	
PoC_Miscelaneo	24/9/2016 8:49 p. m.	File folder	
Requerimientos	24/9/2016 8:49 p. m.	File folder	
.classpath	24/9/2016 8:49 p. m.	CLASSPATH File	1 KB
.gitignore	24/9/2016 8:49 p. m.	GITIGNORE File	1 KB
.project	24/9/2016 8:49 p. m.	PROJECT File	1 KB
ejemplo	24/9/2016 11:12 p....	TXT File	0 KB
README.md	24/9/2016 8:49 p. m.	MD File	1 KB
tareass	24/9/2016 8:49 p. m.	TXT File	5 KB
Test rest webservice	24/9/2016 8:49 p. m.	WinRAR ZIP archive	8 KB
xuvfmymb.hb1	24/9/2016 8:49 p. m.	TXT File	1 KB

Figura 4. Se realiza un cambio en el branch del repositorio.

```
1 git status
2 git add ejemplo.txt
3 git commit -m "Cambio final sprint"
4 git push
5 git tag -a v0.0.3 -m "Tag prueba"
6 git tag
7 git push origin --tags
8 git checkout master
9 git merge branchLocal
10 git tag
```

Tras realizar el cambio anterior, se puede ver cómo se implementa el cambio en el repositorio y en la rama *master* tras implementar el comando `merge`. Así se puede crear una nueva línea base con etiqueta.

Algo útil de las etiquetas es que permiten visualizar el estado del proyecto en ese punto específico de la historia del proyecto. Mediante el siguiente comando, se puede generar un *branch* que se genere a partir del estado del *commit* enlazado a la etiqueta de la versión 0.0.1.

```
1 git checkout -b version0.0.1 v0.0.1
```

```

posh-git ~ ACS_SegmentTemp [master]
E:\GitBitBucket\copia\ACS_SegmentTemp [branchLocal ≡ +1 ~0 -0 1]> git status
On branch branchLocal
Your branch is up-to-date with 'origin/branchLocal'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    ejemplo.txt

nothing added to commit but untracked files present (use "git add" to track)
E:\GitBitBucket\copia\ACS_SegmentTemp [branchLocal ≡ +1 ~0 -0 1]> git add ejemplo.txt
E:\GitBitBucket\copia\ACS_SegmentTemp [branchLocal ≡ +1 ~0 -0 ~1]> git commit -m "Cambio final sprint"
[branchLocal 531115a] Cambio final sprint
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 ejemplo.txt
E:\GitBitBucket\copia\ACS_SegmentTemp [branchLocal 1]> git push
Counting objects: 2, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 273 bytes | 0 bytes/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local objects.
To https://github.com/menocsk27/ACS_SegmentTemp.git
   bdf32dc..531115a  branchLocal -> branchLocal
E:\GitBitBucket\copia\ACS_SegmentTemp [branchLocal ≡]> git tag -a v0.0.3 -m "Tag prueba"
E:\GitBitBucket\copia\ACS_SegmentTemp [branchLocal ≡]> git tag
v0.0.0
v0.0.1
v0.0.2
v0.0.3
E:\GitBitBucket\copia\ACS_SegmentTemp [branchLocal ≡]> git push origin --tags
Counting objects: 1, done.
Writing objects: 100% (1/1), 168 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To https://github.com/menocsk27/ACS_SegmentTemp.git
 * [new tag]         v0.0.3 -> v0.0.3
E:\GitBitBucket\copia\ACS_SegmentTemp [branchLocal ≡]> git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
E:\GitBitBucket\copia\ACS_SegmentTemp [master ≡]> git merge branchLocal
Updating bdf32dc..531115a
Fast-forward
 ejemplo.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 ejemplo.txt
E:\GitBitBucket\copia\ACS_SegmentTemp [master 1]> git tag
v0.0.0
v0.0.1
v0.0.2
v0.0.3
E:\GitBitBucket\copia\ACS_SegmentTemp [master 1]>

```

Figura 5. Se implementa el cambio en el repositorio

3. Control de la configuración

Las actividades de configuración del control solicitan, evalúan, aprueban e implementan cambios a ítems de administración de la configuración pertenecientes a una línea base. A continuación se muestra la serie de actividades a realizar por cada uno de los ítems de configuración enlistados anteriormente. Se ignoran los ítems de herramientas ya que realizar un cambio en ellos conlleva el reemplazo total de dicho ítem. Para los ítems de tipo Implementación (Archivos de código fuente), Diseño y Requerimientos se definieron plantillas referentes a lo que es identificación y documentación de la necesidad de un cambio.

3.1. Identificación y documentación de la necesidad de un cambio

3.1.1. Ítems de requerimientos

Los siguientes ítems pertenecen a esta categoría:

- Documento SyRS
- Enunciado general del proyecto
- Enunciado del Avance 1
- Enunciado del PoC

Para esta categoría se definió la siguiente secuencia de pasos que permiten la identificación y documentación de un alteración en el ítem de la categoría actual:

1. Selección del documento de requerimientos a cambiar.
2. Selección de la sección del documento a cambiar.
3. Selección del requerimiento específico a cambiar.
4. Justificación de la alteración.
5. Nombre de la persona que solicita el cambio y grupo al que pertenece.
6. Personas a las que se le notifica del cambio.
7. Urgencia del cambio (Baja, Media, Alta)
8. Fecha de la solicitud del cambio
9. Diagramas adicionales que deben ser alterados tras dicha alteración.
10. Archivos de código fuente que deben ser alterados tras dicho cambio.

3.1.2. Ítems de diseño

Los siguientes ítems pertenecen a esta categoría:

- Diagrama de Clases
- Diagrama de Estados
- Diagrama de Actividad

- Diagrama de Componentes
- Diagrama de Casos de Uso

Para esta categoría se definió la siguiente secuencia de pasos que permiten la identificación y documentación de un alteración en el ítem de la categoría actual:

1. Selección del diagrama a modificar.
2. Nombre de la persona que solicita el cambio y grupo al que pertenece.
3. pertenece.
4. Personas a las que se le notifica del cambio.
5. Sección o componente a alterar de dicho diagrama.
6. Justificación del cambio.
7. Urgencia del cambio (Baja, Media, Alta)
8. Fecha de la solicitud del cambio
9. Archivos de código fuente que deben ser alterados tras dicho cambio.

3.1.3. Ítems de implementación (código fuente)

Los siguientes ítems pertenecen a esta categoría:

- Aplicación web
- AngularJS
- Servidor
- Segmentador Temporal de Video

Para esta categoría se definió la siguiente secuencia de pasos que permiten la identificación y documentación de un alteración en el ítem de la categoría actual:

1. Selección de la sección del código o función del archivo fuente a modificar.
2. Justificación de la alteración.
3. Nombre de la persona que solicita el cambio y grupo al que pertenece.
4. Personas a las que se le notifica del cambio.
5. Urgencia del cambio (Baja, Media, Alta)
6. Fecha de la solicitud del cambio
7. Diagramas adicionales que deben ser alterados tras dicha alteración.
8. Archivos de código fuente que deben ser alterados tras dicho cambio.

3.2. Análisis, evaluación y aprobación del requerimiento de cambio

Para cada plan de configuración de control, se debe denotar el análisis realizado para denotar el impacto que tendría el cambio sobre el proyecto, así como el medio por el cual este análisis se realiza. También se deben especificar los procedimientos para la revisión de los resultados del análisis. Para este documento, se presentará un procedimiento generalizado para los ítems de configuración de software debido a que no se tiene una definición durable de los ítems de administración de la configuración del software y que enlistar una serie de pasos personalizados para la evaluación y aprobación de cada ítem añadiría una complejidad sumamente innecesaria al proceso de administración de la configuración del software del proyecto.

3.2.1. Ítems de requerimientos

Los siguientes ítems pertenecen a esta categoría:

- Documento SyRS
- Enunciado general del proyecto
- Enunciado del Avance 1
- Enunciado del PoC

Para esta categoría se definió la siguiente secuencia de pasos que permiten el análisis, evaluación y aprobación de una alteración en el ítem de la categoría actual:

1. Notificación utilizando una aplicación de mensajería instantánea a los miembros del equipo de desarrollo acerca del cambio.
2. Contacto al cliente consultándole acerca del cambio a realizar.
3. Videoconferencia del equipo de desarrollo para evaluar el impacto del cambio.

3.2.2. Ítems de diseño

Los siguientes ítems pertenecen a esta categoría:

- Diagrama de Clases
- Diagrama de Estados
- Diagrama de Actividad
- Diagrama de Componentes
- Diagrama de Casos de Uso

Para esta categoría se definió la siguiente secuencia de pasos que permiten el análisis, evaluación y aprobación de una alteración en el ítem de la categoría actual:

1. Notificación utilizando una aplicación de mensajería instantánea a los miembros del equipo de desarrollo acerca del cambio.
2. Evaluación de la relación entendibilidad-complejidad del diseño y la implementación de dicho cambio del diseño.
3. Videoconferencia del equipo de desarrollo para evaluar el impacto del cambio.

3.2.3. Ítems de implementación (código fuente)

Los siguientes ítems pertenecen a esta categoría:

- Aplicación web
- AngularJS
- Servidor
- Segmentador Temporal de Video

Para esta categoría se definió la siguiente secuencia de pasos que permiten el análisis, evaluación y aprobación de una alteración en el ítem de la categoría actual:

1. Notificación utilizando una aplicación de mensajería instantánea a los miembros del equipo de desarrollo acerca del cambio.
2. Investigación acerca de métodos de implementación de dicho cambio.
3. Videoconferencia del equipo de desarrollo para evaluar el impacto del cambio.

3.2.4. Revisión de cambios

Git permite evaluar los cambios específicos de cada archivo legible como lo son archivos de código fuente de manera individual. De modo que si se realiza un cambio en el archivo de texto *ejemplo.txt* que para este punto tenía una línea de texto ya, este cambio puede ser visualizado mediante el siguiente comando:

```
1 git diff ejemplo.txt
```

Con este comando se puede visualizar la diferencia entre la versión a añadir al repositorio y la versión actual en el repositorio. Las líneas mostradas en rojo son aquellas líneas borradas y las líneas mostradas en verde son aquellas añadidas. En este caso se modificó la primera línea que contenía **Ejemplo 1** con la línea **Ejemplo 3** y se añadieron las líneas **Ejemplo 2** y **Ejemplo 4**.

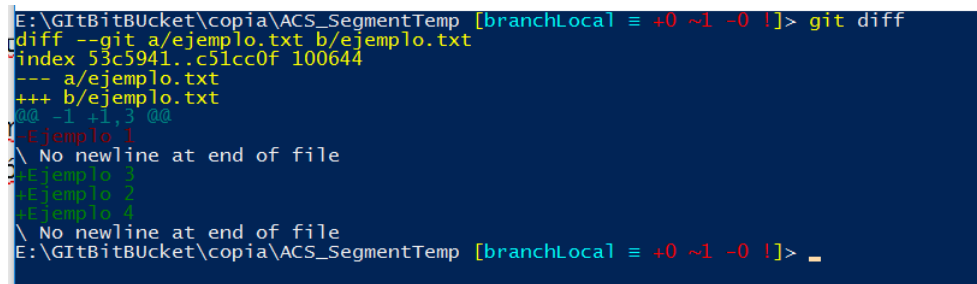


Figura 6. Diferencia entre las versiones del repositorio y la actual local de ejemplo.txt

3.3. Implementación de los cambios

En esta sección se definen las actividades necesarias para verificar, validar e implementar los cambios realizados. Existe una secuencia de pasos predefinidos para cada tipo de ítem de configuración de software y esta es especificada a continuación:

3.3.1. Ítems de requerimientos

Los siguientes ítems pertenecen a esta categoría:

- Documento SyRS
- Enunciado general del proyecto
- Enunciado del Avance 1
- Enunciado del PoC

Para esta categoría se definió la siguiente secuencia de pasos que permiten la validación e implementación de una alteración en el ítem de la categoría actual:

1. Consulta al cliente acerca del cambio.
2. Cambios realizados en los diseños dependientes de dicho requerimiento.

3.3.2. Ítems de diseño

Los siguientes ítems pertenecen a esta categoría:

- Diagrama de Clases
- Diagrama de Estados

- Diagrama de Actividad
- Diagrama de Componentes
- Diagrama de Casos de Uso

Para esta categoría se definió la siguiente secuencia de pasos que permiten la validación e implementación de una alteración en el ítem de la categoría actual:

1. Si se implementa un patrón de diseño, dicho patrón tiene que ser validado con fuentes bibliográficas seguras.
2. Si se agrega un diagrama, el diagrama debe ser añadido al archivo **diagramas.eap** del formato de la herramienta **Enterprise Architect**.

3.3.3. Ítems de implementación (código fuente)

Los siguientes ítems pertenecen a esta categoría:

- Aplicación web
- AngularJS
- Servidor
- Segmentador Temporal de Video

Para esta categoría se definió la siguiente secuencia de pasos que permiten la validación e implementación de un alteración en el ítem de la categoría actual:

- Paso de pruebas unitarias si es un componente java del servidor. Se utiliza la herramienta **JUnit**.
- Validación de la conformidad según el estándar de codificación. Google Java Style si es Java utilizando la herramienta **CheckStyle** y estándar general si es Javascript utilizando la herramienta **JSHint**.
- Paso de pruebas de integración si es un componente java del servidor. Se utiliza la herramienta **JUnit**.

3.3.4. Documentación de la implementación

Cada cambio en la rama debe ser añadido a un componente de git llamado *stash*. Este elemento alberga todos los cambios que serán añadidos en el próximo *commit* de la rama actual y mediante la siguiente cadena de comandos se puede realizar dicho proceso. Algo importante que se tiene que notar es que el comando *git add .* añade todos los archivos adicionales o modificados al *stash*, por lo que si se quiere añadir sólo uno se debería ejecutar *git add archivoParaStash*.

Al igual que como se realizó anteriormente, se continuará trabajando sobre la rama *branchLocal* creada anteriormente. Se modificó el archivo de texto **ejemplo.txt** creado anteriormente y ahora se quiere

```
1 git add .
2 git commit -m "Archivos modificados"
3 git push
4 git checkout branchLocal
```

De esta manera se tienen los cambios guardados de manera sincronizada con el repositorio y así se puede llevar un control más regulado de lo que son los cambios en un sprint específico.

```
E:\GitBitBUCKET\copia\ACS_SegmentTemp [branchLocal ≡ +0 ~2 -0 !]> git add .
E:\GitBitBUCKET\copia\ACS_SegmentTemp [branchLocal ≡ +0 ~2 -0 ~]> git commit -m "Archivos modificados"
[branchLocal 3d19017] Archivos modificados
 2 files changed, 1 insertion(+)
E:\GitBitBUCKET\copia\ACS_SegmentTemp [branchLocal !]> git push
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 1.16 KiB | 0 bytes/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/menocsk27/ACS_SegmentTemp.git
 531115a..3d19017 branchLocal -> branchLocal
```

Figura 7. Se implementa el cambio en el repositorio

4. Conclusión

Para medir el éxito de un proyecto de software hay que tomar en cuenta las condiciones en las que se hizo y también el elemento de trascendencia que este tuvo. Se entiende normalmente en la industria que no existe un software perfecto, más bien, se busca que un software tenga la capacidad de mejorar, crecer y evolucionar con el paso del tiempo sin que estos procesos presenten una dificultad muy elevada. Con esto en mente, la administración de la configuración tiene un nuevo sentido para los desarrolladores de software, pues facilita el proceso de adaptación al que el software es sometido conforme pasa el tiempo.

La forma más sencilla de manejar los cambios es comprendiendo exactamente por qué ocurren (condiciones del negocio, nuevas demandas, reorganización, restricciones) y qué cosas deben modificarse para adaptarse a ellos. Así, la definición de ítems que separen y categoricen los elementos que componen el proyecto facilita enormemente la ubicación y caracterización de los cambios que se deben realizar.

En el trabajo realizado queda clara la persecución por el aseguramiento de políticas y procedimientos para la creación, cambio y testeado del código, así como el cumplimiento de las mismas, y un análisis práctico de las herramientas más usadas respecto al versionamiento de los proyectos como lo es el uso de repositorios. Si hacemos referencia a los elementos de la administración de la configuración, podemos hacer una analogía clara entre las herramientas utilizadas en este trabajo y los elementos componentes (repositorio y versionamiento), de proceso (planificación y tareas), y de construcción (las diversas herramientas nombradas para los estándares y el desarrollo efectivo del proyecto). Además, el elemento humano es manejado en todo momento del trabajo, pues el equipo de desarrollo se mantiene en constante comunicación con las herramientas nombradas, aunque cambios en estas no tengan mayor impacto y se agregan para especificar más a fondo todos los elementos utilizados.

Por último es relevante notar que la configuración no es un elemento irrelevante como algunos podrían argumentar. Vista la necesidad de automatizar tareas y definir estándares que favorezcan el desarrollo fluido y eficiente de los proyectos cuyas dimensiones pueden ser insostenibles, la administración de la configuración se muestra como una herramienta indispensable para alcanzar el éxito y posteriormente mantenerlo.

Referencias

The Institute of Electrical and Electronics Engineers, Inc. (1998). *IEEE Std 828-1998 - IEEE Standard for Software Configuration Management Plans* (IEEE Computer Society).