

**BIGTECH CONSULTING**

**PLAN DE GESTIÓN DE LA CONFIGURACIÓN DEL SOFTWARE**

1. **Introducción**
2. **Situación de la empresa**

En BigTech contamos con un equipo de profesionales altamente capacitados y apasionados por la tecnología, que se dedican a brindar soluciones de software innovadoras y efectivas para nuestros clientes.

Nos esforzamos por mantener altos estándares de calidad en todos los proyectos en los que trabajamos, y buscamos ofrecer soluciones personalizadas y efectivas para satisfacer las necesidades de nuestros clientes. Además, nos enfocamos en la investigación y el desarrollo de nuevas tecnologías y en estar siempre al día con las últimas tendencias en el mercado.

Nos tomamos en serio nuestra responsabilidad social y ambiental, y nos esforzamos por tener un impacto positivo en nuestras comunidades y en el medio ambiente.

Pero como cualquier empresa, estamos experimentando una serie de problemas con nuestro sistema de control de versiones. Hemos notado que hay una falta de protocolos claros establecidos para el uso del control de versiones, lo que ha llevado a algunos miembros del equipo a hacer cambios que no se ajustan a los estándares de la empresa o que no son coherentes con los objetivos del proyecto.

Además, hemos notado dificultades para compartir y colaborar en el código fuente y otros artefactos del proyecto debido a que los miembros del equipo no están utilizando el mismo control de versiones o no están siguiendo los mismos protocolos. También hemos experimentado problemas de integración debido a que se están utilizando múltiples herramientas de control de versiones en diferentes partes de la empresa.

La pérdida de datos ha sido otro problema para nosotros en BigTech Consulting. Hemos descubierto que si no se realiza correctamente el seguimiento de las versiones, se puede perder información importante. Por ejemplo, se han encontrado casos en los que se ha sobrescrito una versión anterior con una nueva versión sin guardar la anterior, lo que ha resultado en la pérdida de toda la información contenida en la versión anterior.

También hemos estado experimentando conflictos de fusión cuando dos o más personas trabajan en el mismo archivo al mismo tiempo y realizan cambios diferentes. Si no se resuelven estos conflictos adecuadamente, se pueden perder cambios importantes o incluso corromper el archivo.

Finalmente, hemos notado que hay una falta de control de acceso, lo que ha llevado a que cualquier persona con acceso al repositorio pueda realizar cambios. Esto ha provocado errores y la eliminación de archivos importantes. Estamos trabajando para establecer protocolos claros para el uso del control de versiones y mejorar la colaboración y el trabajo en equipo en el proyecto.

A continuación hablaremos sobre los problemas que presentamos y sus consecuencias para nuestra empresa:

1. Falta de protocolos claros: Si no hay protocolos claros establecidos para el uso del control de versiones, los miembros del equipo pueden hacer cambios que no se ajusten a los estándares de la empresa o que no sean coherentes con los objetivos del proyecto.
2. Dificultades para compartir y colaborar: Si los miembros del equipo no están utilizando el mismo control de versiones o no están siguiendo los mismos protocolos, puede haber dificultades para compartir y colaborar en el código fuente y otros artefactos del proyecto.
3. Problemas de integración: Si se están utilizando múltiples herramientas de control de versiones en diferentes partes de la empresa, puede ser difícil integrar y gestionar el código fuente y otros artefactos del proyecto.
4. Pérdida de datos: Si no se realiza correctamente el seguimiento de las versiones, se puede perder información importante. Por ejemplo, si se sobrescribe una versión anterior con una nueva versión sin guardar la anterior, se pierde toda la información contenida en la versión anterior.
5. Conflictos de fusión: Si dos o más personas trabajan en el mismo archivo al mismo tiempo y realizan cambios diferentes, puede haber conflictos al fusionar los cambios. Si no se resuelven estos conflictos adecuadamente, se pueden perder cambios importantes o incluso corromper el archivo.
6. Falta de control de acceso: Si no se establecen permisos de acceso adecuados, cualquier persona con acceso al repositorio puede realizar cambios, lo que puede provocar errores o incluso la eliminación de archivos importantes.
7. **Propósito del plan**

El propósito del Plan es establecer procesos y procedimientos para la gestión efectiva de la configuración del software en la empresa, lo que garantiza la calidad y control en los cambios realizados en el software. Además, este plan busca asegurar que los cambios se implementen de manera controlada y efectiva, lo que garantiza que el software funcione correctamente y cumpla con los requisitos del cliente y del negocio. En general, el objetivo es establecer un enfoque estructurado y coherente para la gestión de la configuración del software, lo que mejora la eficiencia y efectividad del proceso de desarrollo de software en la empresa.

1. **Benchmarking de herramientas** Se realizará un análisis comparativo de las herramientas disponibles en el mercado para la gestión de la configuración del software. Se evaluarán las herramientas según su capacidad para gestionar la configuración del software, su facilidad de uso, su compatibilidad con otros sistemas y herramientas utilizadas en la empresa y su costo. Se seleccionará la herramienta más adecuada para la empresa y se definirán los procedimientos para su implementación y uso.

* **MERCURIAL**

En el desarrollo de software, el control de versiones es esencial para colaborar en proyectos y asegurarse de que no se pierda información o se creen conflictos en el código fuente. Una de las herramientas más populares para el control de versiones es Mercurial.

***Características:***

* Sistema de control de versiones distribuido.
* Facilidad de uso, aprendizaje y uso sencillo.
* Velocidad y rendimiento en el manejo de grandes repositorios.
* Flexibilidad y personalización a través de una gran cantidad de herramientas y plugins.
* Compatibilidad multiplataforma (Windows, Linux, Mac OS X).
* Permite una gestión eficiente de ramas y fusión de ramas.
* Ofrece soporte para control de acceso y permisos.
* Cuenta con una arquitectura modular y escalable.
* Ofrece una integración sencilla con otras herramientas de software.

***Uso en el desarrollo de software:***

Mercurial es ampliamente utilizado en el desarrollo de software debido a su facilidad de uso, velocidad y flexibilidad. Los desarrolladores pueden trabajar en equipo y colaborar en un proyecto sin temor a perder información o a crear conflictos en el código fuente. Mercurial también es compatible con múltiples plataformas, incluyendo Windows, Linux y Mac OS X.

***Ventajas:***

* Fácil de aprender y de utilizar.
* Velocidad y rendimiento en el manejo de grandes repositorios.
* Sistema de control de versiones distribuido, lo que permite que cada desarrollador tenga su propia copia del repositorio.
* Permite la gestión eficiente de ramas y la fusión de ramas.
* Compatibilidad multiplataforma.
* Amplia gama de herramientas y plugins que pueden ser utilizados para personalizar y mejorar su funcionalidad.
* Ofrece soporte para control de acceso y permisos.
* Arquitectura modular y escalable.
* Integración sencilla con otras herramientas de software.

**Desventajas:**

* Puede ser menos conocido que otras herramientas de control de versiones como Git.
* Curva de aprendizaje para los desarrolladores que no están familiarizados con el uso de sistemas de control de versiones distribuidos.
* Puede ser difícil de manejar para proyectos muy grandes o complejos.
* Requiere un servidor centralizado para compartir cambios entre desarrolladores, lo que puede aumentar la complejidad en entornos de equipo muy grandes.}
* **APACHE SUBVERSION**

Apache Subversion, también conocido como SVN, es una herramienta de control de versiones de código abierto, desarrollada por la Apache Software Foundation. SVN se ha utilizado en proyectos de desarrollo de software de diferentes tamaños, desde pequeños hasta grandes corporaciones.

SVN fue creado para ser una alternativa a los sistemas de control de versiones centralizados como CVS. En lugar de tener un servidor centralizado que controle la versión del código, SVN utiliza una arquitectura de servidor-cliente, permitiendo a cada desarrollador tener una copia local del repositorio. Los cambios se sincronizan con el servidor central cuando es necesario.

***Características:***

* Sistema de control de versiones centralizado, que utiliza una arquitectura cliente-servidor.
* Fácil de usar, especialmente para usuarios que han utilizado sistemas de control de versiones centralizados.
* Soporte para control de acceso y permisos, permitiendo a los desarrolladores trabajar en diferentes áreas del proyecto.
* Registro completo de cambios y revisiones.
* Amplia documentación y una comunidad de usuarios activa.

***Uso en el desarrollo de software:***

Apache Subversion es ampliamente utilizado en el desarrollo de software para mantener un control efectivo sobre el código fuente y las versiones del proyecto. Los desarrolladores utilizan SVN para colaborar en el desarrollo de un proyecto y mantener un registro completo de los cambios realizados en el código fuente. SVN permite a los desarrolladores trabajar en diferentes áreas del proyecto de forma aislada, sin afectar el trabajo de los demás desarrolladores. Además, SVN ayuda a controlar el acceso y los permisos de los usuarios, lo que garantiza que solo las personas autorizadas puedan realizar cambios en el código fuente. También permite la creación de ramas, lo que permite que se trabajen diferentes versiones del mismo proyecto sin afectar a la versión principal.

***Ventajas:***

* Fácil de aprender y utilizar, especialmente para usuarios que tienen experiencia con sistemas de control de versiones centralizados.
* Amplia documentación y una comunidad de usuarios activa.
* Ofrece una excelente integración con otros sistemas de software.
* Soporte para control de acceso y permisos.
* Registro completo de cambios y revisiones.

**Desventajas:**

* No es tan rápido como otros sistemas de control de versiones distribuidos, como Git.
* No es tan flexible para la creación de ramas y fusiones como los sistemas de control de versiones distribuidos.
* Requiere un servidor centralizado, lo que puede aumentar la complejidad en entornos de equipo muy grandes.
* **BAAZAR**

Bazaar es una herramienta de software de control de versiones distribuido (DVCS, por sus siglas en inglés) que permite a los desarrolladores colaborar en proyectos de software de forma remota. Fue desarrollado por Canonical Ltd. y lanzado en el año 2005 como una alternativa al popular software de control de versiones Git.

***Características:***

* La gestión de múltiples ramas
* La capacidad de trabajar offline
* La integración con otros sistemas de control de versiones
* Una interfaz de línea de comandos fácil de usar

***Uso en el desarrollo de software:***

Bazaar se utiliza en el desarrollo de software para mantener un registro de todos los cambios realizados en el código fuente y para facilitar la colaboración entre desarrolladores. Permite a los equipos trabajar de forma remota en el mismo código fuente, realizar cambios y enviarlos de vuelta al repositorio central para su revisión y aprobación.

***Ventajas***

* Facilidad de uso
* Su capacidad para trabajar con proyectos grandes y complejos
* Su flexibilidad para adaptarse a diferentes flujos de trabajo.
* Es fácil de instalar y configurar
* Tiene una gran comunidad de usuarios y desarrolladores que ofrecen soporte y recursos

***Desventajas***

* Velocidad en operaciones grandes
* Su pobre rendimiento en proyectos muy grandes
* Su dependencia de una red para operaciones básicas
* Su falta de popularidad y recursos en comparación con otras herramientas de control de versiones como Git o Mercurial.
* **GIT**

Git es un sistema de control de versiones distribuido que se utiliza para rastrear cambios en archivos y coordinar el trabajo entre múltiples personas en un proyecto de desarrollo de software. Fue creado por Linus Torvalds en 2005 y se ha convertido en una de las herramientas más populares en la industria del desarrollo de software.

***Características***

* La gestión de múltiples ramas
* La capacidad de trabajar offline
* La integración con otros sistemas de control de versiones
* Una interfaz de línea de comandos fácil de usar

***Uso en el desarrollo de software***

Git se utiliza en el desarrollo de software para controlar versiones, facilitar la colaboración en equipo, manejar ramas de desarrollo, deshacer cambios, gestionar conflictos y trabajar con repositorios remotos. Estas características hacen que Git sea una herramienta esencial para el desarrollo eficiente y organizado de software.

***Ventajas***

* Control de versiones
* Colaboración eficiente
* Ramificación y fusión flexibles.
* Deshacer cambios y revertir a versiones anteriores
* Velocidad y rendimiento

***Desventajas***

* Curva de aprendizaje inicial
* Comandos de línea de comandos
* Resolución de conflictos
* Tamaño del repositorio

**Comparación de Herramientas**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CARACTERÍSTICAS** | **MERCURIAL** | **APACHE SUBVERSION** | **BAAZAR** | **GIT** |
| Manejo centralizado del repositorio |  | **X** |  | **X** |
| Control detallado de acceso y permisos | **X** | **X** | **X** | **X** |
| Multiplataforma | **X** | **X** | **X** | **X** |
| Historial de archivos |  | **X** |  | **X** |
| Gratuito | **X** | **X** | **X** | **X** |
| Mantenimiento de repositorios | **X** | **X** | **X** | **X** |
| Soporte de auto-reparación de repositorios |  | **X** |  | **X** |
| Gestión y fusión de rama |  |  |  | **X** |

Al realizar el análisis de Benchmarking de herramientas se ha decidido utilizar Git como la herramienta de control de versiones preferida en lugar de opciones alternativas como Mercurial, Bazaar y Apache Subversion por las siguientes razones:

* **Amplia adopción y comunidad:** Git es ampliamente utilizado y cuenta con una gran comunidad de desarrolladores que respaldan su desarrollo y proporcionan soporte continuo. Esto se traduce en una amplia gama de recursos disponibles, documentación y soluciones a problemas comunes. La comunidad activa también fomenta la creación y mejora de herramientas y complementos que mejoran la experiencia de uso de Git.
* **Rendimiento y escalabilidad**: Git está diseñado para ser rápido y eficiente, incluso en repositorios grandes con una historia extensa. Su enfoque distribuido y su capacidad para almacenar cambios de manera incremental hacen que las operaciones, como la navegación en la historia del repositorio y la búsqueda de cambios específicos, sean rápidas y eficientes. Esto es especialmente beneficioso en entornos de desarrollo ágiles y colaborativos.
* **Flexibilidad y gestión avanzada de ramas:** Git ofrece una gestión avanzada de ramas, lo que facilita la creación y fusión de ramas para trabajar en paralelo en diferentes características, experimentar con nuevas ideas y colaborar eficientemente en equipos. Esto proporciona una mayor flexibilidad y permite adaptarse a diferentes flujos de trabajo y estrategias de desarrollo.
* **Amplio ecosistema de herramientas y servicios:** Git cuenta con un amplio ecosistema de herramientas y servicios que facilitan la colaboración, la revisión de código y la integración con otros sistemas y servicios. Plataformas populares como GitHub, GitLab y Bitbucket ofrecen una interfaz gráfica amigable, características adicionales como solicitudes de extracción y sistemas de control de acceso, y promueven una colaboración más efectiva y un seguimiento mejorado de los cambios en el proyecto.
* **Capacidad de auto-reparación del repositorio:** Git proporciona mecanismos y herramientas que pueden ayudar a solucionar problemas y restaurar un repositorio en caso de daños o errores. Si bien no tiene una función automática de auto-reparación, la capacidad de verificar la integridad del repositorio, revertir a versiones anteriores o clonar nuevamente el repositorio desde una fuente remota contribuye a la recuperación eficiente en situaciones problemáticas.

Basándonos en estas consideraciones, se ha seleccionado Git como la herramienta de control de versiones principal para garantizar un control eficiente y efectivo de la configuración del software, fomentar la colaboración en equipo y facilitar el desarrollo y mantenimiento de proyectos de manera ágil y escalable.

1. **Identificación**
   1. **Clasificación de Ítem**

La clasificación de ítems es un proceso importante en la gestión de configuración y en la organización de los elementos y documentos dentro de un proyecto. La clasificación se utiliza para categorizar y agrupar los ítems de acuerdo con características similares, lo que facilita su identificación, seguimiento y control a lo largo del ciclo de vida del proyecto.

Existen diferentes enfoques para clasificar los ítems, y la elección de la clasificación adecuada dependerá del tipo de proyecto y de los requerimientos específicos. A continuación, se presentan tres categorías comunes de clasificación de ítems:

* Ítems por evolución: Estos son los ítems que están sujetos a cambios y evolución a lo largo del tiempo. Incluyen documentos y archivos ejecutables que se modifican o actualizan a medida que avanza el proyecto.
* Ítems fuente: Estos son los ítems relacionados con el código fuente y los archivos necesarios para compilar una aplicación o software. Incluyen documentos de diseño y construcción de la arquitectura, así como los archivos de código fuente. Estos ítems suelen estar asociados con el desarrollo y la programación del software.
* Ítems de soporte: Estos son los ítems relacionados con los sistemas operativos, el software base y los requisitos del proyecto. Incluyen documentos de especificación de requisitos, análisis de requisitos, especificación de la base de datos y otros documentos que respaldan el desarrollo y la implementación del proyecto.
  + 1. **Clasificación de los Ítems de la Configuración (CI).**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tipo**  **(E= Evolución, F=Fuente, S=Soporte)** | **Nombre del Ítem (CI)** | **Nomenclatura** | **Extensión** | **Proyecto** |
| E | ***Plan de Proyecto (PROJECT CHARTER)*** | SVCE-PC | .DOCX | SVCE |
| E | *Cronograma del Proyecto* | SVCE-CP | .XLS | SVCE |
| E | *Documento de Especificación de Requisitos 1* | SVCE-DER-1 | .DOCX | SVCE |
| E | *Documento de Especificación de Requisitos 2* | SVCE-DER-2 | .DOCX | SVCE |
| E | *Documento de Especificación de Requisitos 3* | SVCE-DER-3 | .DOCX | SVCE |
| E | *Documento de Especificación de Requisitos 4* | SVCE-DER-4 | .DOCX | SVCE |
| E | *Documento de Especificación de Requisitos 5* | SVCE-DER-5 | .DOCX | SVCE |
| E | *Documento de Especificación de Requisitos 6* | SVCE-DER-6 | .DOCX | SVCE |
| E | *Documento de Especificación de Requisitos 7* | SVCE-DER-7 | .DOCX | SVCE |
| E | *Documento de Especificación de Requisitos 8* | SVCE-DER-8 | .DOCX | SVCE |
| E | *Documento de Especificación de Requisitos 9* | SVCE-DER-9 | .DOCX | SVCE |
| E | *Documento de Análisis de Especificaciones de Requisitos* | SVCE-DAER | .DOCX | SVCE |
| E | *Documento de Construcción y Análisis de Arquitectura* | SVCE-DCAA | .DOCX | SVCE |
| E | *Documento del Diseño del Software 1* | SVCE-DDS-1 | .DOCX | SVCE |
| E | *Documento de Especificación de la BD* | SVCE-DBD | .DOCX | SVCE |
| F | *Documento de Verificación de hito 1* | SVCE-DVH-1 | .DOCX | SVCE |
| E | *Documento del Diseño del Software 2* | SVCE-DDS-2 | .DOCX | SVCE |
| E | *Documento del Diseño del Software 3* | SVCE-DDS-3 | .DOCX | SVCE |
| F | *Documento de Verificación de Hito 2* | SVCE-DVH-2 | .DOCX | SVCE |
| F | *Documento de Verificación de Hito 3* | SVCE-DVH-3 | .DOCX | SVCE |
| E | *Documento de Revisión y Validación de Hitos* | SVCE-DRVH | .DOCX | SVCE |
| F | *Acta de Cierre del Proyecto* | SVCE-ACP | .DOCX | SVCE |

* 1. **Definición de Nomenclaturas**

La Nomenclatura de los Ítems de la Configuración (CI) es un conjunto de reglas y convenciones utilizadas para nombrar y etiquetar los elementos de la configuración de un sistema. Estos elementos pueden ser cualquier cosa que forme parte de la configuración, como hardware, software, documentación, procedimientos, scripts, entre otros.

"La nomenclatura de los ítems de configuración debe ser concisa, descriptiva y fácilmente comprensible por cualquier miembro del equipo de desarrollo" (Pressman & Maxim, 2015, p. 420).

La CI puede variar según la organización o el proyecto específico, pero generalmente sigue una estructura jerárquica que refleja la estructura del sistema.

A continuación se evidenciará la nomenclatura establecida por la empresa para los ítems en evolución:

ACRÓNIMO DEL PROYECTO + “-”+ ACRÓNIMO DEL ELEMENTO

Ejemplo 1:

Proyecto: Sistema de Ventas de Componentes Electrónicos

Documento: Plan del Proyecto

Nomenclatura: SVCE-PP.docx

Observaciones:

* La extensión será de acuerdo al tipo de archivo.
* Si existen documentos que coinciden con la nomenclatura se agregará la siguiente letra de la palabra.

Documento: Plan del Proyecto

Nomenclatura: SVCE-PP.docx

Documento: Plan de Prueba

Nomenclatura: SVCE-PPR.docx

* Si hay archivos los cuales tienen relación en la temática y necesitan numeración se le agregará “-” + “numeración .

Documento: Especificación de Requerimiento 1

Nomenclatura: SVCE-ER-1.docx

Documento: Especificación de Requerimiento 2

Nomenclatura: SVCE-ER-2.docx

* **Número de versión:**

La primera parte del número de versión se refiere a la versión principal del software o documento. Un cambio en la versión principal generalmente significa una revisión importante o un cambio fundamental en el documento.

Ejemplo 2:

Documento: Plan del Proyecto

Nomenclatura: SVCE-PP.docx

Versión: 1.0

Después de un cambio significativo:

Documento: Plan del Proyecto

Nomenclatura: SVCE-PP.docx

Versión: 2.0

* **Número de revisión:**

La segunda parte del número de versión indica una revisión menor o una actualización del documento. Es importante porque ayuda a los desarrolladores y usuarios a identificar rápidamente qué versión de un componente o parte del software están utilizando y cuántas revisiones o cambios se han realizado en ese componente o parte en particular. También puede ayudar a los desarrolladores a rastrear el historial de cambios y revisiones de un componente o parte del software, lo que puede ser útil para fines de mantenimiento y resolución de problemas.

Ejemplo 3:

Documento: Plan del Proyecto

Nomenclatura: SVCE-PP.docx

Versión: 1.0 (revisión 0)

Versión: 1.1 (revisión 1)

Versión: 1.1 (revisión 2)

Entonces para saber en qué revisión se encuentra basta con observar el número después del punto.

**En síntesis:**

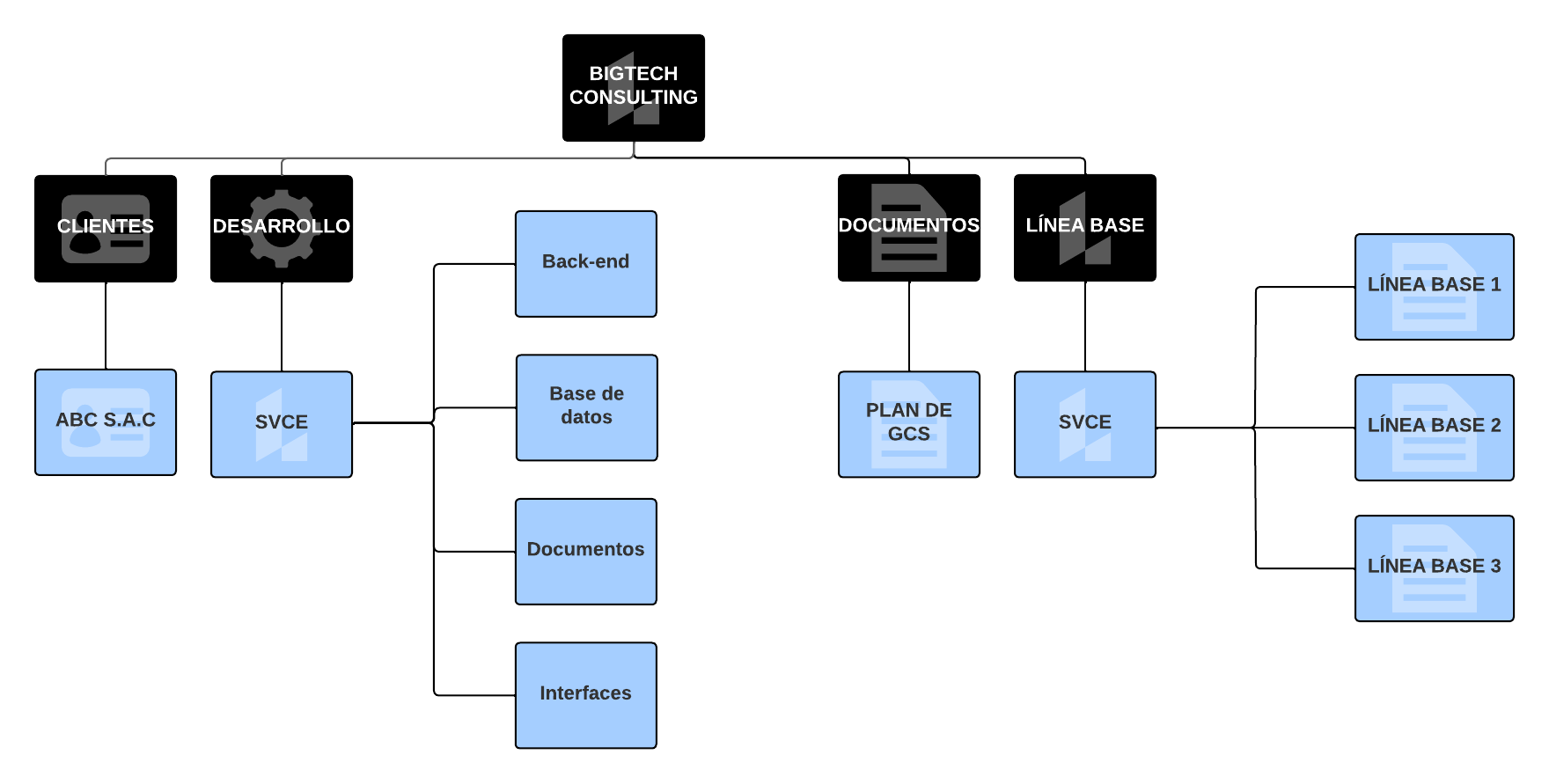
1 . 1

Número de Revisión

Número de Versión

La gestión de la configuración del software mantendrá un registro de estos documentos y sus respectivas versiones y revisiones a medida que el proyecto avance. Además, se seguirán las reglas de nomenclatura establecidas para garantizar la coherencia y comprensibilidad en la identificación de los ítems de configuración en el proyecto.

* 1. **Diseño de la estructura del repositorio (Descripción de las librerías)** En este apartado se puede encontrar graficado la estructura del repositorio de Bigtech Consulting, correctamente organizado. Esta estructura organizada es necesaria y fundamental ya que permite que el mantenimiento y comunicación en el desarrollo de software, de esta forma, se puede desarrollar y tener una idea clara de todos los proyectos que han sido desarrollados o están en proceso, por parte de la empresa Bigtech Consulting

****

* 1. **Definición de Líneas Base**

Definición de Líneas Bases

Hito 1:

Para facilitar la colaboración y el control del proyecto, se creará un repositorio centralizado que servirá como fuente única de verdad para los documentos y el código fuente. A medida que avanzamos en el Hito 1, cada miembro del equipo asumirá la responsabilidad de especificar requisitos específicos. Esto implicará la elaboración de especificaciones claras y detalladas para cada uno de ellos, como el catálogo de productos correctamente clasificados, la gestión de la base de datos, la búsqueda de clientes, el filtrado de productos, la selección y compra de artículos, la edición del listado de compras, los medios de pago, el reporte de compra y el soporte al cliente.

Además, el equipo de desarrollo contará con la contribución del DBA para diseñar una base de datos robusta y eficiente que respalde las funcionalidades del software. Asimismo, los diseñadores UX y el Arquitecto de Software colaborarán para crear un diseño detallado del software, teniendo en cuenta los requisitos establecidos. Una vez que todas estas actividades estén completas, se realizará un análisis y verificación exhaustivos del Hito 1 para asegurarnos de que se hayan alcanzado los objetivos y los entregables correspondientes. Este análisis y verificación nos permitirá garantizar la calidad y el cumplimiento de las metas establecidas.

Hito 2:

En este hito se da comienzo a la fase de programación, además la metodología que se ha adoptado es una ágil, la cual es la metodología Scrum , la cual nos proporcionará facilidades a la hora de trabajar en el proyecto.  
Respecto a los entregables tenemos: La implementación de los requisitos

* Requisito 1: Catálogo de productos correctamente clasificados.
* Requisito 2: Gestión de base de datos de los productos
* Requisito 3: Búsqueda de cliente para generar un reporte.

donde el Arquitecto de Software será el encargado de implementar los requisitos especificados en el Hito 1; luego tenemos el Diseño de Software 2 donde los diseñadores UX y el Arquitecto de Software colaborarán para crear el diseño detallado del software, teniendo en cuenta los requisitos establecidos

* Requisito 4: Búsqueda y filtrado de productos.
* Requisito 5: Selección y compra de artículos mediante un carrito de compras.
* Requisito 6: Edición del listado de compras durante la   
   compra.  
  siguiendo con el apartado de los requisitos, tenemos la Implementación de los Requisitos 4, 5 y 6 (los cuales fueron especificados anteriormente) en donde el Arquitecto de Software será el encargado de implementar los requisitos especificados en el Hito 1; mencionado anteriormente el diseño de software, continuamos con este apartado con el Diseño de Software 3, en el cual los diseñadores UX y el Arquitecto de Software colaborarán para crear el diseño detallado del software, teniendo en cuenta los requisitos establecidos
* Requisito 7: Medios de pago que confirmen la compra.
* Requisito 8: Reporte de compra que permita al usuario

solicitarlo.

* Requisito 9: Soporte al cliente para resolver cualquier   
   problema.

y para finalizar, tenemos el Análisis y Verificación del Hito 2, en este documento se realizará un análisis para asegurar que el diseño del software y la base de datos cumplan con los requisitos establecidos.

Hito 3:

En el siguiente hito del proyecto, nos enfocaremos en la implementación de los requisitos

* Requisito 7: Medios de pago que confirmen la compra.
* Requisito 8: Reporte de compra que permita al usuario

solicitarlo.

* Requisito 9: Soporte al cliente para resolver cualquier   
   problema.

asegurando que se lleven a cabo de manera efectiva. Además, realizaremos un análisis detallado y exhaustivo del hito 3 para garantizar que el diseño y la implementación cumplan con todos los requisitos establecidos en esta fase. Posteriormente, procederemos a llevar a cabo una validación integral de los hitos 1-3, revisando minuciosamente cada uno de ellos para verificar que se hayan cumplido todos los objetivos y entregables correspondientes. Finalmente, el Jefe del Proyecto se encargará de elaborar el acta de cierre del proyecto, documentando los resultados finales y cerrando de manera formal esta etapa del proyecto.

**REFERENCIAS**

**Apache Subversion. (s. f.). Apache Subversion Introduction. Apache Software Foundation. Recuperado el 23 de abril de 2023, de** [**https://subversion.apache.org/docs/introduct.html**](https://subversion.apache.org/docs/introduct.html)

**Mackall, M. (2005). Mercurial: The Definitive Guide. O'Reilly Media, Inc.**

**Mercurial (n.d.). In Mercurial. Retrieved September 15, 2021, from** [**https://www.mercurial-scm.org/**](https://www.mercurial-scm.org/)

**Leon, A., & Leon, M. (2015). Software Configuration Management Handbook. Taylor & Francis. https://www.taylorfrancis.com/books/9781317676946**

**Wheeler, D. A. (2009). Subversion versus git: Myths and facts. In Proceedings of the 2009 3rd IEEE International Conference on E-Learning in Industrial Electronics (pp. 7-11). IEEE.**

**Pressman, R. S., & Maxim, B. R. (2015). Ingeniería del software: Un enfoque práctico (7a ed.). McGraw-Hill Education.**[**http://cotana.informatica.edu.bo/downloads/ld-Ingenieria.de.software.enfoque.practico.7ed.Pressman.PDF**](http://cotana.informatica.edu.bo/downloads/ld-Ingenieria.de.software.enfoque.practico.7ed.Pressman.PDF)