

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS

SOFTWARE PARA EL CONTROL E IMPLEMENTACIÓN DE APLICACIONES GENERADAS POR EMPRESAS DE DESARROLLO

Caso de estudio: Productos base de desarrollo

Ing. FRANKLIN CANDANOZA VILLALBA
Ing. CARLOS ALBERTO ÁLVAREZ IBARRA

Director: OSWALDO ROMERO VILLALOBOS, MSc.
Revisor: SANDRO JAVIER BOLAÑOS CASTRO, Ph.D.

FACULTAD DE INGENIERÍA
ESPECIALIZACIÓN EN INGENIERÍA DE SOFWARE BOGOTÁ D.C
2016

SOFTWARE PARA EL CONTROL E IMPLEMENTACIÓN DE APLICACIONES GENERADAS POR EMPRESAS DE DESARROLLO

Caso de estudio: Productos base de desarrollo

Ing. FRANKLIN CANDANOZA VILLALBA
Ing. CARLOS ALBERTO ÁLVAREZ IBARRA

PROYECTO DE INVESTIGACIÓN PARA OPTAR POR EL TITULO DE
ESPECIALISTA EN INGENIERÍA DE SOFTWARE

Director: OSWALDO ROMERO VILLALOBOS, MSc.
Revisor: SANDRO JAVIER BOLAÑOS CASTRO, Ph.D.

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
FACULTAD DE INGENIERÍA
ESPECIALIZACIÓN EN INGENIERÍA DE SOFTWARE
BOGOTÁ D.C. 2016

Índice general

I CONTEXTUALIZACIÓN DE LA INVESTIGACIÓN 7

1. DESCRIPCIÓN DE LA INVESTIGACIÓN 8

1.1. Planteamiento/Identificación del problema . . . 8
1.1.1. Planteamiento del Problema . . . 8
1.1.2. Formulación del problema . . . 9
1.1.3. Sistematización del problema . . . 9
1.2. Objetivos . . . 9
1.2.1. Objetivo General . . . 9
1.2.2. Objetivos específicos . . . 9
1.3. Justificación . . . 10
1.3.1. Justificación práctica . . . 10
1.4. Hipótesis . . . 10
1.5. Marco Referencial . . . 11
1.5.1. Marco Teórico . . . 11
1.6. Metodología de la investigación . . . 21
1.6.1. Tipo de estudio . . . 21
1.6.2. Metodología utilizada . . . 21
1.6.3. Fuentes y técnicas para la recolección de información . . . 21
1.7. Estudio de sistemas previos . . . 22

II DESARROLLO DE LA INVESTIGACIÓN 26

2. ARQUITECTURA EMPRESARIAL 27

2.1. Arquitectura del Negocio . . . 27
2.1.1. Punto de Vista de Organización . . . 27
2.1.2. Punto de Vista de Cooperación de actor . . . 28
2.1.3. Punto de Vista de Función de Negocio . . . 28
2.1.4. Punto de Vista de Proceso de Negocio . . . 29
2.1.5. Punto de Vista de Cooperación de Proceso de Negocio . . . 31
2.1.6. Punto de Vista de Producto . . . 31
2.2. Arquitectura del Sistema de Información . . . 33
2.2.1. Punto de Vista de Comportamiento de Aplicación . . . 33
2.2.2. Punto de Vista de estructura de Aplicación . . . 33
2.2.3. Punto de Vista de uso de aplicación . . . 34
2.3. Arquitectura de tecnología . . . 35
2.3.1. Punto de Vista de Infraestructura . . . 35
2.3.2. Punto de Vista de uso de infraestructura . . . 36

2.3.3. Punto de Vista de organización e implementación 36
2.3.4. Punto de Vista de estructura de Información 37

3. FASES DEL PROCESO UNIFICADO **38**
3.1. Fase de concepción 38
3.1.1. Ingeniería de Requerimientos 38
3.1.2. Recolección de Requerimientos 38
3.1.3. Reuniones 38
3.1.4. Recolección de objetivos del Sistema 42
3.2. Fase de elaboración 45
3.2.1. Modelos de casos de uso 45
3.2.2. Modelo Relacional Base de Datos 71
3.3. fase de construcción 71
3.4. fase de transición 71
3.4.1. Implementación 71
3.4.2. Pruebas de Aceptación 73
3.5. fase de producción 75

III CIERRE DE LA INVESTIGACIÓN **83**

4. CONCLUSIONES **84**
4.1. Resultados y discusión 85
4.2. Verificación, contraste y evaluación de los objetivos 85
4.3. Síntesis del modelo propuesto 85
4.4. PROSPECTIVA DEL TRABAJO DE GRADO 86
4.4.1. Líneas de investigación futuras 86
4.4.2. Trabajos de Investigación futuros 86

Índice de figuras

1.1. Estructura del control de versiones 11
1.2. Modelo UML 14
1.3. Fases del ADM 18
1.4. Diagrama de control de versiones local. 22
1.5. Diagrama de control de versiones centralizado. 23
1.6. Diagrama de control de versiones distribuido. 24

2.1. Punto de vista de Organización 27
2.2. Punto de vista de Cooperación de Actor 28
2.3. Punto de vista de Función de Negocio 29
2.4. Punto de vista de Proceso de Negocio 30
2.5. Punto de vista de Cooperación de Proceso de Negocio 31
2.6. Punto de vista de Producto 32
2.7. Punto de Vista de Comportamiento de Aplicación 33
2.8. Punto de Vista de estructura de aplicación 34
2.9. Punto de Vista de uso de aplicación 34
2.10. Punto de vista de Infraestructura 35
2.11. Punto de Vista de uso de infraestructura 36
2.12. Punto de Vista de organización e implementación 36
2.13. Punto de Vista de estructura de Información 37

3.1. Diagrama de casos de uso: configurar proyectos 45
3.2. Prototipo: Crear nuevo proyecto 46
3.3. Prototipo: Crear conexión en nuevo proyecto 47
3.4. Prototipo: Administrar versiones 48
3.5. Prototipo: Crear versión 49
3.6. Prototipo: Editar Versión 50
3.7. Prototipo: Eliminar Versión 51
3.8. Prototipo: editar proyecto 52
3.9. Prototipo: Eliminar proyecto 53
3.10. Prototipo: Buscar proyecto 54
3.11. Prototipo: Ver control y evolución 55
3.12. Diagrama de estados: Configuración de Proyectos 56
3.13. Diagrama de casos de uso: Administración de Clientes 57
3.14. Prototipo: Crear Cliente 58
3.15. Prototipo: Crear ambiente 59
3.16. Prototipo: Editar ambiente 60
3.17. Prototipo: Eliminar Ambiente 61
3.18. Prototipo: Editar Cliente 62
3.19. Prototipo: Eliminar Cliente 62

3.20. Prototipo: Buscar Cliente . . . 63
3.21. Diagrama de estados: Administración de clientes . . . 64
3.22. Diagrama de casos de uso: Clientes y proyectos . . . 65
3.23. Prototipo: Administrar proyectos en cliente . . . 66
3.24. Prototipo: Administrar aplicaciones en ambiente . . . 67
3.25. Prototipo: Asociar aplicaciones . . . 69
3.26. Modelo Relacional Base de Datos . . . 71
3.27. Archivo evolucion.xml para registro de cambios . . . 72
3.28. Módulo de Clientes . . . 75
3.29. Registro de nuevo cliente . . . 76
3.30. Editar cliente . . . 76
3.31. Editar ambiente . . . 77
3.32. Eliminar ambiente . . . 77
3.33. Vista de Proyectos . . . 78
3.34. Registro de un nuevo Proyecto . . . 78
3.35. Editar proyecto . . . 79
3.36. Ordenar versiones de proyecto . . . 80
3.37. Eliminar un proyecto . . . 80
3.38. Control y evolución de las aplicaciones . . . 81
3.39. Documento de evolución generado por la aplicación . . . 82

Índice de cuadros

3.1. Reunión I: Problema y situación actual . . . 39
3.2. Reunión II: Problemas en las actualizaciones de aplicaciones . . . 39
3.3. Reunión III: Actualización de Componentes . . . 40
3.4. Reunión IV: Gestión y administración de cambios en los proyectos . . . 40
3.5. Reunión V: Documento de Diccionario de mensajes . . . 40
3.6. Reunión VI: Documento de implementación para los clientes . . . 41
3.7. Reunión VII: Información de Gestión de cambios . . . 41
3.8. Objetivo I: Configurar conexiones y proyectos . . . 42
3.9. Objetivo II: Administrar usuarios . . . 42
3.10. Objetivo III: Administrar perfiles de usuario . . . 43
3.11. Objetivo IV: Administrar permisos de usuario . . . 43
3.12. Objetivo V: Administrar clientes . . . 43
3.13. Objetivo VI: Generar reporte de implementación . . . 44
3.14. Caso de uso: Crear proyecto . . . 46
3.15. Caso de uso: Configurar Conexión . . . 47
3.16. Caso de uso: Administrar Versiones . . . 48
3.17. Caso de uso: Crear versión . . . 49
3.18. Caso de uso: Editar Versión . . . 50
3.19. Caso de uso: Eliminar Versión . . . 51
3.20. Caso de uso: Editar proyecto . . . 52
3.21. Caso de uso: Eliminar proyecto . . . 53
3.22. Caso de uso: Buscar proyecto . . . 54
3.23. Caso de uso: Ver control y evolución . . . 55
3.24. Caso de uso: Crear Cliente . . . 57
3.25. Caso de uso: Crear Ambiente . . . 58
3.26. Caso de uso: Editar Ambiente . . . 59
3.27. Caso de uso: Eliminar Ambiente . . . 60
3.28. Caso de uso: Editar Cliente . . . 61
3.29. Caso de uso: Eliminar Cliente . . . 62
3.30. Caso de uso: Buscar Cliente . . . 63
3.31. Caso de uso: Administrar proyectos en Cliente . . . 65
3.32. Caso de uso: Administrar aplicaciones en ambiente . . . 67
3.33. Caso de uso: Asociar aplicaciones . . . 68
3.34. Caso de uso: Desasociar aplicaciones . . . 70
3.35. Caso de prueba: CRUD Proyectos . . . 73
3.36. Caso de prueba: Control y Evolución . . . 73
3.37. Caso de prueba: CRUD Clientes . . . 74
3.38. Caso de prueba: Clientes y proyectos . . . 74

INTRODUCCIÓN

En el marco de la Especialización en Ingeniería de Software de la Universidad Distrital Francisco José de Caldas, el siguiente trabajo de grado titulado SOFTWARE PARA EL CONTROL E IMPLEMENTACIÓN DE APLICACIONES GENERADAS POR EMPRESAS DE DESARROLLO Caso de estudio: Productos base de desarrollo. Exhibe el proceso de construcción del sistema de información APPEVOLUTION, diseñado para administrar el control de cambios sobre el código fuente de un producto base de desarrollo.

El trabajo de grado apunta a solucionar problemáticas de empresas de desarrollo de software, que en su gran mayoría se enmarcan en el sector privado, pero no excluye de su implementación en entidades del sector público. Si se cuenta con un producto base de desarrollo se puede implementar nuestra solución. Ahora, pretendiendo solucionar la problemática que se presenta al documentar los procesos y el ciclo de vida del software hemos planteado una solución que permite administrar, gestionar y versionar los cambios que sufre un aplicativo. Un punto importante que desarrollamos en el presenta trabajo de grado es el enfoque de scripts en las bases de datos, pensando en futuras instalaciones del producto base de desarrollo, logrando trazabilidad entre las iteraciones que plantea la construcción de un sistema informático. Los capítulos que componen este del trabajo de grado se organizaron en tres partes:
La parte 1, Contextualización de la investigación, presenta una radiografía del problema de investigación, su planteamiento e identificación, planteamos además los objetivos del proyecto de grado su justificación e hipótesis. Abarca también el marco referencial, la metodología de investigación y el estudio de sistemas previos. En términos concretos esta parte identifica porqué es necesario desarrollar una solución informática implementando conceptos, procedimientos, técnicas, métodos y metodologías propios de la Ingeniería de Software.

La parte 2, Desarrollo de la investigación, presenta como está construida nuestra solución, en principio aplicamos conceptos de Arquitectura Empresarial al ámbito donde se implementa el sistema que planteamos, inmediatamente después, enfatizamos en el modelo del proceso y la arquitectura del software. Desarrollamos las cinco fases que la componen la metodología que propone Proceso Unificado, y con el apoyo de UML para el diseño de los diagramas realizamos la construcción como tal del sistema APPEVOLUTION.

La parte 3, Cierre de la investigación, exhibe los resultados que arroja el trabajo de grado, explicamos cuáles fueron los resultados y la discusión que se propone después desarrollar nuestra solución. Verificamos y evaluamos los objetivos y por ultimo trazamos las líneas de trabajo e investigación futuras en torno a nuestro proyecto informático.

En conclusión, hemos oportunamente aplicado lo aprendido en el transcurso de la Especialización en Ingeniería de Software, producto de esto, nuestro proyecto involucra todo el proceso de desarrollo de software, parte desde el análisis, pasando por el modelado, construcción y despliegue, esto se constata a medida que se avanza con el desarrollo del documento.

Parte I

CONTEXTUALIZACIÓN DE LA INVESTIGACIÓN

DESCRIPCIÓN DE LA INVESTIGACIÓN

1.1. Planteamiento/Identificación del problema

Durante el proceso de desarrollo de Software la administración del control de cambios sobre el código fuente requiere un gran esfuerzo que garantice un ciclo exitoso de desarrollo. Para lograr un producto de calidad, se debe tener en cuenta que, tanto en su desarrollo como en su ejecución es necesario realizar una correcta planificación, incluyendo los procesos que ello implica, para de esta forma, disminuir el riesgo de fracaso y predecir de manera precisa los costos, plazos y la calidad del producto de software.
El control de cambios se puede aplicar a muchas etapas del proceso de Desarrollo de software, desde el código fuente hasta la base de datos. Cuando los equipos y proyectos van en crecimiento, la complejidad de los sistemas aumenta, los scripts de cambios de las distintas versiones de las bases de datos, ya sea pruebas, desarrollo o producción, se vuelven más complejos de administrar, tanto a nivel de manejo de versiones, documentación de estos, como al momento de las actualizaciones en los distintos ambientes de los clientes.
Lo anterior deriva en que, cuando se realizan actualizaciones del sistema sobre los distintos ambientes se generen pérdidas de tiempo en soporte, correcciones, recursos y disminución de confiabilidad en el sistema. La incorrecta gestión de errores ocasiona además que el software genere inconsistencias no controladas destruyendo la experiencia de usuario.

1.1.1. Planteamiento del Problema

El interrogante que surge ante el control de cambios en las bases de datos es ¿Si nunca desarrollamos software sin un versionamiento del código fuente, ¿por qué desarrollamos la base de datos sin un control de cambios?
Las prácticas en el Desarrollo de Software incluyen tanto la creación como la modificación de Scripts, a medida que pasa el tiempo estos van creciendo considerablemente dependiendo de la complejidad del sistema, la confusión se presenta al partir de un Script inicial de estructura (DDL) y de datos (DML) y a medida que surgen nuevos desarrollos terminar con un script incoherente entre la versión del aplicativo y su base de datos, agregar scripts a un único control de cambios es un error común que trae más penas que glorias.
La generación constante de scripts conlleva de por si confusión y errores en la construcción de una aplicación, pues no se evalúa con exactitud que scripts se han ejecutado y cuales faltan por ejecutar en cada uno de los entornos que se estén trabajando. No se puede garantizar que una base de datos esté en una versión concreta por los cambios constantes que se realizan sobre esta.
Las salidas a producción de actualizaciones del sistema generan errores porque en muchas ocasiones la base de datos no está en la versión correcta de acuerdo al Build del sistema que se desea actualizar. El build de la aplicación no concuerda con la versión de la base de datos y por ende genera gastos de operación, soporte, recursos e incumplimiento de los niveles de servicio ofrecidos a los distintos clientes dada

la cantidad de errores que surgen en la aplicación y que interrumpen la correcta operación de las entidades por el tiempo en que sean corregidos los errores que sean detectados después de la actualización.

1.1.2. Formulación del problema

¿En qué medida la implementación de un software de control de versiones de base de datos, facilitará y disminuirá los tiempos y errores generados por actualizaciones sobre ambientes de pruebas, desarrollo y producción de los clientes de las empresas desarrolladoras de software que cuentan con un producto base de desarrollo?

1.1.3. Sistematización del problema

- ¿Qué solución tecnológica sería la apropiada para contrarrestar los problemas del que surgen en el sistema después de realizar actualizaciones sobre las versiones de la base de datos de los clientes?
- ¿Por qué se hace necesario el control de las distintas versiones de la base de datos de los clientes?
- ¿De qué forma podríamos volver a un estado anterior de la base de datos de los clientes en caso de que suceda algún error no controlado en la aplicación?
- ¿De qué forma una solución de software podría disminuir los tiempos de actualizaciones de los aplicativos en los distintos ambientes de los clientes minimizando los errores que surgen por instalar versión errada de la base de datos y por ende ocasionando problemas en el correcto funcionamiento correcto de la aplicación?

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar un prototipo de software dirigido a empresas desarrolladoras de software que cuentan con un producto base de desarrollo, que permita el control de la evolución e implementación de las aplicaciones de software generadas con su respectivo versionamiento.

1.2.2. Objetivos específicos

- Controlar tanto las estructuras de objetos como los datos creados, durante el desarrollo de las aplicaciones, a través del versionamiento de los script de cambios que garantice un buen funcionamiento de las aplicaciones actualizadas que son soportadas por dichas bases de datos.
- Diseñar el flujo del proceso de control de cambios a través de herramientas especializadas que permitan llevar seguimiento de la evolución de las aplicaciones en cada punto específico en el tiempo y su versión.
- Diseñar una aplicación de administración para el control de la evolución e implementación de las aplicaciones basado en los scripts de cambios de base de datos, mejoras a las aplicaciones, nuevas funcionalidades y errores que se corrigen para cada versión de estas.

1.3. Justificación

1.3.1. Justificación práctica

La generación constante de versiones en las aplicaciones puede llevar a confusión y errores en estas dado que no se llevan un claro control sobre cada versión que se libera, es decir, no se tiene claro que errores corrige un build específico, que funcionalidades nuevas incluye, que mejoras tiene y que configuraciones se requieren para realizar dicha actualización. En muchas ocasiones, el cliente no tiene claro que se le entrega para poder verificar y sacar el mayor provecho a la versión o aplicación que se está liberando.
Cuando un Build de la aplicación o aplicaciones es liberado se requiere tener claro que cambios hay en esta para así permitir actualizaciones en los clientes que la requieran, es decir, se requiere de un documento de implementación que facilite dicha labor identificando que debo hacer, que debo revisar y que debo configurar para llevar a cabo la actualización o instalación de la aplicación para que esta funcione correctamente.
En este orden de ideas, la ejecución de esta investigación va permitir simplificar significativamente los errores presentados en los diferentes ambientes (desarrollo, pruebas y producción) de los sistemas de las organizaciones por la falta de control sobre la evolución de estas garantizando así una mejor implementación e identificación de los cambios y los ajustes requeridos para dichas actualizaciones o instalaciones desde cero.
De igual forma, el cliente tendrá netamente claro que cambios tiene la versión que le están instalando y así podrá revisar, validar y analizar los cambios para sacarles el máximo provecho.

1.4. Hipótesis

El SOFTWARE PARA EL CONTROL E IMPLEMENTACIÓN DE APLICACIONES GENERADAS POR EMPRESAS DE DESARROLLO. Permitirá a las empresas desarrolladoras de software, que tienen un producto base de desarrollo, controlar los cambios realizados sobre las bases de datos de los clientes con su respectiva versión y por ende actualizar las diferentes aplicaciones y su respectivo versionamiento de estructura y datos entre los diferentes ambientes, ya sea pruebas, desarrollo o producción, con la versión correcta de base de datos, evitando así posibles fallas del sistema dadas las inconsistencias por no compatibilidad entre la aplicación y la estructura de datos que requiere para su correcto funcionamiento.
De igual forma permitirá controlar la evolución de las aplicaciones a medida que se van realizando diferentes versiones sobre estas, es decir, que permitirá identificar que cambios o ajustes se realizan entre una versión y otra facilitando así la implementación en los clientes que están interasados en los ítems de nuevas funcionalidades, mejoras, errores corregidos que vienen con la versión de la aplicación a instalar.

1.5. Marco Referencial

1.5.1. Marco Teórico

Dentro de los conceptos a tener en cuenta para comprender lo que se busca con el presente proyecto de grado se encuentran puntos claves como: control de versiones, gestión de la configuración, liquiBase, sistema de control de versiones, UML y Proceso Unificado, ademas hemos aplicado referencias propias del marco de trabajo TOGAF y Archimate. Todos los anteriores son conceptos amplios, para el desarrollo del marco referencial se abordarán de forma general teniendo en cuenta sus aspectos más importantes.

- **Control de Versiones**

  Un sistema de control de versiones es una herramienta que registra todos los cambios hechos en uno o más proyectos, guardando así versiones del producto en todas sus fases del desarrollo. Las versiones son como fotografías que registran su estado en ese momento del tiempo y se van guardando a medida que se hacen modificaciones al código o a lo que se desea versionar. Si llevamos un control de versiones podemos volver a estados anteriores, volver a un punto específico en el tiempo en el que queremos poner nuestro estado actual y garantizará el buen funcionamiento de lo que se esté versionando. [Garzás, 2016]

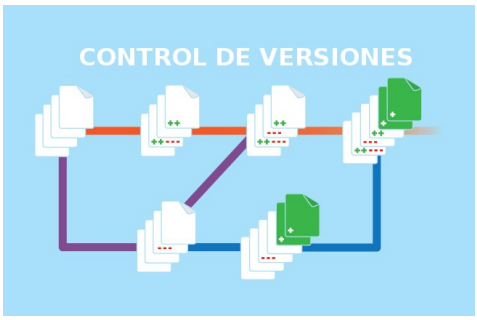


Figura 1.1: Estructura del control de versiones

  a. **Trunk**: El trunk o tronco es el que marca el desarrollo del proyecto, la versión principal. En proyectos pequeños en los que sólo hay una rama, ésta es el tronco. Es un criterio bastante común que contenido del tronco del repositorio, que no el de la copia de trabajo, sea funcional. Por ello se entiende que en el caso de documentación no haya ninguna sección a medias o en el caso de código que todo compile y/o funcione.

  b. **Branches**: El directorio branches o ramas es el que contiene todas las derivaciones del proyecto que contiene el tronco y que no pueden convivir en la rama principal.

  c. **Tags**: Tag podría traducirse como hito. Cuando la versión trunk llega a una versión mayor o menor, es decir, un estado en el que podría recibir el calificativo de completa; pasa al directorio tags. En él no se realizan cambios, es un almacén donde se guardan algunas versiones de valor histórico. [Borrell, 2016]

- **Gestión de la Configuración** La gestión de la configuración (y de los activos) es el conjunto de procesos destinados a asegurar la calidad de todo producto obtenido durante cualquiera de las etapas del desarrollo de un sistema de información (SI), a través del estricto control de los cambios realizados sobre los mismos y de la disponibilidad constante de una versión estable de cada elemento para toda persona involucrada en el citado desarrollo.

Estos dos elementos, el control de cambios y control de versiones de todos los elementos del SI, facilitan también el mantenimiento de los sistemas al proporcionar una imagen detallada del sistema en cada etapa del desarrollo. La gestión de la configuración se realiza durante todas las fases del desarrollo de un sistema de información, incluyendo el mantenimiento y control de cambios, una vez realizada la puesta en producción. ¿Y porque es importante la gestión de la configuración? El proceso de gestión de configuración tiene como principal objetivo asegurar la integridad de los productos y servicios desarrollados. Integridad del producto es:

- Saber exactamente lo que se ha entregado al cliente
- Saber el estado y contenido de las líneas base y elementos de configuración

La gestión de la configuración es una forma efectiva y eficiente de gestionar y comunicar los cambios en líneas base y elementos de configuración a lo largo del ciclo de vida. A continuación, se resaltan algunos beneficios de la implementación del proceso de gestión de configuración para la organización. Los siguientes puntos representan objetivos de negocio, por ejemplo: reducción de riesgos, mejora de la calidad y beneficios de coste en la entrega y soporte de productos.

a. Asegurar la correcta configuración del software.
b. Proporcionar la capacidad de controlar los cambios.
c. Reducir los sobreesfuerzos causados por los problemas de integridad.
d. Garantizar que todo el equipo trabaja sobre una misma línea base de productos.

[Pérez, 2014]

☛ **Liquibase**

- Que es Liquibase? Liquibase es una librería open-source basada en Java que se encarga de la gestión y aplicación de cambios que se producen en nuestra base de datos independientemente de cual sea la que se use (base de datos relacionales). La idea central de esta librería es que cada uno de los miembros del equipo puedan contar con la información relacionada con los distintos cambios que sufrió la base de datos, quien fue la persona que los realizo y en que fecha o versión fueron hechos.

  Ademas esta librería ha sido diseñada para poder ser utilizada por linea de comandos pero en las ultimas versiones de la misma se integra fácilmente con Maven (en lo que es entornos Java), de igual manera la libreria no esta pensaba para ser utilizaba exclusivamente para aplicaciones Java sino que por el contrario la idea es que pueda ser utilizada sin importar que usemos para crear nuestra aplicación.

- Caracteristicas
  a. Soporta múltiples desarrolladores
  b. Soporta multiples tipos de base de datos
  c. Soporta varios formatos para escribir los cambios (XML, YAML, JSON y SQL)
  d. No requiere una conexión activa con la base de datos para realizar los cambios.
  e. Genera documentación acerca de los cambios que se realizo (existen 2 tablas que se encargan de guardar toda la información de los cambios realizados).
  f. Permite deshacer cambios ya realizados.

- ¿Como funciona? Liquibase utiliza archivos XML en los cuales se describen los distintos cambios que se realizaran, también conocidos como changesets. Cada uno de estos changeset se identifican internamente por medio de un id, el autor del cambio como así también el nombre del archivo.

Ahora bien cuando Liquibase aplica un changeset aparte de realizar las modificaciones pertinentes en la base de datos registra en la misma cuales fueron los changeset que se ejecutaron. Esta información se almacena en la tabla "DatabaseChangeLog" la cual es generada por Liquibase y es esta librería la que se encarga de todas las operaciones sobre la misma.
[TERASWAP, 2016]

☛ **UML**

UML Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano"del sistema (modelo), incluyendo aspectos conceptuales tales como procesos, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

Es importante remarcar que UML es un "lenguaje de modelado"para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Se puede aplicar en el desarrollo de software gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional o RUP), pero no especifica en sí mismo qué metodología o proceso usar.

UML no puede compararse con la programación estructurada, pues UML significa Lenguaje Unificado de Modelado, no es programación, solo se diagrama la realidad de una utilización en un requerimiento. Mientras que, programación estructurada, es una forma de programar como lo es la orientación a objetos, la programación orientada a objetos viene siendo un complemento perfecto de UML, pero no por eso se toma UML sólo para lenguajes orientados a objetos.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas.

Los principales beneficios de UML son:

- Mejores tiempos totales de desarrollo (de 50 por ciento o más).
- Modelar sistemas (y no sólo de software) utilizando conceptos orientados a objetos.
- Establecer conceptos y artefactos ejecutables.
- Encaminar el desarrollo del escalamiento en sistemas complejos de misión crítica.
- Crear un lenguaje de modelado utilizado tanto por humanos como por máquinas.
- Mejor soporte a la planeación y al control de proyectos.
- Alta reutilización y minimización de costos.

UML es un lenguaje para hacer modelos y es independiente de los métodos de análisis y diseño. Existen diferencias importantes entre un método y un lenguaje de modelado. Un método es una manera explícita de estructurar el pensamiento y las acciones de cada individuo. Además, el método le dice al usuario qué hacer, cómo hacerlo, cuándo hacerlo y por qué hacerlo; mientras que el lenguaje de modelado carece de estas instrucciones. Los métodos contienen modelos y esos modelos son utilizados para describir algo y comunicar los resultados del uso del método.

Un modelo es expresado en un lenguaje de modelado. Un lenguaje de modelado consiste de vistas, diagramas, elementos de modelo (los símbolos utilizados en los modelos) y un conjunto de mecanismos generales o reglas que indican cómo utilizar los elementos. Las reglas son sintácticas, semánticas y pragmáticas.

Las vistas: Las vistas muestran diferentes aspectos del sistema modelado. Una vista no es una gráfica, pero sí una abstracción que consiste en un número de diagramas y todos esos diagramas

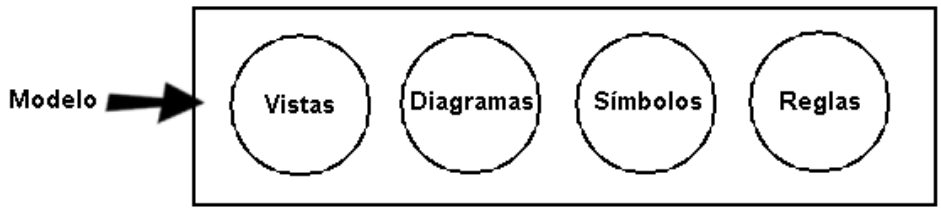


Figura 1.2: Modelo UML

juntos muestran una "fotografíaçompleta del sistema. Las vistas también ligan el lenguaje de modelado a los métodos o procesos elegidos para el desarrollo. Las diferentes vistas que UML tiene son:

- Vista Use-Case: Una vista que muestra la funcionalidad del sistema como la perciben los actores externos.
- Vista Lógica: Muestra cómo se diseña la funcionalidad dentro del sistema, en términos de la estructura estática y la conducta dinámica del sistema.
- Vista de Componentes: Muestra la organización de los componentes de código.
- Vista Concurrente: Muestra la concurrencia en el sistema, direccionando los problemas con la comunicación y sincronización que están presentes en un sistema concurrente.
- Vista de Distribución: muestra la distribución del sistema en la arquitectura física con computadoras y dispositivos llamados nodos.

Diagramas: Los diagramas son las gráficas que describen el contenido de una vista. UML tiene nueve tipos de diagramas que son utilizados en combinación para proveer todas las vistas de un sistema: diagramas de caso de uso, de clases, de objetos, de estados, de secuencia, de colaboración, de actividad, de componentes y de distribución.

Símbolos o Elementos de modelo: Los conceptos utilizados en los diagramas son los elementos de modelo que representan conceptos comunes orientados a objetos, tales como clases, objetos y mensajes, y las relaciones entre estos conceptos incluyendo la asociación, dependencia y generalización. Un elemento de modelo es utilizado en varios diagramas diferentes, pero siempre tiene el mismo significado y simbología.

Reglas o Mecanismos generales: Proveen comentarios extras, información o semántica acerca del elemento de modelo; además proveen mecanismos de extensión para adaptar o extender UML a un método o proceso específico, organización o usuario. [Studylib, 2016]

Fases de Desarrollo de un Sistema Las fases del desarrollo de sistemas que soporta UML son: Análisis de requerimientos, Análisis, Diseño, Programación y Pruebas.

1. **Análisis de requerimientos**
   UML tiene casos de uso (use-cases) para capturar los requerimientos del cliente. A través del modelado de casos de uso, los actores externos que tienen interés en el sistema son modelados con la funcionalidad que ellos requieren del sistema (los casos de uso). Los actores y los casos de uso son modelados con relaciones y tienen asociaciones entre ellos o éstas son divididas en jerarquías. Los actores y casos de uso son descritos en un diagrama use-case. Cada use-case es descrito en texto y especifica los requerimientos del cliente: lo que él (o ella) espera del sistema sin considerar la funcionalidad que se implementará. Un análisis de requerimientos puede ser realizado también para procesos de negocios, no solamente para sistemas de software.

2. **Análisis**
   La fase de análisis abarca las abstracciones primarias (clases y objetos) y mecanismos que están presentes en el dominio del problema. Las clases que se modelan son identificadas, con sus relaciones y descritas en un diagrama de clases. Las colaboraciones entre las clases para ejecutar los casos de uso también se consideran en esta fase a través de los modelos dinámicos en UML. Es importante notar que sólo se consideran clases que están en el dominio del problema (conceptos del mundo real) y todavía no se consideran clases que definen detalles y soluciones en el sistema de software, tales como clases para interfaces de usuario, bases de datos, comunicaciones, concurrencia, etc.
3. **Diseño**
   En la fase de diseño, el resultado del análisis es expandido a una solución técnica. Se agregan nuevas clases que proveen de la infraestructura técnica: interfaces de usuario, manejo de bases de datos para almacenar objetos en una base de datos, comunicaciones con otros sistemas, etc. Las clases de dominio del problema del análisis son agregadas en esta fase. El diseño resulta en especificaciones detalladas para la fase de programación.
4. **Programación**
   En esta fase las clases del diseño son convertidas a código en un lenguaje de programación orientado a objetos. Cuando se crean los modelos de análisis y diseño en UML, lo más aconsejable es trasladar mentalmente esos modelos a código.
5. **Pruebas**
   Normalmente, un sistema es tratado en pruebas de unidades, pruebas de integración, pruebas de sistema, pruebas de aceptación, etc. Las pruebas de unidades se realizan a clases individuales o a un grupo de clases y son típicamente ejecutadas por el programador. Las pruebas de integración integran componentes y clases en orden para verificar que se ejecutan como se especificó. Las pruebas de sistema ven al sistema como una çaja negraz validan que el sistema tenga la funcionalidad final que le usuario final espera. Las pruebas de aceptación conducidas por el cliente verifican que el sistema satisface los requerimientos y son similares a las pruebas de sistema.

[ERIKSSON and PENKER, 2016]

☛ **Proceso Unificado**
En cierto modo, el proceso unificado es un intento por obtener los mejores rasgos y características de los modelos tradicionales del proceso del software, pero en forma que implemente muchos de los mejores principios del desarrollo ágil de software. El proceso unificado reconoce la importancia de la comunicación con el cliente y los métodos directos para describir su punto de vista respecto de un sistema (el caso de uso). Hace énfasis en la importancia de la arquitectura del software y "ayuda a que el arquitecto se centre en las metas correctas, tales como que sea comprensible, permita cambios futuros y la reutilización" [Jac99]: Sugiere un flujo del proceso iterativo e incremental, lo que da la sensación evolutiva que resulta esencial en el desarrollo moderno del software.

Fases del proceso unificado
Al principio de este capítulo se estudiaron cinco actividades estructurales generales y se dijo que podían usarse para describir cualquier modelo de proceso del software. El proceso unificado no es la excepción. La figura 2.9 ilustra las "fases" del PU y las relaciona con las actividades generales estudiadas en el capítulo 1 y al inicio de éste.
La fase de concepción del PU agrupa actividades tanto de comunicación con el cliente como de planeación. Al colaborar con los participantes, se identifican los requerimientos del negocio, se propone una arquitectura aproximada para el sistema y se desarrolla un plan para la naturaleza iterativa e incremental del proyecto en cuestión. Los requerimientos fundamentales del negocio se

describen por medio de un conjunto de casos de uso preliminares que detallan las características y funciones que desea cada clase principal de usuarios. En este punto, la arquitectura no es más que un lineamiento tentativo de subsistemas principales y la función y rasgos que tienen. La arquitectura se mejorará después y se expandirá en un conjunto de modelos que representarán distintos puntos de vista del sistema. La planeación identifica los recursos, evalúa los riesgos principales, define un programa de actividades y establece una base para las fases que se van a aplicar a medida que avanza el incremento del software.

La fase de elaboración incluye las actividades de comunicación y modelado del modelo general del proceso. La elaboración mejora y amplía los casos de uso preliminares desarrollados como parte de la fase de concepción y aumenta la representación de la arquitectura para incluir cinco puntos de vista distintos del software: los modelos del caso de uso, de requerimientos, del diseño, de la implementación y del despliegue. En ciertos casos, la elaboración crea una "línea de base de la arquitectura ejecutable" [Arl02] que representa un sistema ejecutable de "primer corte".20 La línea de base de la arquitectura demuestra la viabilidad de ésta, pero no proporciona todas las características y funciones que se requieren para usar el sistema. Además, al terminar la fase de elaboración se revisa con cuidado el plan a fin de asegurar que el alcance, riesgos y fechas de entrega siguen siendo razonables. Es frecuente que en este momento se hagan modificaciones al plan.

La fase de construcción del PU es idéntica a la actividad de construcción definida para el proceso general del software. Con el uso del modelo de arquitectura como entrada, la fase de construcción desarrolla o adquiere los componentes del software que harán que cada caso de uso sea operativo para los usuarios finales. Para lograrlo, se completan los modelos de requerimientos y diseño que se comenzaron durante la fase de elaboración, a fin de que reflejen la versión final del incremento de software. Después se implementan en código fuente todas las características y funciones necesarias para el incremento de software (por ejemplo, el lanzamiento). A medida de que se implementan los componentes, se diseñan y efectúan pruebas unitarias21 para cada uno. Además, se realizan actividades de integración (ensamble de componentes y pruebas de integración). Se emplean casos de uso para obtener un grupo de pruebas de aceptación que se ejecutan antes de comenzar la siguiente fase del PU.

La fase de transición del PU incluye las últimas etapas de la actividad general de construcción y la primera parte de la actividad de despliegue general (entrega y retroalimentación). Se da el software a los usuarios finales para las pruebas beta, quienes reportan tanto los defectos como los cambios necesarios. Además, el equipo de software genera la información de apoyo necesaria (por ejemplo, manuales de usuario, guías de solución de problemas, procedimientos de instalación, etc.) que se requiere para el lanzamiento. Al finalizar la fase de transición, el software incrementado se convierte en un producto utilizable que se lanza.

La fase de producción del PU coincide con la actividad de despliegue del proceso general. Durante esta fase, se vigila el uso que se da al software, se brinda apoyo para el ambiente de operación (infraestructura) y se reportan defectos y solicitudes de cambio para su evaluación.

Es probable que al mismo tiempo que se llevan a cabo las fases de construcción, transición y producción, comience el trabajo sobre el siguiente incremento del software. Esto significa que las cinco fases del PU no ocurren en secuencia sino que concurren en forma escalonada.

El flujo de trabajo de la ingeniería de software está distribuido a través de todas las fases del PU. En el contexto de éste, un flujo de trabajo es análogo al conjunto de tareas (que ya se describió en este capítulo). Es decir, un flujo de trabajo identifica las tareas necesarias para completar una acción importante de la ingeniería de software y los productos de trabajo que se generan como consecuencia de la terminación exitosa de aquéllas. Debe notarse que no toda tarea identificada para el flujo de trabajo del PU es realizada en todos los proyectos de software. El equipo adapta el proceso (acciones, tareas, subtareas y productos del trabajo) a fin de que cumpla sus necesidades.

[Pressman, 2010]

☛ TOGAF

Método de Desarrollo de la Arquitectura ADM

¿Que es el ADM? El ADM es el resultado de las contribuciones de numerosos profesionales de la arquitectura y constituye el núcleo de TOGAF. Es un método para obtener Arquitecturas empresariales que son específicas para la organización, y está especialmente diseñado para responder a los requerimientos del negocio. El ADM describe:

- Un modo confiable y probado para desarrollar y utilizar una Arquitectura Empresarial
- Un método para desarrollar arquitecturas en diferentes niveles (negocio, aplicaciones, datos tecnología) que permiten al arquitecto asegurar que un conjunto complejo de requerimientos se aborden adecuadamente.
- Un conjunto de guías y técnicas para el desarrollo de arquitectura.

¿Cuáles son las fases del ADM? El ADM consiste en varias fases que se desplazan cíclicamente a través de una serie de dominios y Arquitectura y permiten al arquitecto asegurar que un conjunto complejo de requerimientos se aborden adecuadamente. La estructura básica es la siguiente:

Las fases dentro del ADM son las siguiente:

- **Fase preliminar:** En este capítulo se describen las actividades de preparación e iniciación necesarias para cumplir la directiva de negocio para una nueva arquitectura de la empresa, incluyendo la definición de un marco de la Organización de una arquitectura específica y la definición de principios.

  Los objetivos de esta fase son:

  1. Determinar la capacidad Arquitectura deseada por la organización:
     - Revisar el contexto de la organización para la realización de la arquitectura empresarial.
     - Identificar el alcance de los elementos de las organizaciones empresariales afectadas por la Capacidad de Arquitectura
     - Identificar los marcos establecidos, métodos y procesos que se cruzan con la capacidad de Arquitectura
     - Establecer destino para la madurez de capacidad
  2. Establecer la Capacidad de Arquitectura:
     - Definir y establecer el modelo de organización de la Arquitectura Empresarial
     - Definir y establecer el proceso detallado y recursos para la gobernanza de la arquitectura
     - Seleccionar y aplicar herramientas que apoyan la capacidad de Arquitectura
     - Definir los principios de la arquitectura
- **Fase A: Visión de Arquitectura:** describe la fase inicial de un ciclo de desarrollo de la arquitectura. Incluye información sobre cómo definir el alcance de la iniciativa de desarrollo de la arquitectura, la identificación de las partes interesadas, la creación de visión de arquitectura, y obtener la aprobación para proceder con el desarrollo de la arquitectura.

  Los objetivos de esta fase son:

  1. Desarrollar una visión aspiracional de alto nivel de las capacidades y el valor del negocio para ser entregados como resultado de la arquitectura de la empresa propuesta

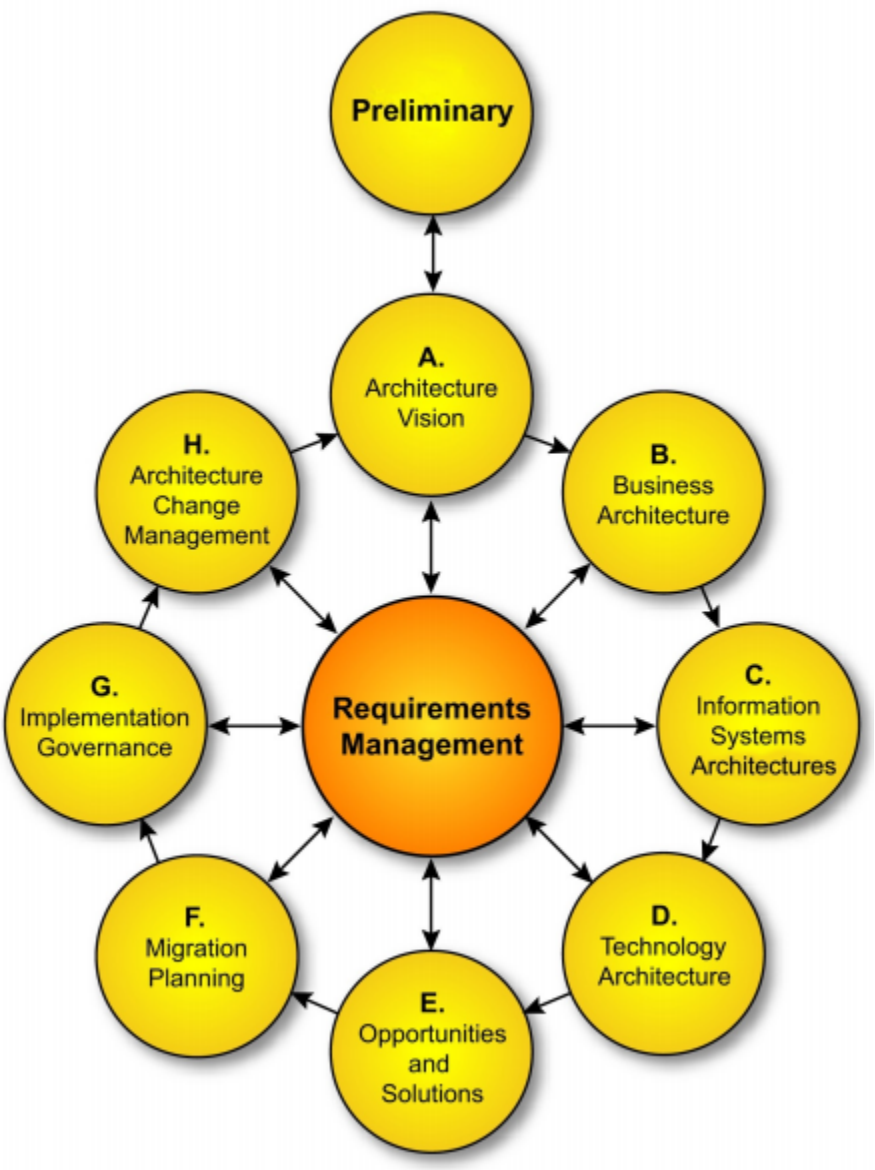


Figura 1.3: Fases del ADM

2. Obtener la aprobación de una Declaración de un trabajo de Arquitectura que define un programa de trabajos para desarrollar e implementar la arquitectura como se establece en la Vision de Arquitectura

- **Fase B: Arquitectura de Negocios:** describe el desarrollo de una arquitectura de negocios para apoyar el acuerdo de Visión de Arquitectura. Los objetivos de esta fase son:
  1. Desarrollar la arquitectura destino de negocios que describe cómo la empresa la necesita para operar para lograr los objetivos de negocio y responder a los conductos estratégicos establecidos en la visión de Arquitectura
  2. Identificar los componentes de la Hoja de Ruta de Arquitectura candidatos sobre la base de las brechas entre la línea de base y objetivo de negocio de Arquitecturas
- **Fase C: Arquitectura de Sistemas de información:** describe el desarrollo de Arquitectura de Sistemas de Información para apoyar el acuerdo Visión de arquitectura. Los objetivos de esta fase son:
  1. Desarrollar los sistemas de información del objetivo (datos y aplicaciones) de la Arquitectura, describiendo cómo los Sistemas de Información de la empresa permitirá a la Arquitectura de Negocios y a la visión de arquitectura.
  2. Identificar los componentes de la Arquitectura de hoja de ruta candidatos sobre la base de las brechas entre las arquitecturas de referencia y sistemas de información del objetivo
- **Fase D: Arquitectura de tecnología:** describe el desarrollo de la arquitectura de la tecnología para apoyar el acuerdo de Visión de Arquitectura. Los objetivos de esta fase son:
  1. Desarrollar la Arquitectura Tecnológica Objetivo que permite la aplicación lógica y física y los componentes de datos y la visión de Arquitectura, dirigiéndose a la Solicitud de trabajo de arquitectura y las preocupaciones de los interesados.
  2. Identificar los componentes de la Hoja de Ruta candidatos sobre la base de las brechas entre la tecnología objetivo y la Arquitecturas de referencia
- **Fase E: Oportunidades y Soluciones:** lleva a cabo la planificación de la implementación inicial. Los objetivos de esta fase son:
  1. Generar la versión completa inicial de la Hoja de Ruta de la Arquitectura, en base al análisis de las deficiencias de los componentes de la hoja de ruta de las fase B, C y D
  2. Determinar si se requiere un enfoque gradual, y si es así identificar las arquitecturas de transición que ofrecen un valor empresarial continuo
- **Fase F: Planeación de la migración:** aborda cómo pasar de la línea de base a las arquitecturas objetivo al finalizar una implementación detallada y Plan de Migración. Los objetivos de esta fase son:
  1. Finalizar la Hoja de ruta de la arquitectura y la aplicación de soporte y Plan de Migración
  2. Asegurarse de que la aplicación y el Plan de Migración se coordina con el enfoque de la empresa para la gestión y la implementación de cambios en la cartera de cambios generales de la empresa
  3. Asegurarse de que el valor para el negocio y el costo de los paquetes de trabajo y la arquitectura de transicción sea entendida por las partes interesadas
- **Fase G: Gobierno de aplicación:** proporciona una supervisión de arquitectura de la aplicación. Los objetivos de esta fase son:
  1. Asegurar la conformidad con la arquitectura destino por los proyectos de implementación

2. Realizar funciones de gobierno de arquitectura para la solución y cualquier solicitud de cambio

- **Fase H: Arquitectura de Gestión del cambio:** establece los procedimientos para la gestión del cambio a la nueva arquitectura. Los objetivos de esta fase son:
  1. Asegurarse de que se mantiene el ciclo de vida de la arquitectura
  2. Asegurarse de que se ejecute el Marco de Gobierno Arquitectura
  3. Asegurarse de que la capacidad de arquitectura de la empresa cumple los requisitos actuales
- **Gestión de requisitos:** examina el proceso de gestión de requisitos de arquitectura en todo el ADM. Los objetivos de esta fase son:
  1. Asegurarse de que el proceso de gestión de requisitos es sostenido y funciona para todas las fases pertinentes de ADM
  2. Gestionar requisitos de arquitectura identificados durante cualquier ejecución del ciclo ADM o una fase
  3. Asegurarse de que los requisitos de arquitectura relevantes están disponibles para uso de cada fase que se ejecuta

Conceptos Generales Sistema Es una colección de componentes organizados para llevar a cabo una función o conjunto de funciones específicas

Arquitectura Es la organización fundamental del sistema, encarnada en sus componentes, sus relaciones entre sí y con el medio ambiente, y los principios que guían su diseño y evolución.

Descripcion de la Arquitectura Es una colección de artefactos que documentan una arquitectura. En TOGAF, vistas de arquitectura son los artefactos claves en una descripción de la arquitectura.

Partes interesadas Son personas que tienen un papel clave en, o preocupaciones sobre el sistema; por ejemplo, como usuarios, desarrolladores o administradores.Diferentes actores con diferentes roles en el sistema tendrán diferentes inquietudes. Las partes interesadas pueden ser individuos, grupos u organizaciones (o clases de los mismos).

Preocupación Son los intereses dominantes que son de crucial importancia para las partes interesadas en el sistema, y determinar la aceptabilidad del sistema. Las preocupaciones pueden referirse a cualquier aspecto de funcionamiento del sistema, el desarrollo o la operación, incluyendo consideraciones tales como el rendimiento, la fiabilidad, la seguridad, la distribución y capacidad de evolución.

Vista Es una representación de todo un sistema desde la perspectiva de un conjunto relacionado de preocupaciones. Al capturar o representar el diseño de un sistema de arquitectura, el arquitecto normalmente crear uno o más modelos de arquitectura, posiblemente utilizando diferentes herramientas. Una vista comprenderá partes seleccionadas de uno o más modelos, elegidos con el fin de demostrar a un actor o grupo de actores que sus preocupaciones están siendo abordadas adecuadamente en el diseño de la arquitectura del sistema.

Punto de vista define la perspectiva desde la cual se toma una vista. Más específicamente, un punto de vista define: cómo construir y utilizar un punto de vista (por medio de un esquema o plantilla adecuada); la información que debe aparecer en la vista; las técnicas de modelado para expresar y analizar la información; y una justificación de estas decisiones (por ejemplo, describiendo el propósito y destinatario de la vista).

[Group, 2013]

1.6. Metodología de la investigación

1.6.1. Tipo de estudio

El presente proyecto de grado tiene características de un estudio Proyectivo, ya que según la bibliografía consultada y la propuesta del prototipo que se desea desarrollar, se presentaran resultados que serán advertidos por las empresas al momento de realizar actualizaciones de las aplicaciones en los distintos ambientes (pruebas, desarrollo o producción) partiendo de la premisa de optimizar los tiempos ejecución y la correcta compatibilidad entre la aplicación y la respectiva versión de base de datos, garantizando así un exitosa construcción y documentación de un sistema informático.
De igual forma, es absolutamente necesario tener en cuenta que, al versionar y llevar el control de cambios de la base de datos, se optimizan los tiempos en soporte y correcciones, estos tiempos serán utilizados para otras tareas por parte del grupo de desarrollo y las áreas implicadas en el desarrollo, mantenimiento e implementación de aplicaciones.

1.6.2. Metodología utilizada

Después de indagar que metodologías se adaptan a la solución que planteamos, decidimos utilizar las fases de construcción que plantea la metodología Proceso Unificado. Inicialmente se modeló el negocio, se obtuvieron los requisitos o requerimientos y posteriormente se realizó el diseño, la implementación, pruebas y por último el despliegue de la aplicación.
Adicional a esto, se tuvieron en cuenta aspectos para el desarrollo del sistema de información que garantizan un producto de calidad, características como usabilidad, funcionalidad, seguridad e interoperabilidad para permitir medir la calidad del producto desarrollado.

1.6.3. Fuentes y técnicas para la recolección de información

Las fuentes de información que permitieron que el presente proyecto de grado haya sido ejecutado fueron: entrevistas a usuarios, revisión de documentos e informes realizados sobre experiencias pasadas de los equipos de desarrollo para el paso a producción de sus productos, charlas con desarrolladores sobre los procedimientos que se siguen para documentar las diferentes actualizaciones, documentos de versionamiento de base de datos, indagación de investigaciones que se han realizado sobre el tema como también la entrevista a docentes y/o conocedores del tema de investigación, notas de clase y por ultimo utilizamos bibliografía como [Stevens and Pooley, 2007], [Schmuller, 2000], [Jacobson, 2000] [Elmasri and Navathe, 2007], [Bruegge and Dutoit, 2002] [Sommerville, 2005] [Castro,], entro otros.

- Entrevistas: Se realizaron entrevistas con el objetivo de medir tiempos, demoras, gasto de horas en soporte, problemas más frecuentes, entre otros datos. El personal a entrevistado corresponde a desarrolladores, soporte técnico, implementación, infraestructura, etc.
- La información obtenida a partir de las entrevistas se fue diligenciando en el Software libre REM. Dicha información fue la base para el análisis posterior que nos permitió generar u obtener los requisitos funcionales y no funcionales de la solución planteada.

1.7. Estudio de sistemas previos

Sistemas de control de versiones locales

Un método de control de versiones usado por mucha gente es copiar los archivos a otro directorio (quizás indicando la fecha y hora en que lo hicieron, si son avispados). Este enfoque es muy común porque es muy simple, pero también tremendamente propenso a errores. Es fácil olvidar en qué directorio te encuentras, y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.

Para hacer frente a este problema, los programadores desarrollaron hace tiempo VCSs locales que contenían una simple base de datos en la que se llevaba registro de todos los cambios realizados sobre los archivos.

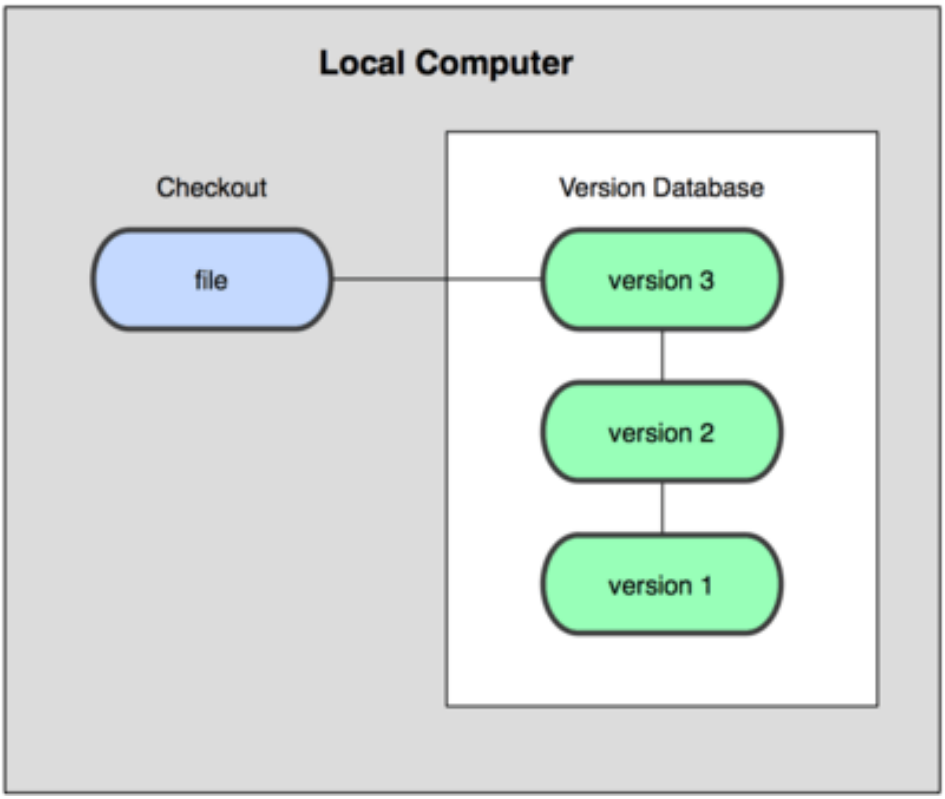


Figura 1.4: Diagrama de control de versiones local.

Una de las herramientas de control de versiones más popular fue un sistema llamado rcs, que todavía podemos encontrar en muchos de los ordenadores actuales. Hasta el famoso sistema operativo Mac OS X incluye el comando rcs cuando instalas las herramientas de desarrollo. Esta herramienta funciona básicamente guardando conjuntos de parches (es decir, las diferencias entre archivos) de una versión a otra en un formato especial en disco; puede entonces recrear cómo era un archivo en cualquier momento sumando los distintos parches.

Sistemas de control de versiones centralizados

El siguiente gran problema que se encuentra la gente es que necesitan colaborar con desarrolladores

en otros sistemas. Para solventar este problema, se desarrollaron los sistemas de control de versiones centralizados (Centralized Version Control Systems o CVCSs en inglés). Estos sistemas, como CVS, Subversion, y Perforce, tienen un único servidor que contiene todos los archivos versionados, y varios clientes que descargan los archivos desde ese lugar central. Durante muchos años éste ha sido el estándar para el control de versiones.

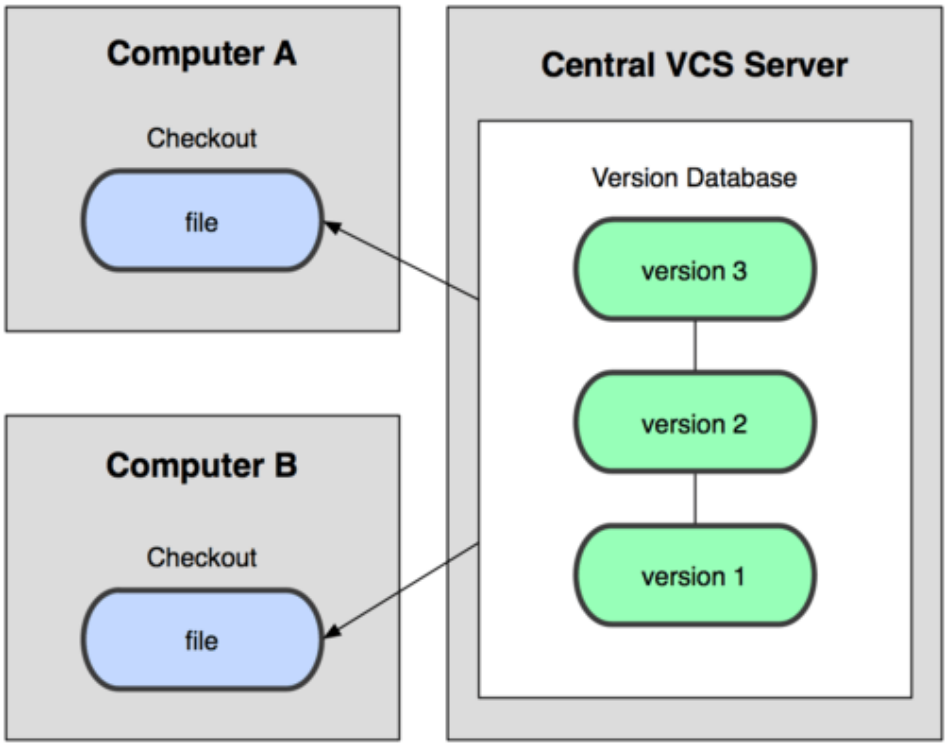


Figura 1.5: Diagrama de control de versiones centralizado.

Esta configuración ofrece muchas ventajas, especialmente frente a VCSs locales. Por ejemplo, todo el mundo puede saber (hasta cierto punto) en qué están trabajando los otros colaboradores del proyecto. Los administradores tienen control detallado de qué puede hacer cada uno; y es mucho más fácil administrar un CVCS que tener que lidiar con bases de datos locales en cada cliente.

Sin embargo, esta configuración también tiene serias desventajas. La más obvia es el punto único de fallo que representa el servidor centralizado. Si ese servidor se cae durante una hora, entonces durante esa hora nadie puede colaborar o guardar cambios versionados de aquello en que están trabajando. Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han llevado copias de seguridad adecuadamente, pierdes absolutamente todo —toda la historia del proyecto salvo aquellas instantáneas que la gente pueda tener en sus máquinas locales. Los VCSs locales sufren de este mismo problema— cuando tienes toda la historia del proyecto en un único lugar, te arriesgas a perderlo todo.

Sistemas de control de versiones distribuidos

Es aquí donde entran los sistemas de control de versiones distribuidos (Distributed Version Control Systems o DVCSs en inglés). En un DVCS (como Git, Mercurial, Bazaar o Darcs), los clientes no sólo descargan la última instantánea de los archivos: replican completamente el repositorio. Así, si un servidor muere, y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios de los clientes puede copiarse en el servidor para restaurarlo. Cada vez que se descarga una instantánea,

en realidad se hace una copia de seguridad completa de todos los datos.

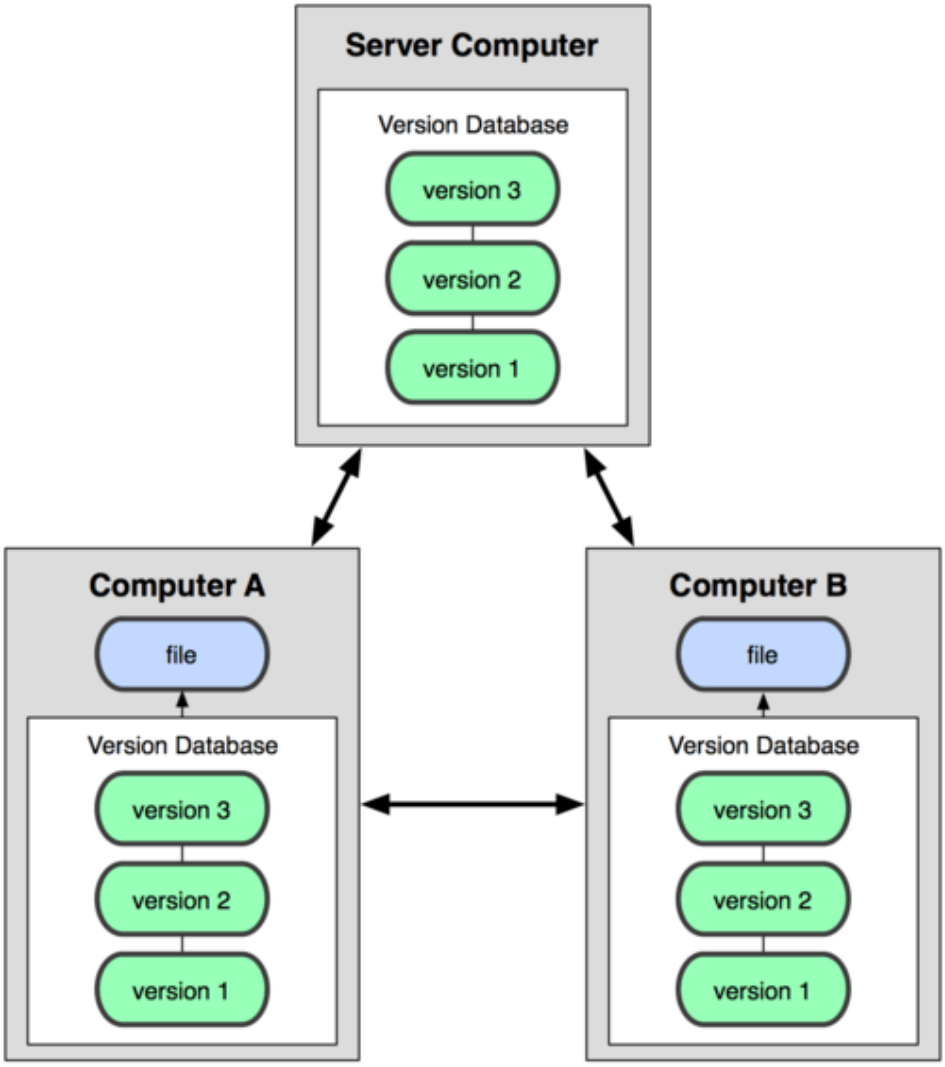


Figura 1.6: Diagrama de control de versiones distribuido.

Es más, muchos de estos sistemas se las arreglan bastante bien teniendo varios repositorios con los que trabajar, por lo que puedes colaborar con distintos grupos de gente simultáneamente dentro del mismo proyecto. Esto te permite establecer varios flujos de trabajo que no son posibles en sistemas centralizados, como pueden ser los modelos jerárquicos.

[GIT, 2016a]

Una breve historia de Git

Como muchas de las grandes cosas en esta vida, Git comenzó con un poco de destrucción creativa y encendida polémica. El núcleo de Linux es un proyecto de software de código abierto con un alcance bastante grande. Durante la mayor parte del mantenimiento del núcleo de Linux (1991-2002), los cambios en el software se pasaron en forma de parches y archivos. En 2002, el proyecto del núcleo de Linux empezó a usar un DVCS propietario llamado BitKeeper.

En 2005, la relación entre la comunidad que desarrollaba el núcleo de Linux y la compañía que desarrollaba BitKeeper se vino abajo, y la herramienta dejó de ser ofrecida gratuitamente. Esto impulsó a la comunidad de desarrollo de Linux (y en particular a Linus Torvalds, el creador de Linux) a desarrollar su propia herramienta basada en algunas de las lecciones que aprendieron durante el uso de BitKeeper. Algunos de los objetivos del nuevo sistema fueron los siguientes:

- Velocidad
- Diseño sencillo
- Fuerte apoyo al desarrollo no lineal (miles de ramas paralelas)
- Completamente distribuido
- Capaz de manejar grandes proyectos (como el núcleo de Linux) de manera eficiente (velocidad y tamaño de los datos)

Desde su nacimiento en 2005, Git ha evolucionado y madurado para ser fácil de usar y aún conservar estas cualidades iniciales. Es tremendamente rápido, muy eficiente con grandes proyectos, y tiene un increíble sistema de ramificación (branching) para desarrollo no lineal.
[GIT, 2016b]

Algunos de los sistemas de control de versiones más famosos son Subversion (también conocido como Svn), Git y Mercurial.

Subversion
Fue lanzado en el año 2000 bajo una licencia Apache/BSD y su última versión estable fue liberada en febrero de este mismo año. Subversion es uno de los sistemas más legendarios, sin embargo su uso ha ido decreciendo con el paso de los años. Hay quienes afirman que está cerca del final de su ciclo de vida pero todavía miles de empresas lo usan en el día a día.

Git
Fue desarrollado nada más y nada menos que por Linus Torvals, el mismo padre del kernel Linux, en el año 2005. Git surge como alternativa a BitKeeper, un control de versiones privativo que usaba en ese entonces para el kernel. Es liberado bajo una licencia GNU GPLv2 y su última versión estable fue publicada a inicios de Abril de este año. Se ha convertido en uno de los más usados alrededor del mundo.

Mercurial
Este proyecto nació en 2005 y su desarrollo comenzó a pocos días del de Git, por esto y por algunas similitudes en sus características es que muchos los consideran sistemas hermanos. Mercurial fue desarrollado por Matt Mackall, y al igual que Linus, Matt buscaba una alternativa a BitKeeper para el control del versiones del kernel Linux. También es liberado bajo una licencia GNU GPL v2 y su última versión estable fue publicada en Abril de este mismo año.

Subversion es considerado el abuelo de los sistemas de control de versiones; es centralizado, lento y más pesado que sus sucesores. Git y Mercurial por el contrario son los jóvenes de la generación de relevo; distribuidos, rápidos y ligeros, acordes a las exigencias de hoy en día.

Al final de la historia el proyecto Linux se decidió por Git, sin embargo no podemos dudar que los tres sistemas son de gran calidad y de muchísima utilidad en el desarrollo de software, sin olvidar que también son open source y gratuitos.
[Hipertextual, 2014]

Parte II

DESARROLLO DE LA INVESTIGACIÓN

Basado en la IEEE, un punto de vista es “una especificación de las convenciones para la construcción y el uso de una vista. Un patrón o plantilla a partir de la cual desarrollar vistas individuales estableciendo los propósitos y la audiencia para una vista y las técnicas para su creación y análisis”.

2.1. Arquitectura del Negocio

2.1.1. Punto de Vista de Organización

La organización se encuentra en la Ciudad de Bogotá y está liderada o representada por un gerente General. Al interior de la organización están visibles las área importantes y que hacen parte de su función de negocio. Aqui podemos encontrar la Gerencia comercial, Gerencia de proyectos, Gerencia de tecnología, Gestión administrativa y financiera y la gerencia de Gestión Humana.

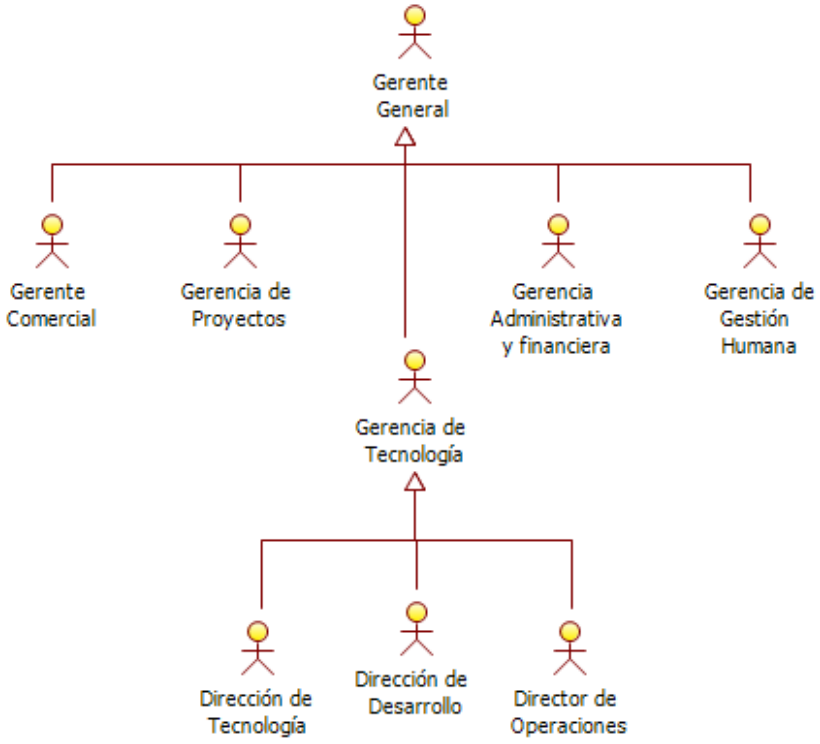


Figura 2.1: Punto de vista de Organización

- Gerencia General: Establece directrices y lineamientos necesarios para dar cumplimiento a la misión y visión de la empresa bajo la premisa de la mejora continua.
- Gerencia Comercial: Convierte oportunidades de negocio en clientes permanentes para la compañía.
- Gerencia de Proyectos: Establece y ejecuta las mejores prácticas para la gestión de los proyectos en los diferentes clientes y áreas de la compañía.
- Gerencia de Tecnología: Desarrolla y construye productos de software para satisfacer los requerimientos del cliente.
- Gestión Administrativa y financiera: Gestiona los recursos administrativos y financieros para satisfacer las necesidades internas de la compañía.
- Gerencia de Gestión Humana: Suministra trabajadores idóneos, con las competencias y habilidades requeridas por la Organización para un correcto desempeño en los diferentes cargos, alcance de los objetivos estratégicos y ajuste a la cultura organizacional de la Compañía, generando estrategias que promuevan el desarrollo y bienestar del personal y que faciliten el cumplimiento de los requisitos legales y de seguridad y salud en el trabajo. De igual forma con esta área se busca el bienestar de los trabajadores basado en incentivos, carrera dentro de la organización, programa de capacitaciones, entre otros items importantes con los cuales cuenta dicha área.

2.1.2. Punto de Vista de Cooperación de actor

El negocio y los actores asociados a este trabajan colaborativamente para ciertos procesos dentro de la organización. Las áreas que tendrán colaboración entre si son las áreas del sector comercial, el área de soporte, el área de Calidad e implementación y el área de Desarrollo de Software. La interacción de éstas áreas entre sí dará un objetivo útil a la evolución de las aplicaciones.

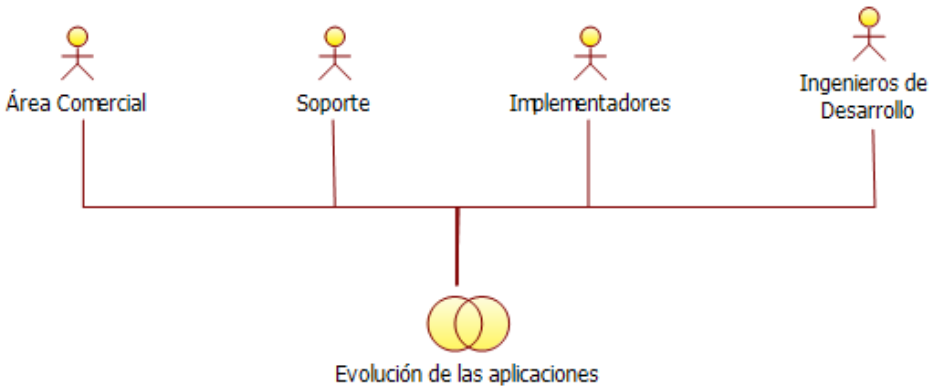


Figura 2.2: Punto de vista de Cooperación de Actor

2.1.3. Punto de Vista de Función de Negocio

La función de negocio analizada se basada en los siguiente tres ítems principales que se ven afectados por el proyecto. Cabe anotar que las áreas implicadas tendrán directa relación con el presente proyecto dado que el uso de dicha aplicación les facilitará el cumplimiento de sus labores diarias sobre todo al

área de desarrollo y de calidad y soporte técnico como tambiéon el área comercial para las licitaciones que estos realizan constantemente para la búsqueda de futuros potenciales clientes:

- Desarrollo de Software: área encarga de desarrollar nuevos productos y servicios que estén a la vanguardia de la tecnología. Con esto se busca mantener los productos actualizados, estables y con alta fiabilidad y confiabilidad.
- Implementación: área encargada: área encargada del soporte directo con el cliente y se basa en atender las peticiones de estos, redirigirlas al área de encargada o simplemente dar respuesta inmediata al cliente. De igual forma, se encarga de las actualizaciones y/o instalaciones de los productos en el cliente.
- Gestión comercial: Encargada de ofrecer el producto a nuevos clientes o las nuevas funcionalidades a los clientes existentes. Para esto se basa en documentos de evolución de las aplicaciones que contengan las diferentes funcionalidades de las aplicaciones.

Cada uno de las funciones anteriormente mencionadas tienen afectación directa por el proyecto ya que facilitará su labor diario y de cierta forma, la generación de documentos sobre las aplicaciones se ven reflejada en la facilidad para la realización de sus procesos y del cumplimiento de sus labores. Es de aclarar que el punto de vista de función de negocio propuesto implica la utilización del producto generado de este proyecto que es appEvolution y que por ende las áreas implicadas deben incluirlo en sus procesos para que puedan y tengan la obligación de generación y utilizar la implementación y evolución de las aplicaciones.

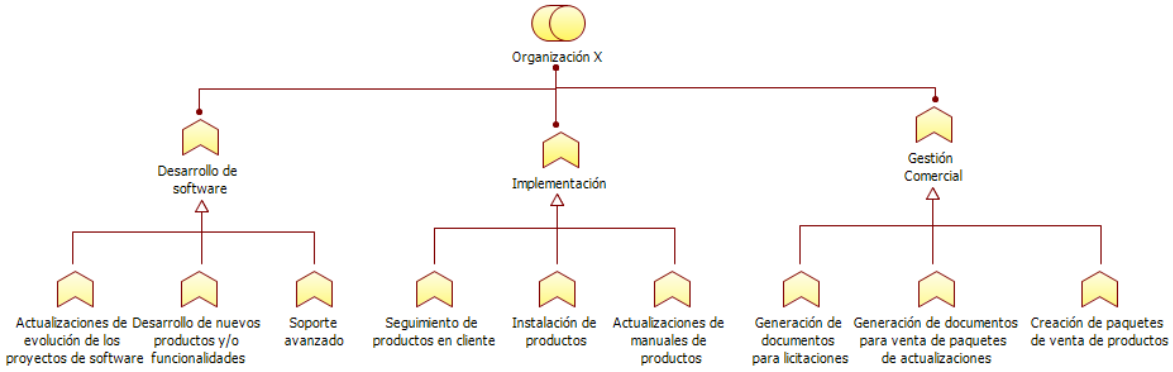


Figura 2.3: Punto de vista de Función de Negocio

2.1.4. Punto de Vista de Proceso de Negocio

La siguiente vista nos muestra los procesos y subprocesos que se ven vinculados con el objetivo final de este proyecto. El proceso de negocio es el siguiente:

a. Control y evolución de las aplicaciones: este proceso de negocio incluye:

- Desarrollo de requerimientos
  1) Revisión y análisis: incluye la revisión de los requerimientos solicitados por el cliente.

2) Asignación de equipo de trabajo: incluye la evaluación de los requerimientos y deterinar el personal que se encargará de cada ítem de desarrollo.
3) Desarrollo de requerimiento: Incluye las tareas necesarias para el desarrollo de los requerimientos.
4) Documentación de solución en el documento de implementación: Incluye documentar en el formato xml los cambios que se realizaron sobre la aplicación ya sea funcionalidad, mejora, error, configuración o scripts de base de datos.

- Evolución de las aplicaciones

1) Registro de aplicaciones y/o proyectos: Incluye el registro de las aplicaciones sobre las cuales se desea administrar su evolución.
2) Registro de clientes: Registro de los clientes de las aplicaciones de software que desarrolle la compañia.
3) Actualización de aplicaciones en cliente: Incluye el procedimiento de las actualizaciones. Desde la app se generará el documento de implementación que tendrá las caracteristicas del software o de la nueva versión a instalar.
4) Generación de documento de evolución: Esta generación hace reference al documento que indica como es la evolución de la aplicación entre dos versiones, es decir, se podrá determinar que tiene la aplicación en una versión específica en comparación con versiones anteriores.

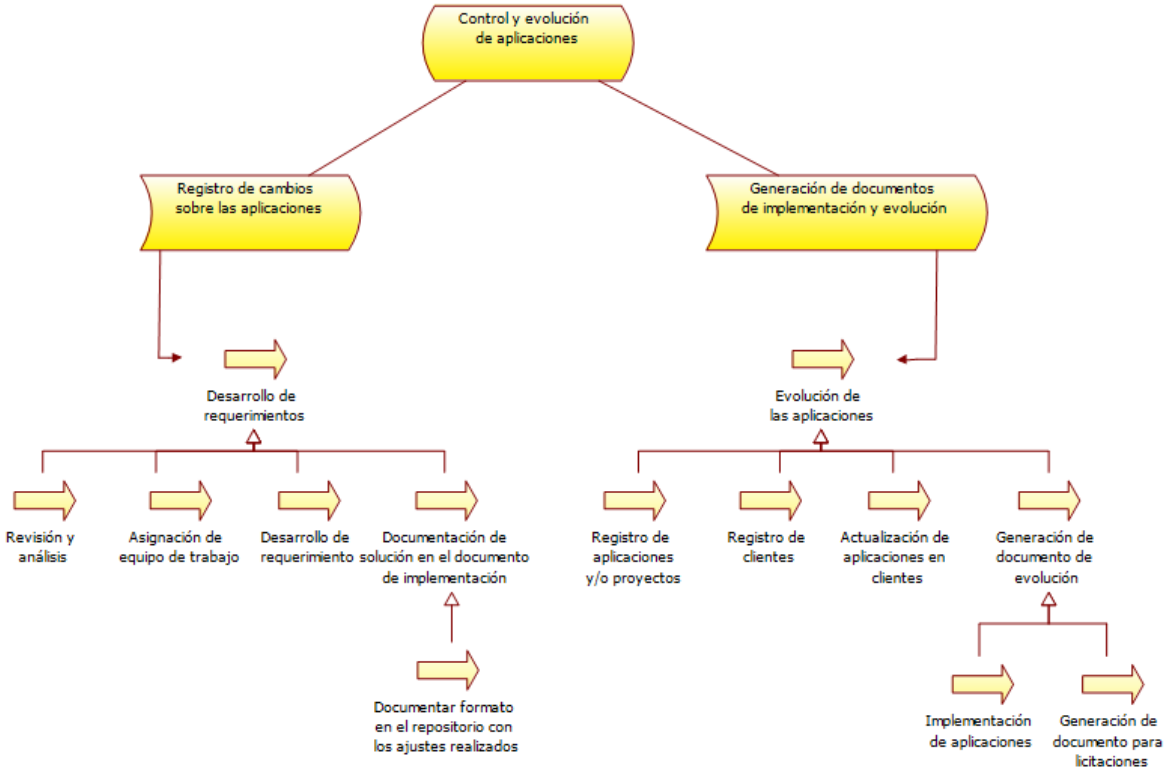


Figura 2.4: Punto de vista de Proceso de Negocio

2.1.5. Punto de Vista de Cooperación de Proceso de Negocio

Los diferentes procesos de negocio cooperan entre si para buscar un objetivo en común que es entrega de productos y/o servicios al cliente. Los procesos que cooperan entre si son Calidad y soporte técnico y Desarrollo de software.

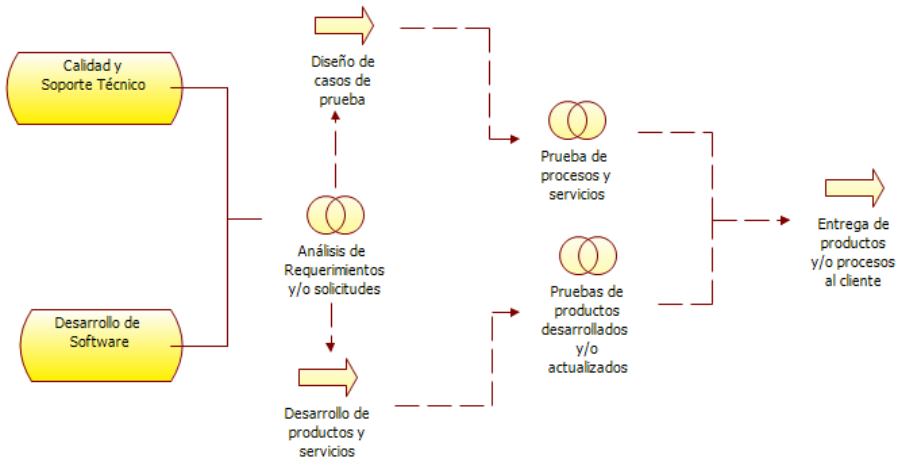


Figura 2.5: Punto de vista de Cooperación de Proceso de Negocio

2.1.6. Punto de Vista de Producto

El producto de software está compuesto por varias opciones y/o funcionalidades que permiten el control de la evolución de las aplicaciones. Esta solución incluye las siguientes características:

- Administración de Clientes: permite administrar los clientes de las diferentes aplicaciones incluyendo sus ambientes, entornos, versión de la aplicación instalada, etc. Con dicha administración se busca identificar los clientes de las aplicaciones, sus diferentes ambientes ya sea pruebas, desarrollo, producción o cualquier otro donde se instalen las aplicaciones y permitir identificar que características tiene cada ambiente, como base de datos, entre otras caracteristicas.

- Administración de aplicaciones y versiones: permite administrar las aplicaciones sobre las cuales se administra la evolución e implementación. Es decir, con dicha administración buscamos que se registren las aplicaciones, sus repositorios, sus versiones de tal forma que en cualquier momento se pueda identificar claramente como ha evolucionado la aplicación, es decir, analizar que funcionalidades, mejoras, correción de errores, Scripts de base de datos, configuraciones son requeridas para una versión en específica. Esto nos permitirá identificar, entre un rango de versiones, como ha evolucionado la aplicación en cuanto a los ítems anteriormente mencionados.

- Administración de usuarios: permite administrar los usuarios que tendrán acceso a la aplicación y sus diferentes permisos. Esto facilitará la labor para ciertos perfiles y que solo tengan acceso a lo permitido. Por ejemplo, un implementador tendrá la necesidad de identificar que cambios hay entre una versión y otra para facilitar dicha instalación al cliente. Por el contrario, un desarrollador tendrá acceso de proyectos para registrarlos y controlar las versiones que se vayan creando en el repositorio.

- Administración de la implementación: Permite a los implementadores definir y obtener el documento de implementación base para la instalación en los clientes. Es decir, se podrán identificar que aspectos se deben tener en cuenta para la instalación de la aplicación en una versión específica. Esto permitirá identificar los ítems en específico que determinan las nuevas características con que cuenta la aplicación. De dicha administración se busca generar documentos que sean útiles para el cliente en la instalación de las actualizaciones de las aplicaciones como tambien para los implementadores cuando son ellos quienes realizan directamente la actualización.

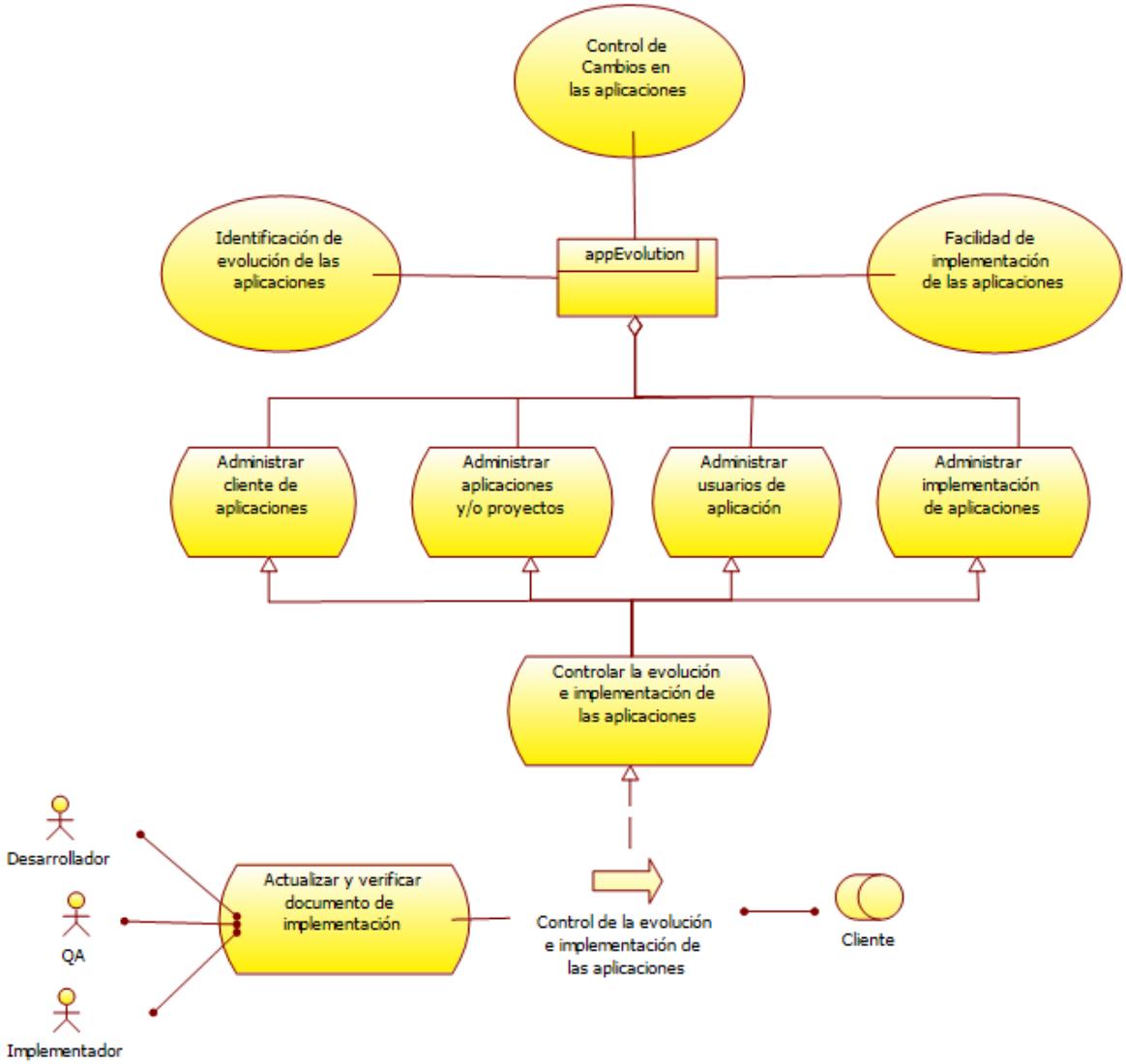


Figura 2.6: Punto de vista de Producto

2.2. Arquitectura del Sistema de Información

2.2.1. Punto de Vista de Comportamiento de Aplicación

A través de la siguiente vista buscamos hacer frente al comportamiento de aplicación basado en los requerimientos previamente definidos y que sirven como base para el desarrollo de la aplicación. Con el comportamiento esperado de aplicación se da a conocer las funciones que agregarían valor al negocio para cada una de las grandes características de esta.

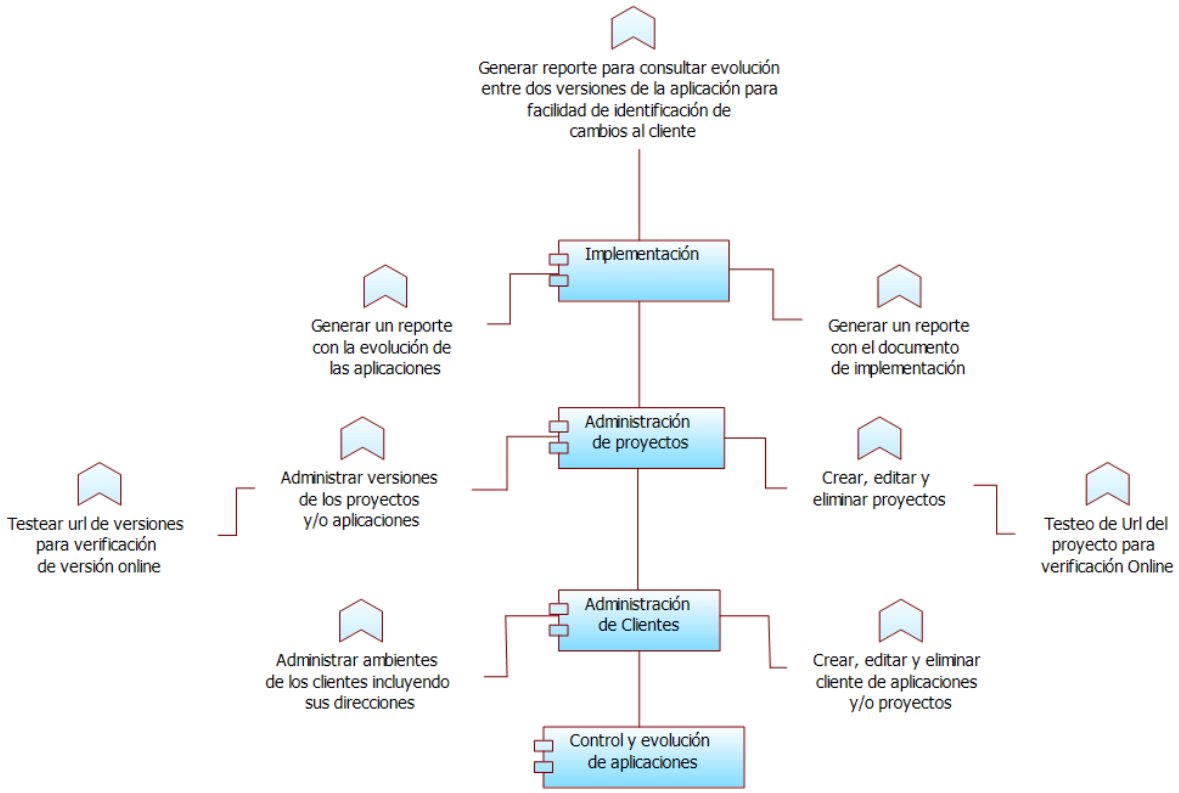


Figura 2.7: Punto de Vista de Comportamiento de Aplicación

2.2.2. Punto de Vista de estructura de Aplicación

Con el siguiente punto de vista se conocer el uso de la infraestructura por parte del presente proyecto y por ende de la aplicación generada. AppEvolution hace uso de un generador de PDF para generar los diferentes documentos de implementación o evolución. De igual forma hace uso de un generador de Scripts de base de datos que nos permitirá tener los scripts necesarios para actualizar una aplicación de una versión a otra en cuanto a la base de datos. Adicional a esto se conecta con un repositorio de código fuente para extraer la información propia de las aplicaciones y su evolución basado en funcionalidades, mejoras, corrección de errores, configuraciones, entre otros ítems. Además tendrá conexión con un servidor de Base de datos donde alojará la información referente a los clientes, proyectos, actualizaciones e implementación de las aplicaciones de software sobre las cuales se está llevando control.

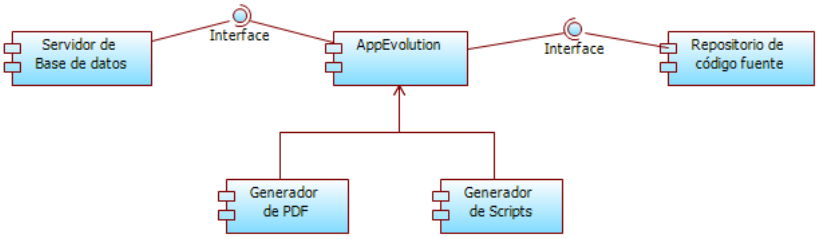


Figura 2.8: Punto de Vista de estructura de aplicación

2.2.3. Punto de Vista de uso de aplicación

Con el siguiente punto de vista se da a conocer al uso de la aplicación generada y sus principales características que ayudaran de alguna u otra forma en los procesos internos de las empresas desarrolladoras de software. Los grandes paquetes que hacen parte de estos son:

a. Administrar proyectos: permitirá realizar las operaciones sobre los proyectos sobre los cuales se podrá llevar control y evolución. Esto incluye repositorio, versiones, y caracteristicas propias de cada proyecto.

b. Administración de clientes: permitirá realizar operaciones sobre los clientes de las aplicaciones de sofware. Esto incluye ambientes, url, y características propias del cliente que nos permitirán llevar seguimiento sobre las aplicaciones instaladas y su constante evolución en la implementación de estas.

c. Administración de la evolución: permitirá generar documentos, reportes de las diferentes versiones de las aplicaciones. Esto incluye diferencia entre versiones seleccionadas, diferencia entre build de la misma versión, configuraciones requeridas para cada actualización, scripts de base de datos requeridos para dicha versión.

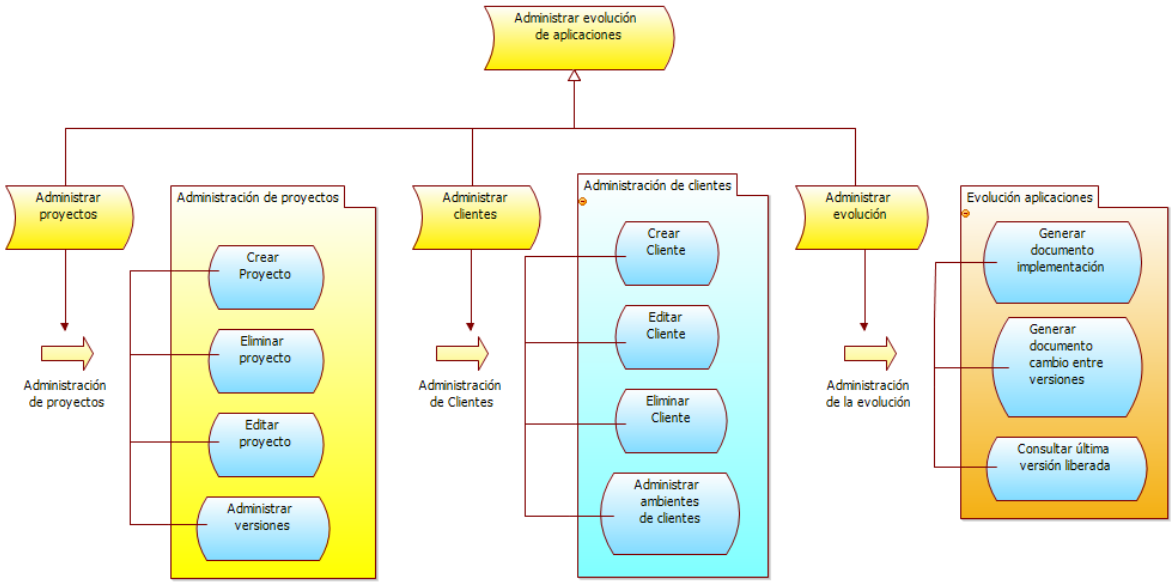


Figura 2.9: Punto de Vista de uso de aplicación

2.3. Arquitectura de tecnología

2.3.1. Punto de Vista de Infraestructura

La siguiente vista nos muestra la infraestructura necesaria para cumplir el objeto de este proyecto y por ende permitir el cumplimiento de los requerimientos definidos en la fase de ingeniería de requerimientos. Para la implementación se requiere:

a Una máquina cliente donde se instalará la aplicación appEvolution.

b Una base de datos sobre la cual se correrán los scripts iniciales y donde se conectará la aplicación.

c repositorio de código fuente donde se encuentren los proyectos. Esto es requerido y prioritario puesto que la aplicación obtendrá sus insumos a partir de conexión a un repositorio donde se encuentran los proyectos que se desean versionar. Es requerido dicha repositorio porque el presente proyecto se basa en dicha información para funcionar correctamente.

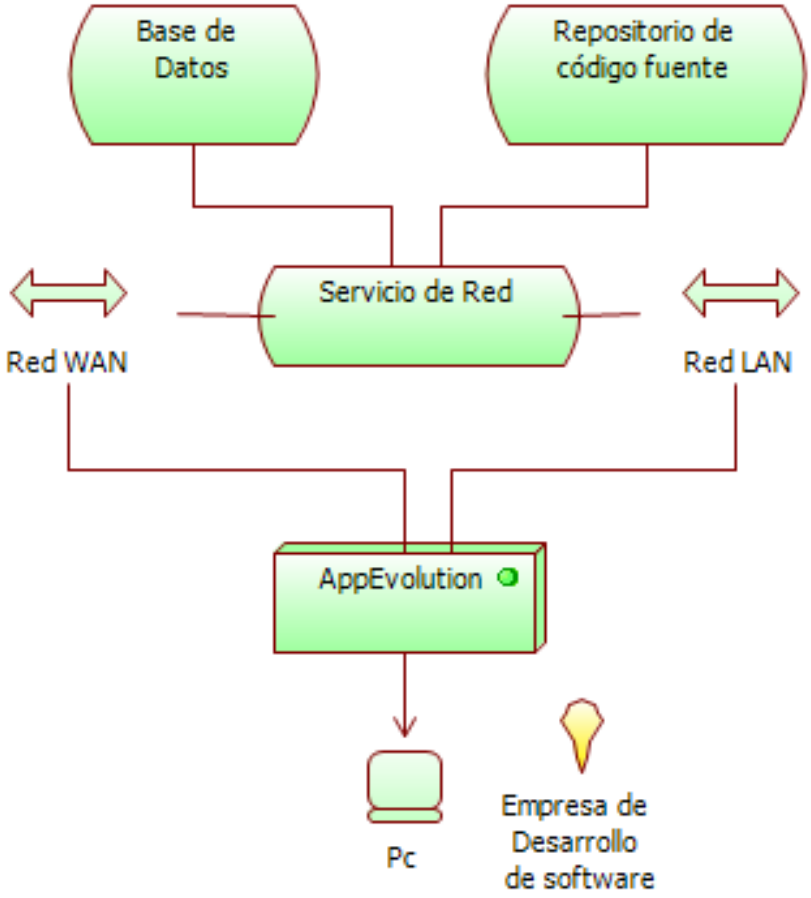


Figura 2.10: Punto de vista de Infraestructura

2.3.2. Punto de Vista de uso de infraestructura

Con el siguiente punto de vista se conocer el uso de la infraestructura por parte del presente proyecto y por ende de la aplicación generada. AppEvolution hace uso de un generador de PDF para generar los diferentes documentos de implementación o evolución. De igual forma hace uso de un generador de Scripts de base de datos que nos permitirá tener los scripts necesarios para actualizar una aplicación de una versión a otra en cuanto a la base de datos. Adicional a esto se conecta con un repositorio de código fuente para extraer la información propia de las aplicaciones y su evolución basado en funcionalidades, mejoras, corrección de errores, configuraciones, entre otros ítems.

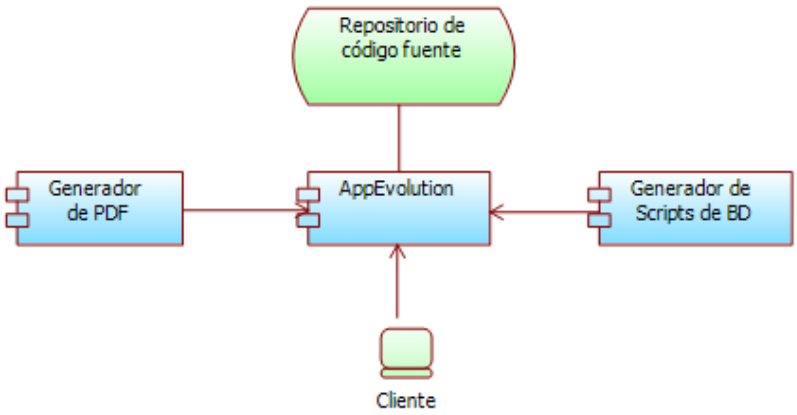


Figura 2.11: Punto de Vista de uso de infraestructura

2.3.3. Punto de Vista de organización e implementación

Para la implementación del proyecto fue necesario tener instalado la versión de java (mínimo versión 7). Se requiere de mínimo un equipo para la persona que lo desea usar donde instalará la aplicación appEvolution. Adicional a esto es necesario tener acceso a internet para tener conexión con el servidor de base de datos y con el repositorio de código fuente de los proyectos a controlar su evolución.

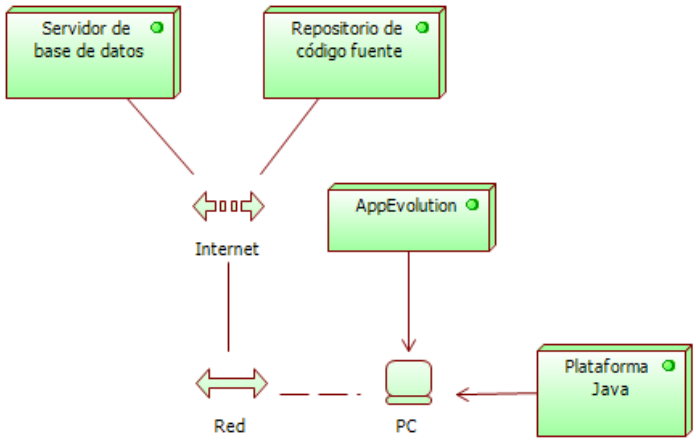


Figura 2.12: Punto de Vista de organización e implementación

2.3.4. Punto de Vista de estructura de Información

Con el siguiente punto de vista se busca estructurar el proyecto y como este incluye ciertos ítems que son importantes para la organización. Partimos desde documentos, capacitaciones, hasta detalles del software que nos muestran un arquitectura previa de como está este diseñado. De igual forma, se debe tener en cuenta que el punto de vista a mostrar muestra los rasgos mas globales de la aplicación para el manejo de la estructura de información manejada.

a. Documentos: Los documentos generados con la entrega de este proyecto son el manual de usuario, el manual de instalación y el manual de implementación. Estos documentos son importantes ya que pemitirán tener un encuentro cercano con la aplicación para facilitar su uso a través de dichos documentos.

b. Capacitaciones: Se dictarán capacitaciones sobre el uso de la aplicación a las diferentes áreas y como podrán usarla para las labores que desempeñan diariamente. Esto es importante para aprender a conocer el funcionamiento de la aplicación objetivo de este proyecto.

c. Software: El software está compuesto por tres capas que son la capa de presentación, la capa de negocio y la capa de persistencia. Adicional a esto hace uso de dos apis que son clientes para conectarse al repositorio de código fuente y api para generación de documentos en PDF. Esta conexión externa se realiza en la capa de negocio y permitirá extraer la información de cada una de las aplicaciones que son previamente registradas.

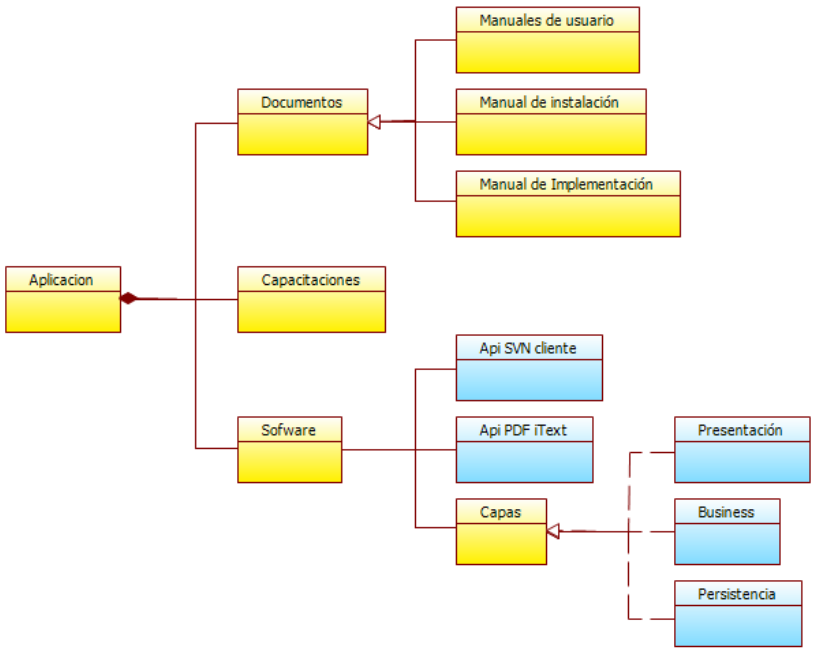


Figura 2.13: Punto de Vista de estructura de Información

FASES DEL PROCESO UNIFICADO

3.1. Fase de concepción

3.1.1. Ingeniería de Requerimientos

"La ingeniería de requerimientos proporciona el mecanismo apropiado para entender lo que desea el cliente, analizar las necesidades, evaluar la factibilidad, negociar una solución razonable, especificar la solución sin ambigüedades, validar la especificación y administrar los requerimientos a medida de que se transforman en un sistema funcional". [Pressman and Troya, 1988]

3.1.2. Recolección de Requerimientos

Para la recolección de información basado en el problema existente y lo que se requiere para solucionarlo se realizaron varias reuniones que nos aclaran mas la temática y por ende nos dan una idea inicial sobre la identificación de los objetivos del Sistema. De igual forma, la importancia de esta radica en que se identifican las causas reales de los problemas encontrados.

3.1.3. Reuniones

"Los Requerimientos de Software son las necesidades de los Stakeholders que requiere que el Sistema deba de cumplir de manera Satisfactoria. Son los que definen las funciones que el sistema será capaz de realizar, describen las transformaciones que el sistema realiza sobre las entradas para producir salidas. Es importante que se describa el ¿Qué? y no el ¿Cómo? se deben hacer esas transformaciones. Estos requerimientos al tiempo que avanza el proyecto de software se convierten en los algoritmos, la lógica y gran parte del código del sistema". Tomado de http://www.northware.mx/wp-content/uploads/2013/04/Tecnicas-efectivas-para-la-toma-de-requerimientos.pdf

Üna vez que tenemos identificados los tipos de requerimientos que existen, y las características que deben cumplir, podemos comenzar con la descripción de las actividades que nos ayudarán a realizar una buena obtención de requerimientos. No vamos a descubrir el hilo negro, ya está más que definido el proceso, solo hay que aprender a llevarlo adecuadamente". Tomado de http://www.northware.mx/wp-content/uploads/2013/04/Tecnicas-efectivas-para-la-toma-de-requerimientos.pdf

1. Reunión I: situación actual y problemas

Fecha	07/08/2016
Hora	09:04 a.m
Lugar	Telefónicamente
Asistentes	Ing. Desarrollo - Ing. Franklin Candanoza
Resultados	Inicialmente se tocaron los temas de problemas actuales que se presentaban por la falta de control de un versionamiento de las aplicaciones, es decir, cuando otra área realizaba preguntas o quería conocer en profundidad que cambios tenía la aplicación o aplicaciones en una versión específica, el área de Desarrollo no tenía claro dichas funcionalidades y la identificación de estos en cada versión era bastante complicado. - No tienen claro que funcionalidades nuevas hay en cada entrega. - No tienen claro que errores se corrigen para cada versión entregada. - No tienen control claro sobre los cambios en base de datos que requiere una versión específica. - No se puede generar documento de implementación que incluya todos estos cambios.

Cuadro 3.1: Reunión I: Problema y situación actual

2. Reunión II: Problemas en las actualizaciones de aplicaciones

Fecha	14/08/2016
Hora	09:13 a.m
Lugar	Telefónicamente
Asistentes	Ing. Desarrollo - Ing. Franklin Candanoza
Resultados	Las actualizaciones de las aplicaciones generan muchos ruido en el cliente porque no está plenamente identificados los cambios a tener en cuenta para dicha actualización. - En muchas ocasiones, por ejemplo, al no tener manejo de los scripts de base de datos debidamente gestionados, se generan errores no controlados que en última instancia bajan la credibilidad y la confianza en la aplicación. De igual forma, en ambientes de producción los errores se presentan frecuentamente y mientras no se controlen y gestionen correctamente estos cambios los errores van a seguir apareciendo. - La gerencia comercial muchas veces solicita un paquete de funcionalidades, mejoras, errores corregidos que tiene una aplicación que les permita poder vender el producto y ofrecer lo que realmente tiene con todas sus características. Pero como este control no se tiene y no se tienen plenamente identificados por cada versión mayor y menos entregada, al área de comercial le queda dificil ofrecer el producto con todas sus bondades y características. Es por esto, que en la mayoría de las veces no ofrecen caracteristicas porque no tienen información de que realmente existan.

Cuadro 3.2: Reunión II: Problemas en las actualizaciones de aplicaciones

3. Reunión III: Actualización de Componentes

Fecha	20/08/2016
Hora	10:22 a.m
Lugar	Telefónicamente
Asistentes	Ing. Desarrollo - Ing. Franklin Candanoza
Resultados	El producto principal está integrado con muchos componentes que trabajan independientemente. Sin embargo, hay ocasiones en que se requiere actualización de dichas aplicaciones o componentes de la mano de la aplicación, es decir, para una versión específica se requiere que el componente XY esté en dicha versión. El problema radica en que no se tiene gestión sobre estas actualizaciones y cuando se instalan productos con componentes desactualizados el sistema no realiza lo que realmente el usuario espera y es cuando aparecen las quejas de estos sobre el funcionamiento de la aplicación.

Cuadro 3.3: Reunión III: Actualización de Componentes

4. Reunión IV: Gestión y administración de cambios en los proyectos

Fecha	21/08/2016
Hora	10:29 a.m
Lugar	Telefónicamente
Asistentes	Ing. Desarrollo - Ing. Franklin Candanoza
Resultados	El equipo de Desarrollo y en general en beneficio de toda la empresa buscan que exista una aplicación o un Sistema que les permita administrar todos los cambios en las diferentes aplicaciones, incluyendo clientes, versiones, cambios, entre otros. Esto con el fin de permitir a diferentes usuarios poder obtener dicha información y de acuerdo a su perfil utilizar el reporte ya sea para comercial o para una nueva implementación.

Cuadro 3.4: Reunión IV: Gestión y administración de cambios en los proyectos

5. Reunión V: Documento de Diccionario de mensajes

Fecha	28/08/2016
Hora	10:29 a.m
Lugar	Telefónicamente
Asistentes	Ing. Desarrollo - Ing. Franklin Candanoza
Resultados	Actualmente manejan un concepto llamado Diccionario de mensajes y buscan que a través de la herramienta pueda generarse el diccionario de mensajes ideal para el cliente en la versión que se va a instalar. Esto quiere decir, que los mensajes codificados ya están, pero es necesario extraer dicha información y geneararla correctamente de acuerdo a la versión a instalar.

Cuadro 3.5: Reunión V: Documento de Diccionario de mensajes

6. Reunión VI: Documento de implementación para los clientes

Fecha	25/08/2016
Hora	10:29 a.m
Lugar	Sala de Juntas
Asistentes	Ing. Desarrollo - Ing. Franklin Candanoza
Resultados	El área de la empresa encargada de la implementación de nuevas instalación o nuevos clientes requieren que a través de una herramienta puedan generar información referente a las acciones que deben tener en cuenta los implementadores para instalar el software en una versión específica. Requieren saber con exactitud que configuración, cambios en base de datos requieren para actualizar una versión. Esta información es útil para ellos porque el proceso de implementación será mas sencillo de ejecutar

Cuadro 3.6: Reunión VI: Documento de implementación para los clientes

7. Reunión VII: Información de Gestión de cambios en los proyectos debe estar centrada

Fecha	03/09/2016
Hora	10:29 a.m
Lugar	Telefónicamente
Asistentes	Ing. Desarrollo - Ing. Franklin Candanoza
Resultados	La empresa hace tiempo presenta un problema sobre el control de cambios en los proyectos. Un problema grande es llevar un control y seguimiento sobre estos y que sea desde un punto central pero con dicha característica o solución no se cuenta. Se desea que exista una heramienta que los usuarios puedan ingresar y puedan ver el estado de todos los proyectos y realizar operaciones sobre la gestión de estos sin ningún problema, es decir, puedan darse build por cerrados, analizar los ajustes realizado para cada build, etc.

Cuadro 3.7: Reunión VII: Información de Gestión de cambios

3.1.4. Recolección de objetivos del Sistema

Durante esta actividad se busca identificar los objetivos del sistema, que corresponden a requerimientos propios del sistema pero se le da un enfoque de negocio.

- Objetivo I: Configurar conexiones y proyectos

OBJ-0001	Configurar conexiones y proyectos
Versión	1.0 (07/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá permitir configurar las conexiones a los diferentes proyectos o aplicaciones que se van administrar para poder obtener los cambios sobre estas.
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.8: Objetivo I: Configurar conexiones y proyectos

- Objetivo II: Administrar usuarios

OBJ-0002	Administrar usuarios
Versión	1.0 (14/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá permitir la administración de usuarios que se conectarán al sistema de administración , es decir, que se podrán crear, editar, eliminar y buscar usuarios en la aplicación.
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.9: Objetivo II: Administrar usuarios

- Objetivo III: Administrar perfiles de usuario

OBJ-0003	Administrar perfiles de usuario
Versión	1.0 (15/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá permitir administrar los perfiles de los usuarios de tal forma que se puedan asignar a cada uno dependiendo su función u objetivo buscado con la aplicación.
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.10: Objetivo III: Administrar perfiles de usuario

- Objetivo IV: Administrar permisos de usuario

OBJ-0004	Administrar permisos de usuario
Versión	1.0 (31/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá permitir administrar permisos de los usuarios para temas de seguridad y darles acceso sólo al módulo requerido de acuerdo a su perfil .
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.11: Objetivo IV: Administrar permisos de usuario

- Objetivo V: Administrar clientes

OBJ-0005	Administrar clientes
Versión	1.0 (31/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá permitir administrar clientes que usan la aplicación incluyendo datos como versión instalada, fecha de generación de la aplicación, ambientes que tiene configurado y las versiones en cada uno de estos ambientes.
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.12: Objetivo V: Administrar clientes

- Objetivo VI: Generar reporte de implementación

OBJ-0006	Generar reporte de implementación
Versión	1.0 (04/10/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá permitir administrar clientes que usan la aplicación incluyendo datos como versión instalada, fecha de generación de la aplicación, ambientes que tiene configurado y las versiones en cada uno de estos ambientes.
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.13: Objetivo VI: Generar reporte de implementación

3.2. Fase de elaboración

3.2.1. Modelos de casos de uso

Para los requerimientos de usuario se realizarán los casos de uso que describan la funcionalidad y las necesidades de este. Se hará uso de UML como lenguaje. A continuación se detallaran cada uno de los paquetes y sus casos de uso incluyendo información detallada de cada uno.

1. Configuración de Proyectos:

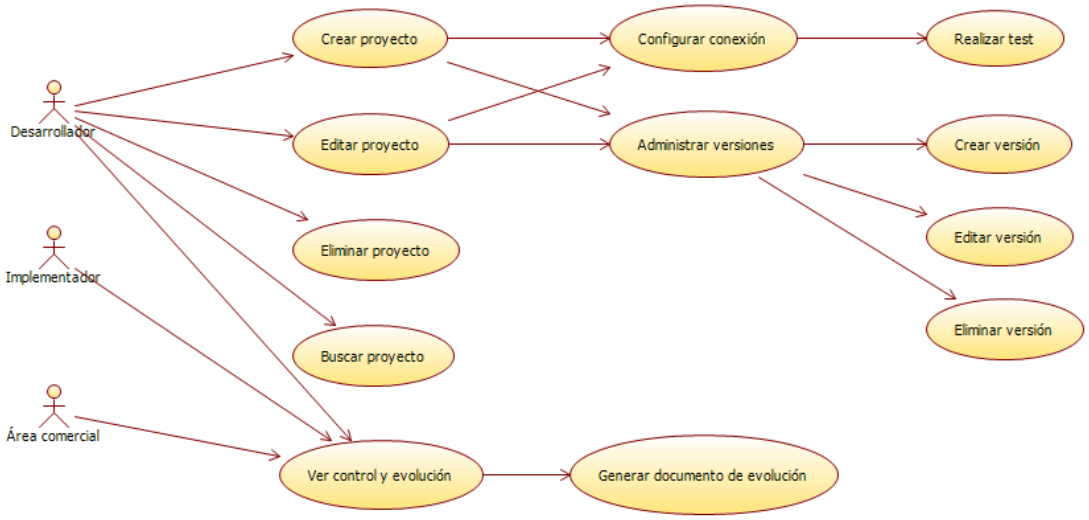


Figura 3.1: Diagrama de casos de uso: configurar proyectos

En relación con los requerimientos básicos, se definen tres actores principales:

1. Desarrollador: Es un empleado de la empresa, encargado de la codificación de software. Éste interactúa con las funciones elementales de seguimiento del producto base de desarrollo. Controla las operaciones CRUD de cada proyecto y realiza como tal la administración de versiones.
2. Implementador: Es un empleado de la empresa, encargado de realizar la implementación y puesta en marcha de los sistemas base de desarrollo. Para este módulo y según la necesidad puede ver la evolución de un sistema y generar un documento que ejemplifica el seguimiento del mismo.
3. Área Comercial: Es un departamento de la empresa, encargada de vender los productos base de desarrollo. Puede ver la información concerniente a la evolución de un sistema para fines de comunicación con el cliente y comercialización de un producto.

a) Crear Proyecto

UC-0001	**Crear proyecto**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se presiona sobre la opción de crear nuevo proyecto.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de crear nuevo proyecto 3. El sistema muestra un popup con los datos del nuevo proyecto a realizar 4. El actor Desarrollador (ACT-0001) diligencia los datos solicitados y presiona aceptar
Postcondiciones	1. Se crea el proyecto exitosamente
Excepciones	1. Si el nombre del nuevo proyecto ya existe se genera el mensaje: "Ya existe unproyecto con este nombre" 2. Si no hay conexión con el Repositorio se emite el mensaje: "No hay conexión con el proyecto"
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.14: Caso de uso: Crear proyecto

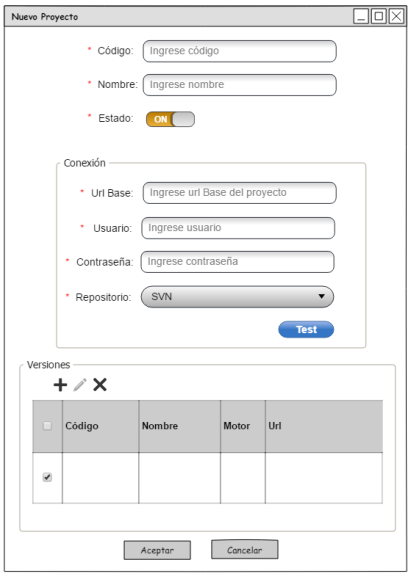


Figura 3.2: Prototipo: Crear nuevo proyecto

b) Configurar Conexión:

UC-0002	**Configurar Conexión**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se presiona sobre la conexión que corresponde al proyecto que se está creando, es decir, cuando se inicie la creación de una nuevo aplicación a realizar gestión de cambios.
Precondiciones	1. El usuario debe estar logeado en la aplicación. 2. Abrir ventana para crear un nuevo cliente
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de crear nuevo cliente 3. El sistema muestra un popup con los datos de la nueva conexión a realizar. 4. El actor Desarrollador (ACT-0001) diligencia los datos solicitados y presiona test para verificar conexión con el repositorio para dicho proyecto
Postcondiciones	1. Conexión al proyecto creada exitosamente
Excepciones	1. Si no existe conexión con el repositorio se genera el mensaje: "No hay conexión con el repositorio".
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.15: Caso de uso: Configurar Conexión

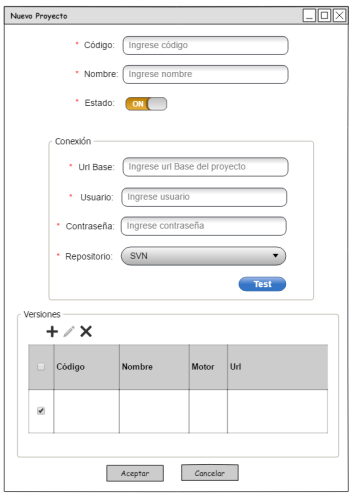


Figura 3.3: Prototipo: Crear conexión en nuevo proyecto

c) Administrar Versiones

UC-0003	**Administrar Versiones**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se presiona sobre la opción de crear proyecto y se visualiza el área de Versiones.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de crear nuevo proyecto 3. El sistema muestra un popup con los datos del nuevo proyecto a realizar 4. El actor Desarrollador (ACT-0001) puede crear, eliminar o editar versiones de los proyectos
Postcondiciones	1. Se visualizan en la grilla las versiones creadas
Excepciones	
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.16: Caso de uso: Administrar Versiones

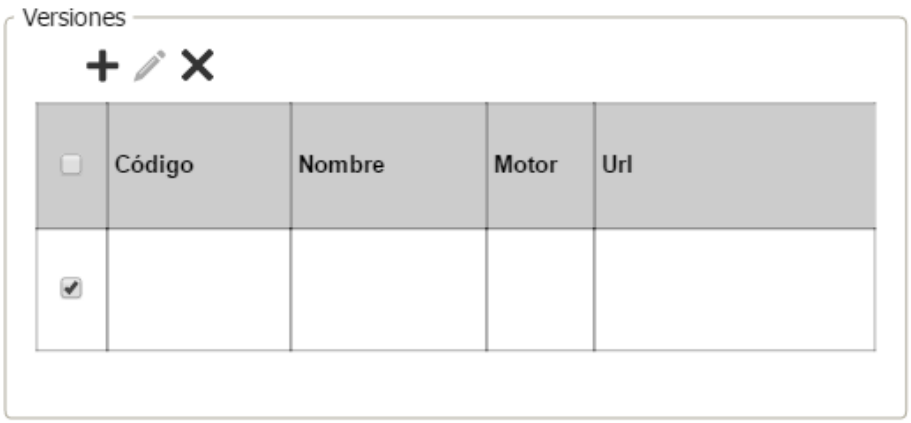


Figura 3.4: Prototipo: Administrar versiones

d) Crear Versión

UC-0004	**Crear Versión**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se presiona sobre la opción de crear versión.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de crear nuevo proyecto 3. El sistema muestra un popup con los datos del nuevo proyecto a realizar 4. El actor Desarrollador (ACT-0001) hace clic sobre la opción de crear versión. 5. El sistema muestra un formulario con los datos de la versión a registrar.
Postcondiciones	1. Se visualizan en la grilla de las versiones la nueva versión creada
Excepciones	1. Si no hay conexión con el proyecto al repositorio se emite el mensaje: No hubo conexión con la versión diligenciada
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.17: Caso de uso: Crear versión

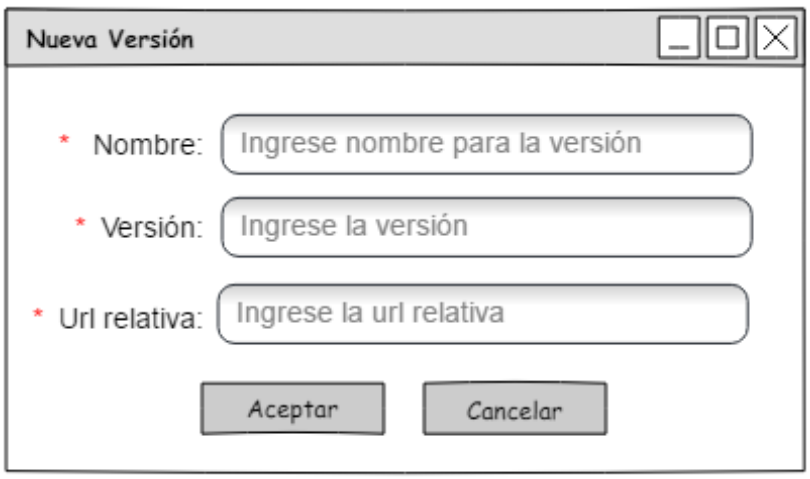


Figura 3.5: Prototipo: Crear versión

e) Editar Versión

UC-0005	**Editar Versión**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se presiona sobre la opción de editar versión.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de crear nuevo proyecto 3. El sistema muestra un popup con los datos del nuevo proyecto a realizar 4. El actor Desarrollador (ACT-0001) hace clic sobre la opción de editar versión. 5. El sistema muestra un formulario con los datos de la versión a editar.
Postcondiciones	1. Se visualizan en la grilla de las versiones la versión editada
Excepciones	1. Si no hay conexión con el proyecto al repositorio se emite el mensaje: No hubo conexión con la versión diligenciada
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.18: Caso de uso: Editar Versión

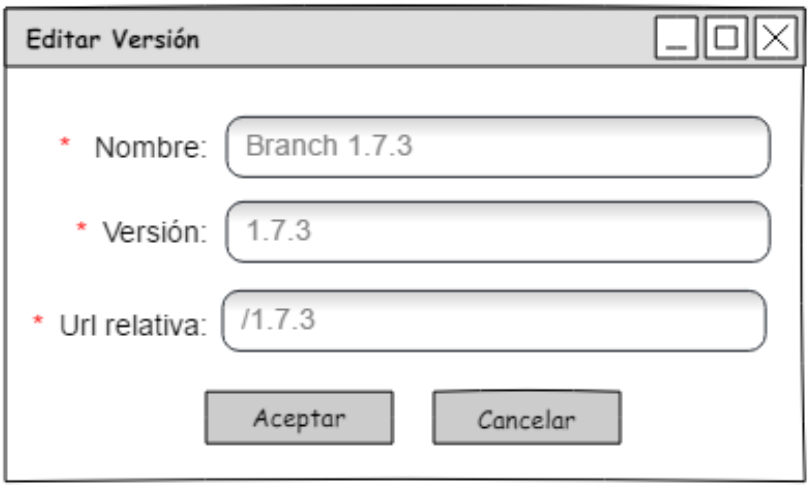


Figura 3.6: Prototipo: Editar Versión

f) Eliminar Versión

UC-0006	**Eliminar Versión**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se presiona sobre la opción de Eliminar versión.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de crear nuevo proyecto 3. El sistema muestra un popup con los datos del nuevo proyecto a realizar 4. El actor Desarrollador (ACT-0001) hace clic sobre la opción de Eliminar versión. 5. El sistema muestra un formulario de confirmación de eliminación de la versión. 6. El actor Desarrollador (ACT-0001) hace clic en Aceptar para eliminar la versión o en cancelar para cerrar el popup y no ejecutar ninguna operación.
Postcondiciones	1. Se elimina de la grilla de las versiones la versión eliminada
Excepciones	1. Si la versión a eliminar está asociada a algún cliente no se podrá eliminar y saldrá el mensaje: No se puede eliminar la versión porque está existen clientes con esta.
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.19: Caso de uso: Eliminar Versión

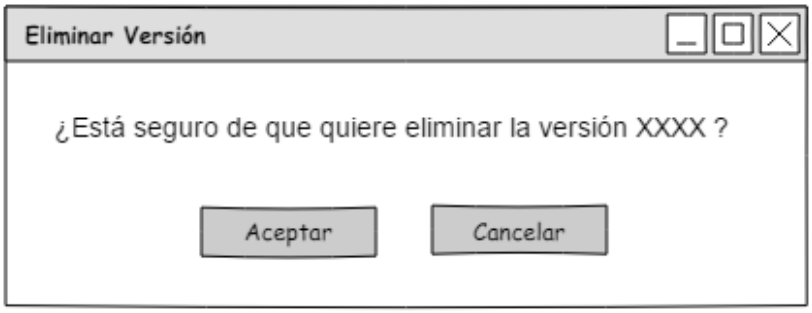


Figura 3.7: Prototipo: Eliminar Versión

g) Editar proyecto

UC-0007	**Editar proyecto**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se presiona sobre la opción de editar proyecto.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de editar proyecto cuando por lo menos haya uno seleccionado 3. El sistema muestra un popup con los datos del proyecto existente para su modificación. 4. El actor Desarrollador (ACT-0001) edita los datos solicitados y presiona aceptar
Postcondiciones	1. Se edita el proyecto exitosamente
Excepciones	1. Si el nombre asignado al proyecto ya está siendo usado por otro proyecto se genera el mensaje: "Ya existe un proyecto con este nombre" 2. Si no hay conexión con el Repositorio se emite el mensaje: "No hay conexión con el proyecto"
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.20: Caso de uso: Editar proyecto

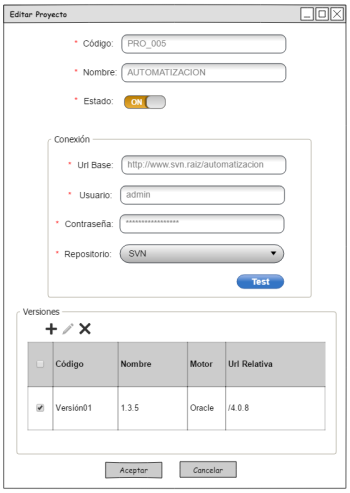


Figura 3.8: Prototipo: editar proyecto

h) Eliminar proyecto

UC-0008	**Eliminar proyecto**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se presiona sobre la opción de eliminar proyecto.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de eliminar proyecto cuando por lo menos haya uno seleccionado. 3. El sistema muestra un popup de confirmación de eliminación del proyecto. 4. El actor Desarrollador (ACT-0001) puede cancelar la operación y darle aceptar.
Postcondiciones	1. Si el usuario elige la opción Aceptar, Se debe eliminar el proyecto. 2. Si el usuario elige la opción cancelar, se vuelve a la vista inicial y el sistema no elimina el proyecto seleccionado
Excepciones	1. Si existen clientes configurados y que tienen dicho proyecto el sistema debe emitir mensaje: "No se puede eliminar el proyecto, ya existen clientes con esta aplicación"
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.21: Caso de uso: Eliminar proyecto

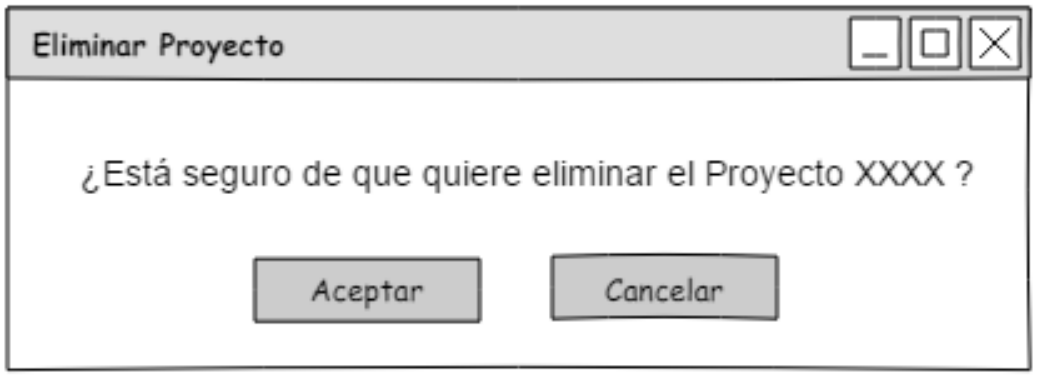


Figura 3.9: Prototipo: Eliminar proyecto

i) Buscar proyecto

UC-0009	**Buscar proyecto**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se filtra sobre la grilla de proyectos.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) puede buscar sobre cualquier de los filtros disponibles en la grilla de proyectos. 3. El sistema filtra los proyectos que cumplen la condición indicada por el usuario.
Postcondiciones	1. Se listan en la grilla los proyectos que cumplan con los filtros especificados por el usuario.
Excepciones	
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.22: Caso de uso: Buscar proyecto

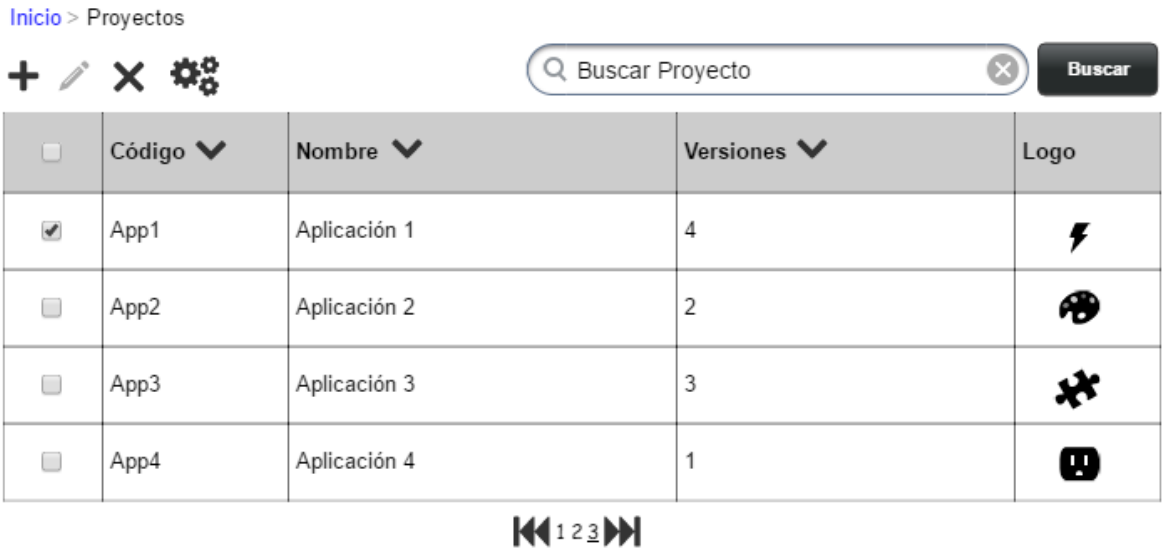


Figura 3.10: Prototipo: Buscar proyecto

j) Ver Control y Evolución

UC-0010	**Ver control y evolución**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se selecciona la opción de ver Control y evolución de proyecto.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) hace clic sobre la opción de Ver control y evolución de proyecto. 3. El sistema abre el popup con la información asociada a la aplicación y con los filtros para generar documento de evolución. 4. El actor Desarrollador (ACT-0001) selecciona la versión inicial y la final de las cuales quiere obtener la evolución. 5. El actor desarrollador (ACT-0001) selecciona los ítems que quiere que hagan parte del documento. 6. El actor Desarrollador (ACT-0001) hace clic sobre el botón Generar documento de evolución. 7. El sistema le genera un documento en PDF con la información de la evolución de la aplicación solicitada en el rango de versiones seleccionado.
Postcondiciones	1. El sistema descarga un documento de evolución de la aplicación selecconada.
Excepciones	
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.23: Caso de uso: Ver control y evolución



Figura 3.11: Prototipo: Ver control y evolución

Diagrama de estados: Configuración de Proyectos El siguiente diagrama inicia con al main de la aplicación o login de la aplicación y termina con un proyecto actualizado, eliminado o editado.

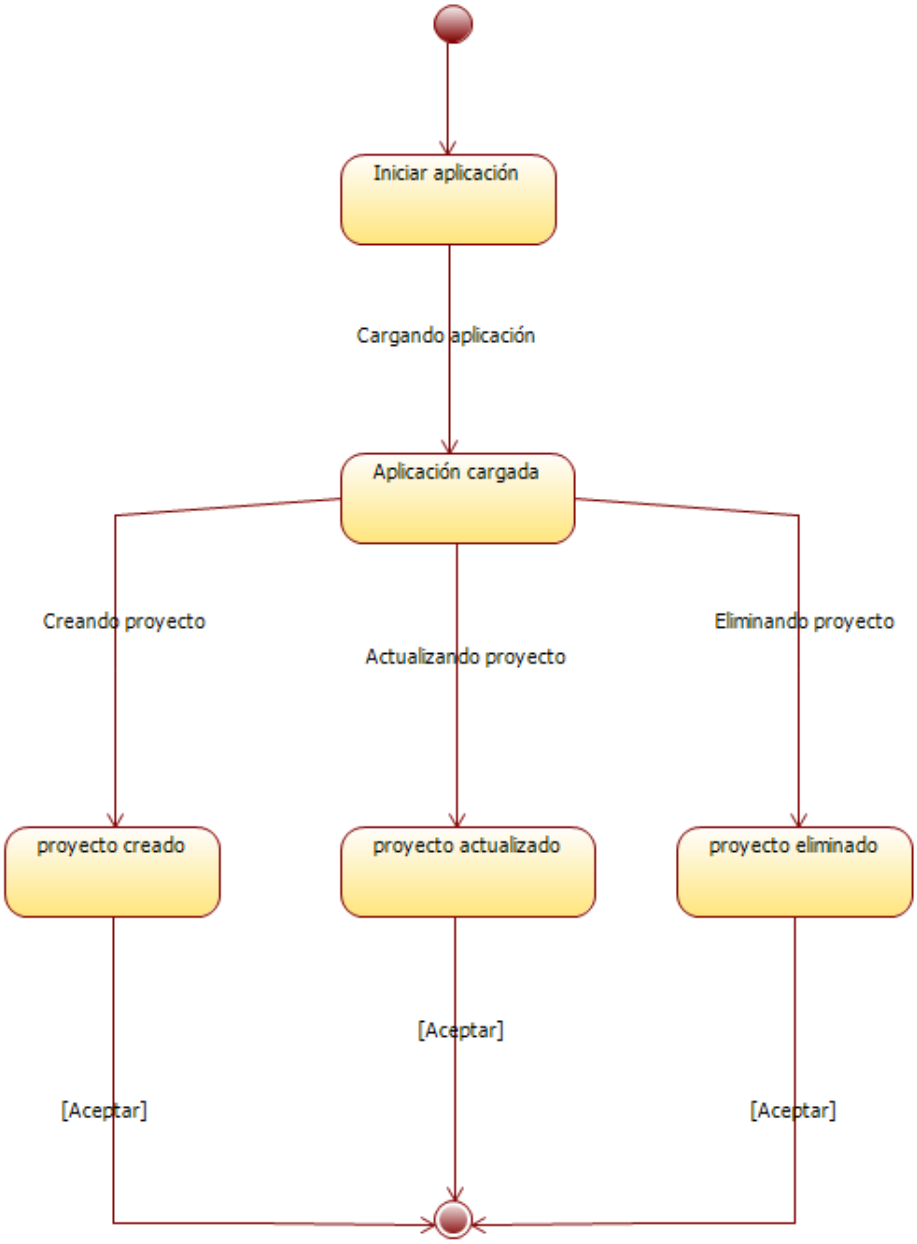


Figura 3.12: Diagrama de estados: Configuración de Proyectos

2. Administración de clientes
   Con la administración de clientes se busca tener control sobre los diferentes clientes de las diferentes aplicaciones de tal forma que permita, en cualquier momento, tener claro con que aplicación cuentan, que requieren y como obtenerlo.

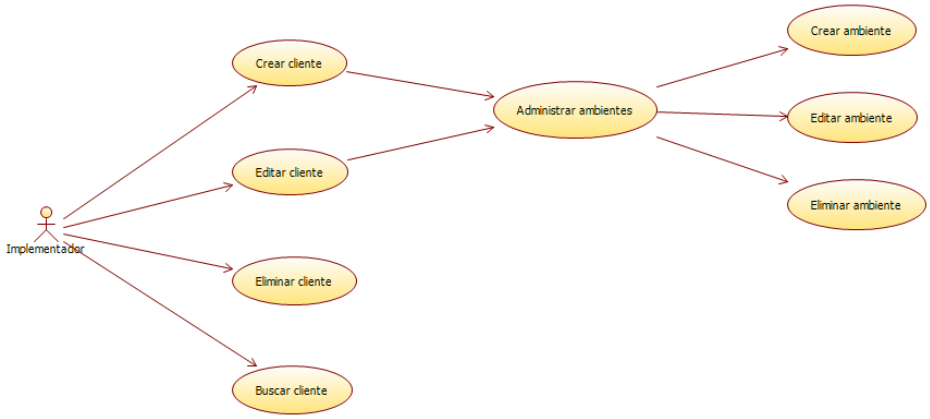


Figura 3.13: Diagrama de casos de uso: Administración de Clientes

a) Crear Cliente

UC-0001	**Crear cliente**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se crea un cliente.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de crear cliente. 3. El sistema muestra un popup con los datos del Cleinte que se desea crear.
Postcondiciones	1. Se crea el cliente correctamente.
Excepciones	1. Si el nombre del cliente ya existe el sistema arroja el siguiente mensaje: 'El nombre del cliente que intenta crear ya existe en el Sistema'.
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.24: Caso de uso: Crear Cliente

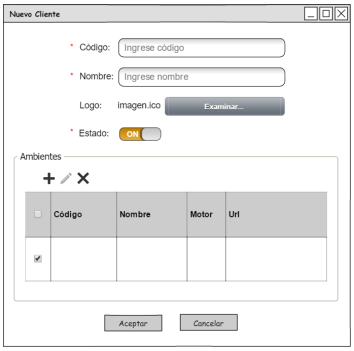


Figura 3.14: Prototipo: Crear Cliente

b) Crear Ambiente

UC-0002	**Crear ambiente**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se desea crear un ambiente para un cliente.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de crear cliente. 3. El sistema muestra un popup con los datos del Cliete a ingresar que se desea crear. 4. El actor Desarrollador (ACT-0001) diligencia los datos del cliente. 5. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de Crear ambiente. 3. El sistema muestra un popup con los datos a ingresar del ambiente a crear 6. El actor Desarrollador (ACT-0001) Diligencia los datos del ambiente y presiona aceptar
Postcondiciones	1. Se crea correctamente el ambiente del cliente a crear.
Excepciones	1. Si el nombre del ambiente a crear ya existe el sistema arroja el siguiente mensaje: 'El ambiente que intenta crear ya está registrado en el Sistema'.
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.25: Caso de uso: Crear Ambiente

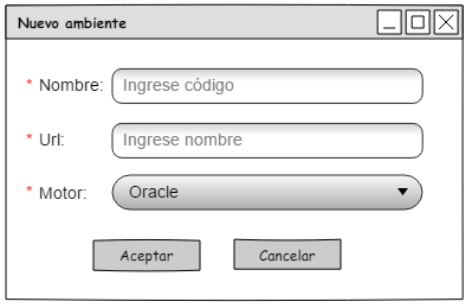


Figura 3.15: Prototipo: Crear ambiente

c) Editar Ambiente

UC-0003	**Editar ambiente**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se desea editar un ambiente para un cliente.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de crear/o editar cliente. 3. El sistema muestra un popup con los datos del Cliente a ingresar que se desea crear o editar. 4. El actor Desarrollador (ACT-0001) diligencia o edita los datos del cliente. 5. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de Editar ambiente después de haber seleccionado un ambiente de la grilla. 3. El sistema muestra un popup con los datos a editar del ambiente seleccionado. 6. El actor Desarrollador (ACT-0001) edita los datos del ambiente y presiona aceptar
Postcondiciones	1. Se edita correctamente los datos del ambiente seleccionado y se ven reflejados en la grilla de ambientes del cliente.
Excepciones	1. Si el nombre o url del ambiente a editar ya está siendo usado por otro ambiente el sistema arroja el mensaje: 'Ya existe un ambiente en el Sistema con la misma url o nombre del que desea editar'.
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.26: Caso de uso: Editar Ambiente

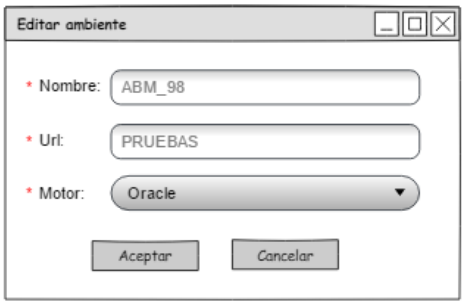


Figura 3.16: Prototipo: Editar ambiente

d) Eliminar Ambiente

UC-0004	**Eliminar ambiente**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se desea eliminar un ambiente para un cliente.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de crear/o editar cliente. 3. El sistema muestra un popup con los datos del Cliente a ingresar que se desea crear o editar. 4. El actor Desarrollador (ACT-0001) diligencia o edita los datos del cliente si desea. 5. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de Eliminar ambiente después de haber seleccionado un ambiente de la grilla. 3. El sistema muestra un popup para confirmar la eliminaciónd el ambiente. 6. El actor Desarrollador (ACT-0001) presiona clic sobre la opción aceptar del popup.
Postcondiciones	1. Se elimina correctamente los datos del ambiente seleccionado y se eliminan de la grilla de ambientes del cliente.
Excepciones	
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.27: Caso de uso: Eliminar Ambiente

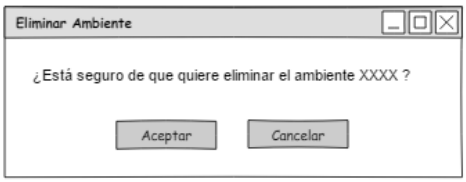


Figura 3.17: Prototipo: Eliminar Ambiente

e) Editar Cliente

UC-0005	**Editar cliente**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se Edita un cliente.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de editar cliente. 3. El sistema muestra un popup con los datos del cliente que se desea editar. 4. El usuario edita los datos del cliente que requiera y presiona aceptar. 5. El usuario visualiza la grilla con los nuevos datos del cliente editado.
Postcondiciones	1. Se edita el cliente correctamente.
Excepciones	1. Si el nombre del cliente ya existe el sistema arroja el siguiente mensaje: 'El nombre del cliente que intenta editar ya existe en el sistema'. Para esto el cliente debe cambiar los datos y volver a presionar el botón aceptar.
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.28: Caso de uso: Editar Cliente

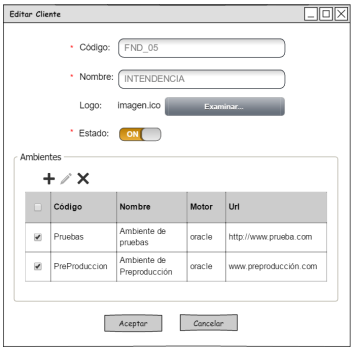


Figura 3.18: Prototipo: Editar Cliente

f) Eliminar Cliente

UC-0006	**Eliminar cliente**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se desea eliminar un cliente.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación. 2. El actor Desarrollador (ACT-0001) presiona clic sobre la pestaña de Clientes. 3. El actor Desarrollador (ACT-0001) presiona clic sobre la opción de eliminar cliente. 4. El sistema muestra un popup de alerta con el nombre del cliente que se desea eliminar. 5. El usuario presiona aceptar o cancelar la eliminación.
Postcondiciones	1. Se elimina el cliente correctamente.
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.29: Caso de uso: Eliminar Cliente

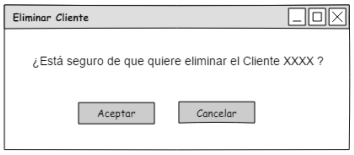


Figura 3.19: Prototipo: Eliminar Cliente

g) Buscar Cliente

UC-0007	Buscar cliente
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se desea buscar un cliente.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación. 2. El actor Desarrollador (ACT-0001) presiona clic sobre la pestaña de Clientes. 3. El actor Desarrollador (ACT-0001) busca clientes filtrando por la grilla de clientes. 4. El sistema muestra los clientes que cumplen con los filtros diligenciados por el usuario.
Postcondiciones	1. Si existe cliente que cumpla con los filtros diligenciados por el usuario, este se muestra en la grilla.
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.30: Caso de uso: Buscar Cliente



Figura 3.20: Prototipo: Buscar Cliente

Diagrama de estados: Administración de Clientes

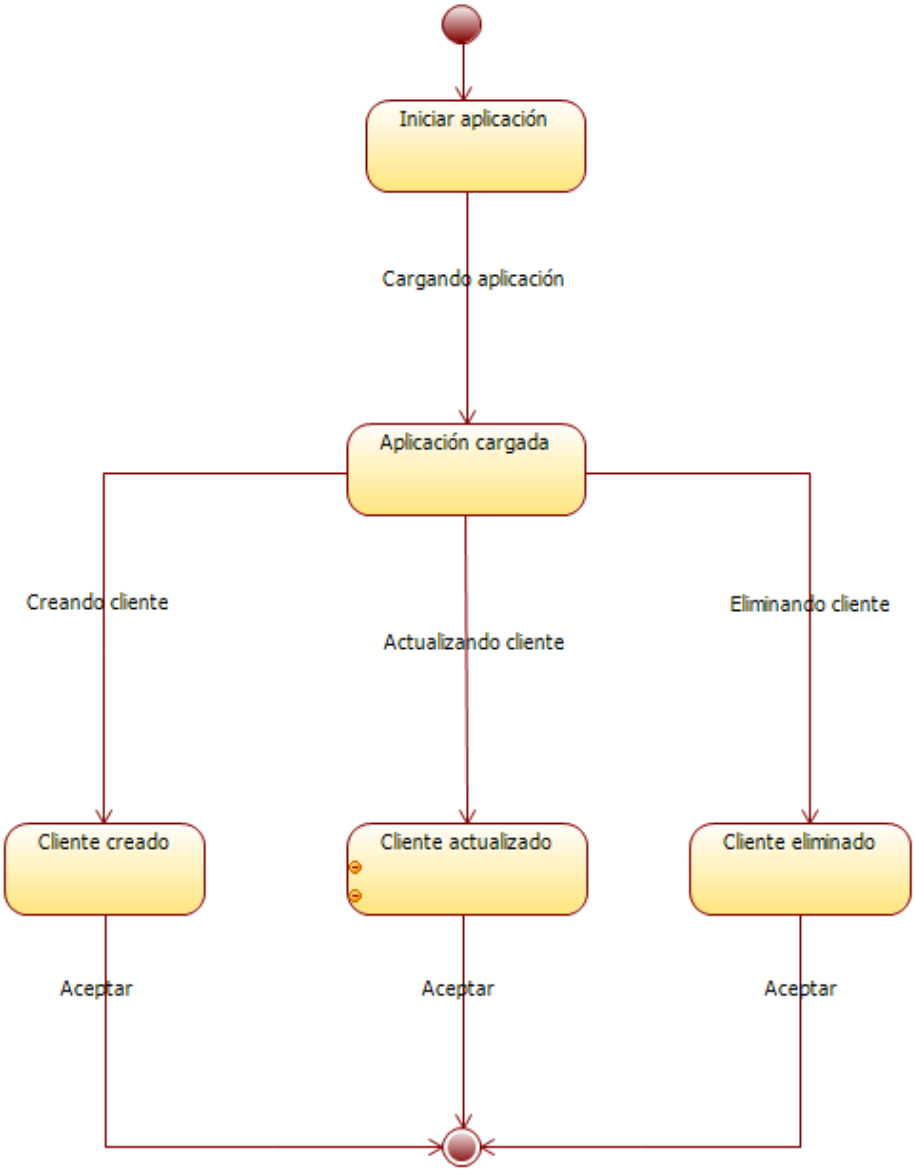


Figura 3.21: Diagrama de estados: Administración de clientes

3. Clientes y Proyectos
   Con el manejo de las aplicaciones y/o proyectos de los clientes se busca que se tenga control sobre las versiones de las aplicaciones instaladas en los diferentes ambientes de los clientes.

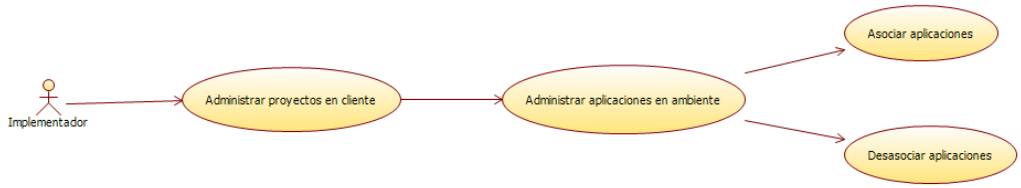


Figura 3.22: Diagrama de casos de uso: Clientes y proyectos

a) Administrar Proyectos en cliente

UC-0001	**Administrar proyectos en cliente**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se administran proyectos en cliente.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) hace clic sobre la pestaña de Clientes y Proyectos. 3. El Actor Desarrollador (ACT-0001) visualiza una grilla con la información de los clientes y la cantidad de ambientes que tiene configurado. 4. El actor Desarrollador (ACT-0001) puede seleccionar la opción de Administrar los proyectos en el Cliente.
Postcondiciones	
Excepciones	
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.31: Caso de uso: Administrar proyectos en Cliente

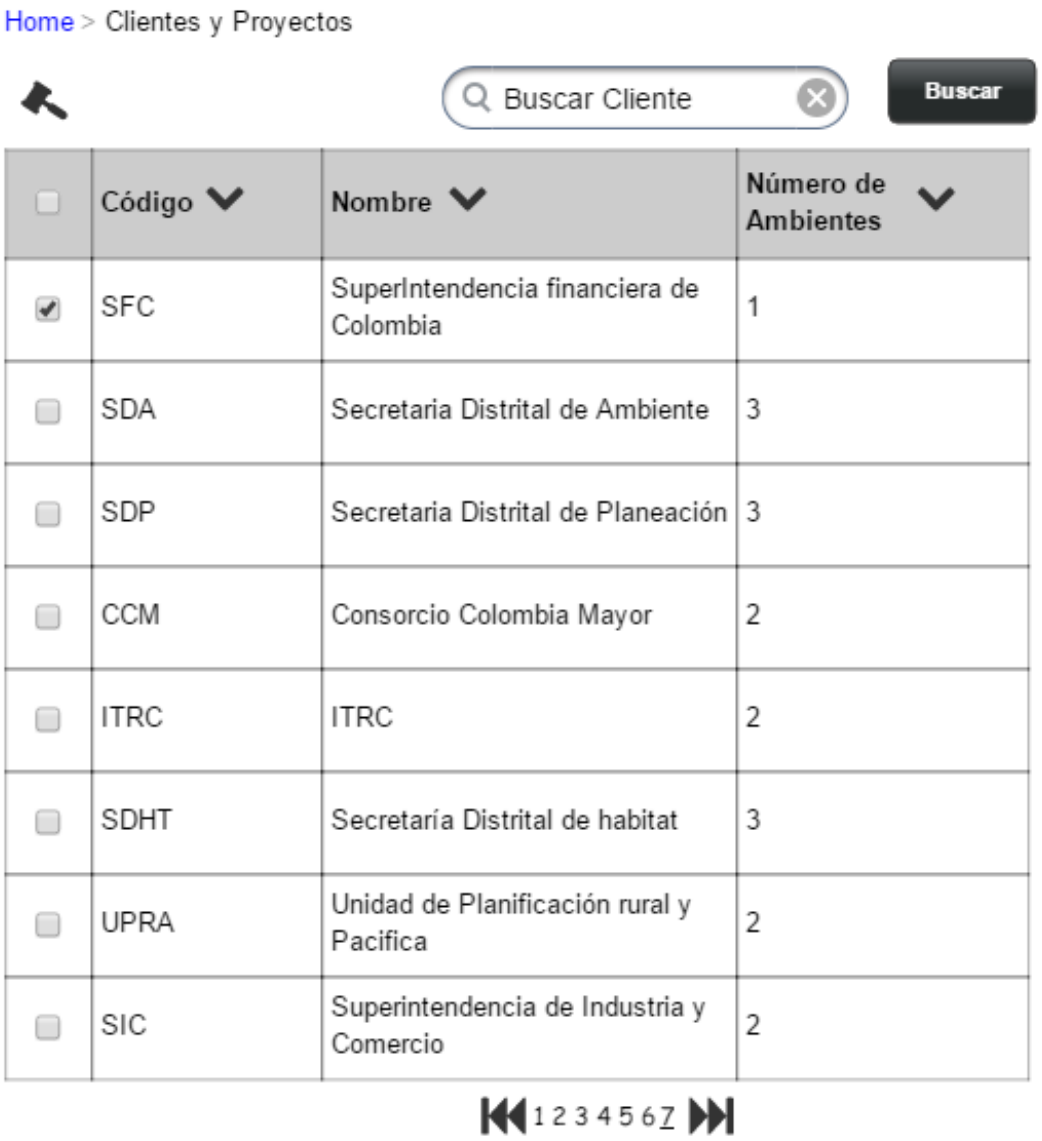


Figura 3.23: Prototipo: Administrar proyectos en cliente

b) Administrar Aplicaciones en ambientes

UC-0002	**Administrar aplicaciones en ambientes**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se administran aplicaciones en clientes.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1 El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) hace clic sobre la pestaña de Clientes y Proyectos. 3. El Actor Desarrollador (ACT-0001) presiona clic sobre la opción de Proyectos en clientes. 4. El sistema muestra un popup con la información de los ambientes del cliente seleccionado y las aplicaciones que tiene en cada una de estas. 5. El Actor puede ver en cualquier momento las aplicaciones que tiene un cliente en un ambiente en particular dando clic sobre la opción de ver sobre la grilla de Ambientes. 6. El Sistema muestra las aplicaciones instaladas en el ambiente previamente selecionado y la versión actual de esta. Esto lo puede ver en la sección de Aplicaciones por ambiente.
Postcondiciones	
Excepciones	
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.32: Caso de uso: Administrar aplicaciones en ambiente

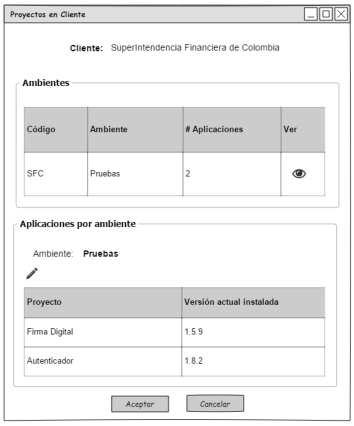


Figura 3.24: Prototipo: Administrar aplicaciones en ambiente

c) Asociar aplicaciones

UC-0003	**Asociar aplicaciones**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se deseen asociar aplicaciones a ambientes de clientes.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1. El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) hace clic sobre la pestaña de Clientes y Proyectos. 3. El Actor Desarrollador (ACT-0001) presiona clic sobre la opción de Proyectos en clientes. 4. El sistema muestra un popup con la información de los ambientes del cliente seleccionado y las aplicaciones que tiene en cada una de estas. 5. El Actor puede ver en cualquier momento las aplicaciones que tiene un cliente en un ambiente en particular dando clic sobre la opción de ver sobre la grilla de Ambientes. 6. El Sistema muestra las aplicaciones instaladas en el ambiente previamente selecionado y la versión actual de esta. Esto lo puede ver en la sección de Aplicaciones por ambiente. 7. El actor hace clic sobre la opción de Editar aplicaciones en ambiente 8. El Sistema muestra una ventana donde el usuario podrá seleccionar las aplicaciones para dicho ambiente. 9. El usuario escoge si desea asociar la aplicación al ambiente (pasar la aplicación de la izquierda a la parte derecha). 10. Si el usuario asocia aplicaciones estas se deben ver reflejadas en la ventana del punto 6.
Postcondiciones	La nueva configuración de aplicaciones se debe ver reflejada en la grilla de la ventana de proyectos en cliente.
Excepciones	
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.33: Caso de uso: Asociar aplicaciones

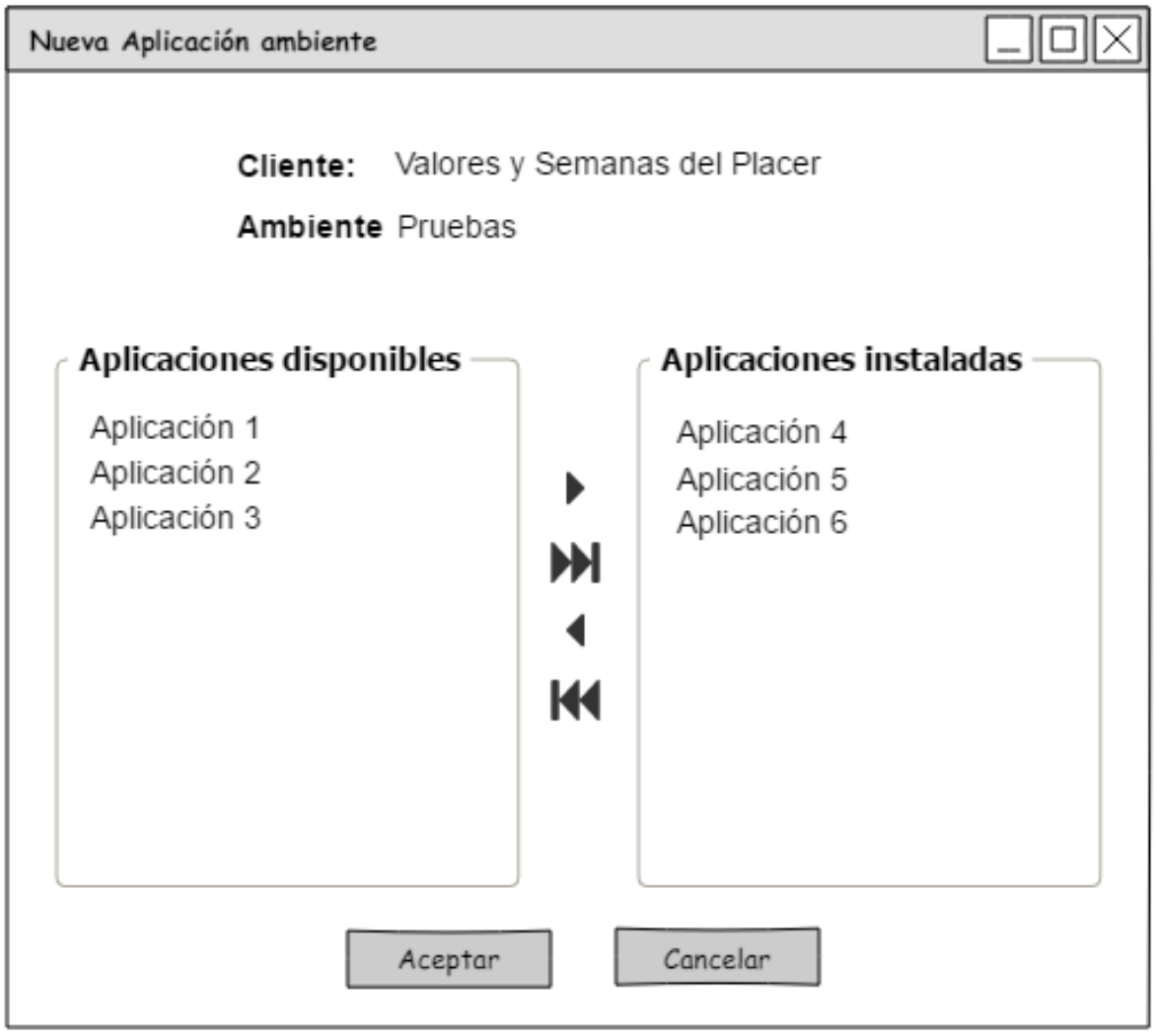


Figura 3.25: Prototipo: Asociar aplicaciones

d) Desasociar aplicación

UC-0004	**Desasociar aplicación**
Versión	1.0 (02/09/2016)
Autores	Ing. Franklin Candanoza
Fuentes	Ing. de Desarrollo
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se deseen desasociar aplicaciones a ambientes de clientes.
Precondiciones	1. El usuario debe estar logeado en la aplicación.
Secuencia de pasos	1. El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador (ACT-0001) hace clic sobre la pestaña de Clientes y Proyectos para desasociar aplicaciones. 3. El Actor Desarrollador (ACT-0001) presiona clic sobre la opción de Proyectos en clientes. 4. El sistema muestra un popup con la información de los ambientes del cliente seleccionado y las aplicaciones que tiene en cada una de estas como tambien las aplicaciones que no tiene instalada pero que puede instalar. 5. El Actor puede ver en cualquier momento las aplicaciones que tiene un cliente en un ambiente en particular dando clic sobre la opción de ver sobre la grilla de Ambientes y se visualizará la ventana con las aplicaciones registradas en el ambiente del cliente. 6. El Sistema muestra las aplicaciones instaladas en el ambiente previamente selecionado y la versión actual de esta. Esto lo puede ver en la sección de Aplicaciones por ambiente. 7. El actor hace clic sobre la opción de Editar aplicaciones en ambiente 8. El Sistema muestra una ventana donde el usuario podrá seleccionar las aplicaciones para dicho ambiente. 9. El usuario escoge si desea asociar la aplicación al ambiente (pasar la aplicación de la derecha a la parte izquierda). 10. El usuario presiona el botón aceptar para relacionar las aplicaciones que registró al lado derecho, es decir, para asociar las aplicaciones con el ambiente del cliente seleccionado previamente en la grilla de proyectos y clientes.
Postcondiciones	La aplicación desasociada no debe ver reflejada en la grilla de la ventana de proyectos en cliente, es decir, debe desaparecer dicha relación entre el ambiente y la aplicación previamente desasociada. Tener en cuenta que solo se desasociará en caso de no tener trazabilidad en el Sistema
Excepciones	1. Si la aplicación que se desasocia ya tiene trazabilidad en el Sistema, este arrojará un mensaje indicando tal situación.
Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Cuadro 3.34: Caso de uso: Desasociar aplicaciones

3.2.2. Modelo Relacional Base de Datos

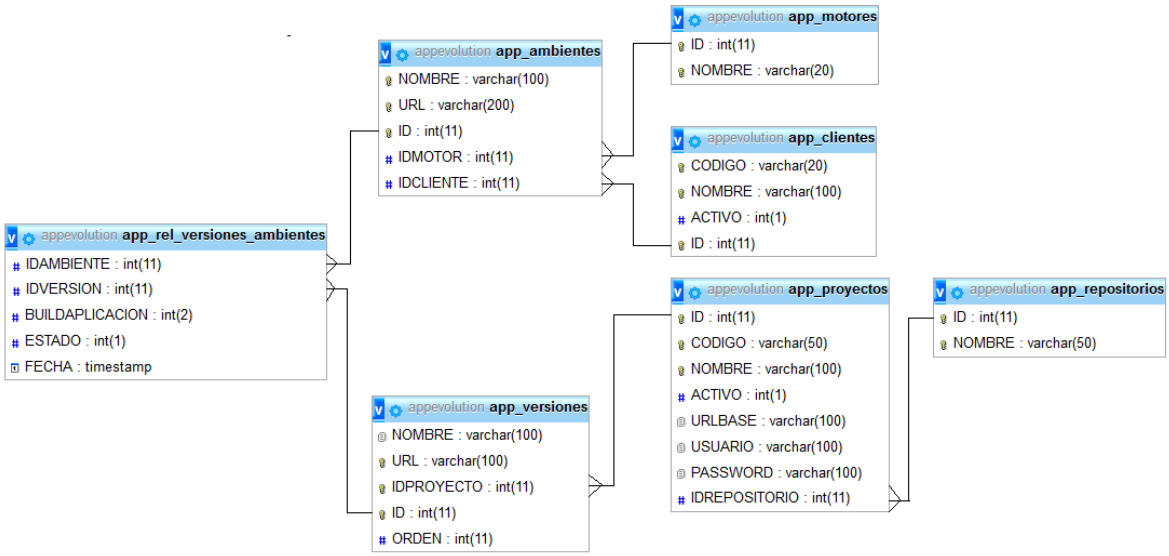


Figura 3.26: Modelo Relacional Base de Datos

3.3. fase de construcción

Con el uso de los modelos de análisis y diseño como entrada, en la fase de construcción fueron desarrollados los componentes de software pretendiendo que cada caso de uso sea operativo para los usuarios finales. El código fuente de todas las características y funciones se evidencian en el anexo (appEvolution_Code.zip).

3.4. fase de transición

3.4.1. Implementación

Para la implementación es necesario contar con una Base de datos sobre la cual se puedan correr o ejecutar los scripts que se encuentran en los anexos.

De igual forma, es necesario, para el funcionamiento de la aplicación, configurar el acceso a la base de datos. Esto se hace en un archivo de propiedad llamado **start.properties** que se encuentra ubicado en la carpeta del usuario actual. Las propiedades aquí registradas son:

- username (El valor está en base 64)
- password (El valor está en base 64)
- url: url de la base de datos a conectarse (El valor está en base 64)
- driver: driver asociado a la base de datos a conectar
- pathDim: Url relativa donde se encuentra el archivo xml donde se guardar los cambios por parte de los desarrolladores.

Otro de los ítems importantes a tener en cuenta es el archivo de configuración que se debe ubicar en la raiz de los proyectos de la siguiente forma:

1. En la raiz de cada proyecto crear una carpeta llamada **evolucion**
2. Dentro de esta carpeta crear un archivo llamado **evolucion.xml** y este mismo debe estar configurado en el archivo start.properties en la propiedad **pathDim** que hace referencia a la url relativa a partir de la raiz del proyecto.

El archivo **evolucion.xml** tendrá la siguiente estructura:

```
<?xml version="1.0" encoding='UTF-8'?>
<changeLog>
    <build numero="0">
        <funcionalidades>
            <funcionalidad>
                <descripcion>
                    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                </descripcion>
            </funcionalidad>
        </funcionalidades>
        <mejoras>
            <mejora>
                <descripcion>
                    BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
                </descripcion>
            </mejora>
        </mejoras>
    <errores>
            <error>
                <descripcion>
                    DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
                </descripcion>
            </error>
        </errores>
        <configuraciones>
            <configuracion>
                <descripcion>
                    EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
                </descripcion>
            </configuracion>
        </configuraciones>
    </build>
 <build numero="1">
        <mejoras>
            <mejora>
                <descripcion>
                    FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
                </descripcion>
            </mejora>
        </mejoras>
        <errores>
            <error>
                <descripcion>
                    GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
                </descripcion>
            </error>
        </errores>
    </build>
</changeLog>
```

Figura 3.27: Archivo evolucion.xml para registro de cambios

De igual forma, se debe tener en cuenta que al momento de iniciar la aplicación el sistema valida que todas las propiedades estén correctas. En caso de que falte alguna se informa al usuario para que la configure e inicie nuevamente la aplicación. Esto nos garantiza que la aplicación al momento de estar funcionando no presente fallos por configuraciones faltantes que el usuario no tuvo en cuenta al momento de su instalación.

Adicional a esto, se crea un usuario por defecto para iniciar la aplicación el cual es admin-admin.

La versión mínima de Java requerida es 7.

Como dato adicional, si el usuario desea incluir mas etiquetas sobre el xml inicial de evolucion.xml puede definirlas en un archivo de configuracion llamado **baseitems.xml** que está ubicado en la carpeta

del usuario que se encuentra logeado. Aquí podrán agregar etiquetas que se deseen aparezcan en el reporte de implementación si y solo sí estas estén presentes en los archivos de evolución de cada uno de los proyectos registrados en el repositorio.

3.4.2. Pruebas de Aceptación

En este caso las pruebas de aceptación las centramos en las características y funcionalidades generales del sistema de información, estas derivaron de los casos de uso implementados.

Pruebas de Aceptación: Configuración de Proyectos

Caso de prueba 001	**CRUD Proyectos**
Número caso de uso	UC-0001, 02, 03, 04, 05, 06, 07, 08, 09
Autores	Ing. Carlos Alvarez
Actor	Desarrollador
Descripción	1. El sistema muestra la vista inicial de la aplicación 2. El actor Desarrollador presiona clic sobre las opciones: crear, editar, eliminar o buscar proyecto. 3. Según la opción que se elija, el sistema muestra las pantallas definidas en los prototipos. 4. Al elegir las opciones crear o editar proyecto se debe asociar la conexión con el proyecto. 5. En la opción editar proyecto - administrar versiones, se desarrollan las opciones de versionamiento.
Condiciones de ejecución	El usuario debe estar logeado en la aplicación.
Resultado esperado	El sistema debe permitir crear, leer, actualizar y eliminar proyectos, configurar conexiones y realizar la administración de las versiones del software.
Evaluación	Funciona correctamente

Cuadro 3.35: Caso de prueba: CRUD Proyectos

Caso de prueba 002	**Control y Evolución**
Número caso de uso	UC-0010
Autores	Ing. Carlos Alvarez
Actor	Desarrollador, Implementador, Área Comercial
Descripción	1. El sistema muestra la vista inicial de la aplicación 2. El sistema muestra la información correspondiente a una aplicación y agrega filtros para generar el documento de evolución.
Condiciones de ejecución	El usuario debe estar logeado en la aplicación.
Resultado esperado	Generar un documento en formato PDF que contiene la información de evolución de la aplicación seleccionada.
Evaluación	Funciona correctamente

Cuadro 3.36: Caso de prueba: Control y Evolución

Pruebas de Aceptación: Administración de clientes

Caso de prueba 003	CRUD Clientes
Número caso de uso	UC-0001, 02, 03, 04, 05, 06, 07
Autores	Ing. Carlos Alvarez
Actor	Implementador
Descripción	1. El sistema muestra la vista inicial de la aplicación 2. El actor presiona clic sobre las opciones: crear, editar, eliminar o buscar cliente. 3. Según la opción que se elija, el sistema muestra las pantallas definidas en los prototipos. 4. Al elegir las opciones crear o editar cliente se deben habilitar las opciones de administración de ambientes.
Condiciones de ejecución	El usuario debe estar logeado en la aplicación.
Resultado esperado	El sistema debe permitir crear, leer, actualizar y eliminar clientes y realizar la administración de los ambientes.
Evaluación	Funciona correctamente

Cuadro 3.37: Caso de prueba: CRUD Clientes

Pruebas de Aceptación: Clientes y Proyectos

Caso de prueba 004	Clientes y proyectos
Número caso de uso	UC-0001, 02, 03, 04
Autores	Ing. Carlos Alvarez
Actor	Implementador
Descripción	1. El sistema muestra la vista inicial de la aplicación 2. El actor presiona clic sobre la opción Administrar proyectos en cliente. 3. El sistema habilita la opción Administrar aplicaciones en ambiente que muestra la información correspondiente a los clientes y el número de ambientes que tiene configurados. 4. Partiendo de la grilla anterior se observa que aplicaciones tiene instaladas un cliente en particular, haciendo posible asociar o desasociar aplicaciones.
Condiciones de ejecución	El usuario debe estar logeado en la aplicación.
Resultado esperado	El sistema debe permitir asociar o desasociar aplicativos segun cliente y ambiente seleccionado.
Evaluación	Funciona correctamente

Cuadro 3.38: Caso de prueba: Clientes y proyectos

3.5. fase de producción



CON APPEVOLUTION SE NOS FACILITA EL CONTROL Y LA EVOLUCIÓN DE LAS APLICACIONES DE SOFTWARE

A través de las siguientes indicaciones podemos ver como la herramientas no ayudará a llevar control sobre las aplicaciones y ver cómo evolucionan facilitando la implementación y el seguimiento de estas.

1. Vista inicial de la aplicación: Esta vista nos permitirá registrar, editar o eliminar los clientes de nuestras aplicaciones. Estos son necesarios dado que a dichos clientes son a los que se les realiza actualización de las aplicaciones y es a ellos, en gran medida, quienes están interesados en los ítems de configuración que contiene el aplicativo que les instala.

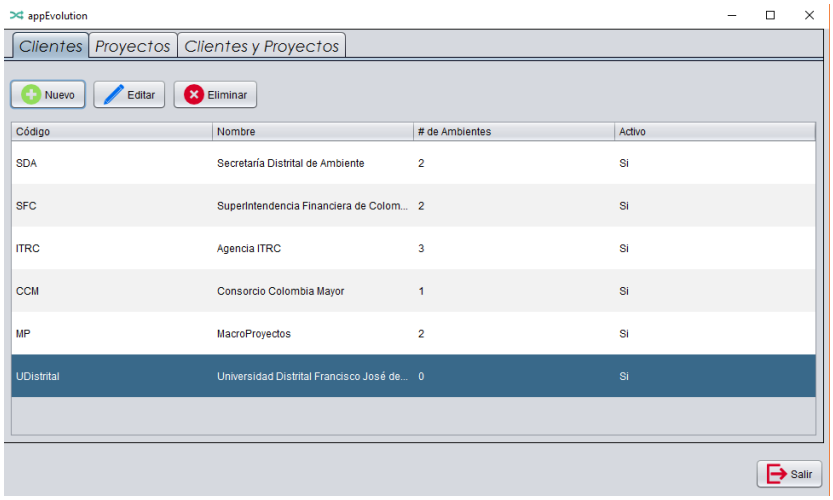


Figura 3.28: Módulo de Clientes

a) Con la opción “Nuevo” se brinda la posibilidad de que se cree un nuevo cliente que será portador de ambientes y de aplicaciones en estas. Los datos a crear son el código, el nombre y el estado que indica si está activo o no.

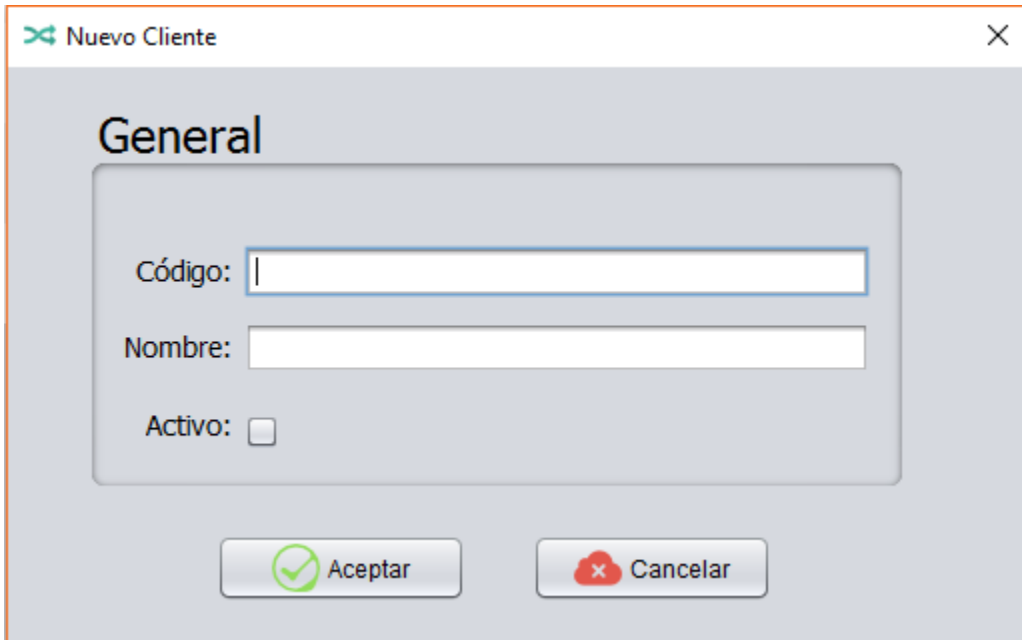


Figura 3.29: Registro de nuevo cliente

b) Con la opción “Editar” se brinda la posibilidad de que se puedan editar los datos actuales de un cliente previamente registrado. Se podrá inactivar, cambiar su código o en su defecto el nombre. Adicional a esto se podrán registrar los ambientes que tenga el cliente, es decir, ambientes de pruebas, producción, desarrollo, entre otros. Esto incluye url, motor y el nombre de este.

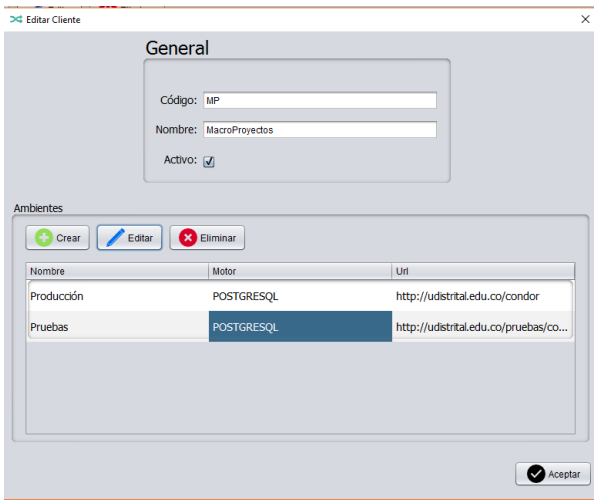


Figura 3.30: Editar cliente

Los ambientes a crear tendrán datos como nombre, motor y url que permitirán identificar claramente los ambientes asociados al cliente seleccionado. Esto nos permitirá identificar, de alguna u otra forma, los ambientes de los clientes y el modo en el que se encuentran (producción, desarrollo,

pruebas, etc). Esto es necesario dado que una aplicación puede estar en diferentes versiones en distintos ambientes. Por tal motivo se requiere dicha información de los clientes.

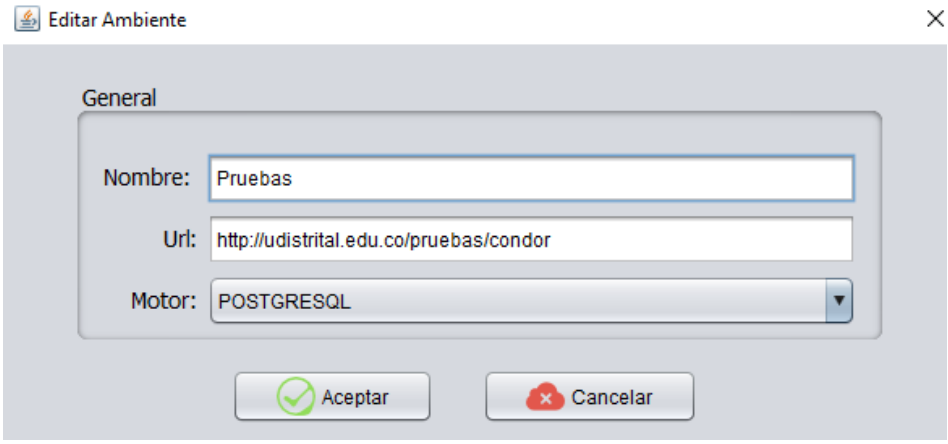


Figura 3.31: Editar ambiente

c) Con la opción de "Eliminar" se podrán eliminar los clientes que se desean siempre y cuando no tengan registrada trazabilidad en el Sistema, es decir, actualizaciones, ambientes, etc.

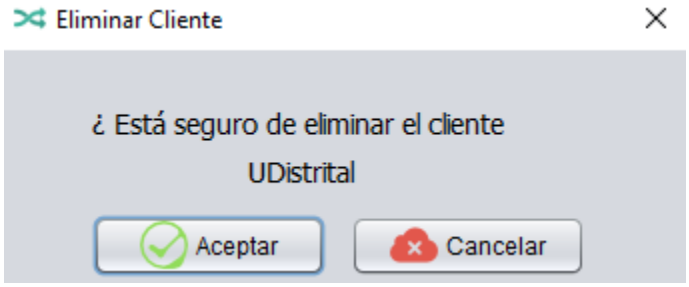


Figura 3.32: Eliminar ambiente

2. Vista Proyectos: La vista de proyectos nos permitirá registrar los proyectos o aplicaciones sobre los cuales queremos llevar control y seguimiento, es decir queremos conocer su evolución. De igual forma, queremos llevar control con la instalación de estas en los clientes.

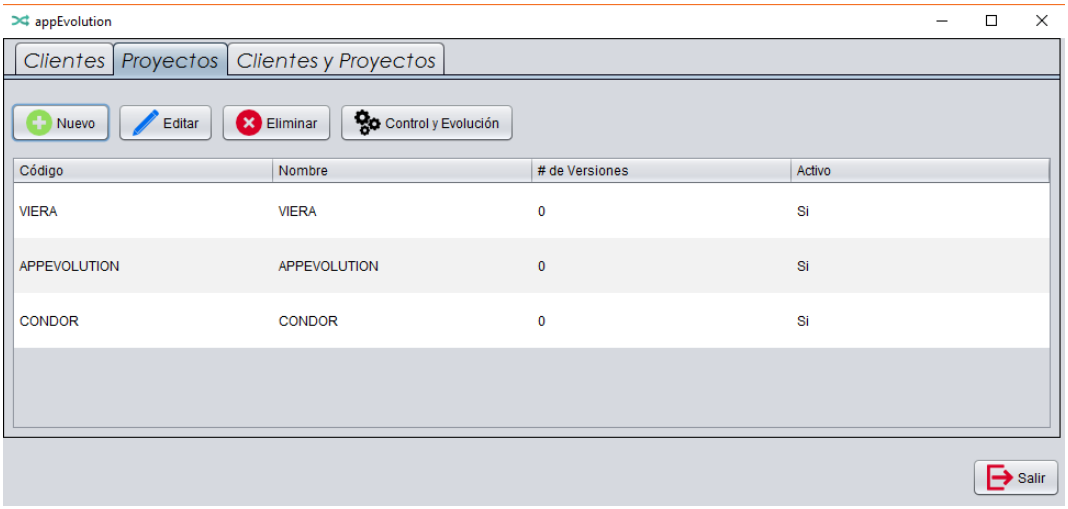


Figura 3.33: Vista de Proyectos

a) Con la opción "nuevo" se podrán crear proyectos en la aplicación. Estos proyectos requieren que tengan algún repositorio donde se almacene la información referente a su evolución.



Figura 3.34: Registro de un nuevo Proyecto

En esta vista podemos encontrar los datos del proyecto que estamos deseando crear. Estos son un código, un nombre y el estado. Adicional a esto se solicitan los datos de conexión al repositorio donde está el proyecto. Esto es importante dado que es desde dicho repositorio donde se extraerá la información diligenciada por los desarrolladores sobre los cambios que van teniendo las aplicaciones. De igual forma, después de configurada la conexión al repositorio se brinda la opción de poder probar la conexión para verificar que efectivamente se tiene conexión con dicho proyecto.

b) Con la opción de “Editar” se podrá editar los datos del proyecto previamente registrados. Aunque lo no menos importante de esta vista es que permitirá registrar la versiones que tiene dicha aplicación, es decir, podemos decidir que versiones vamos a controlar y cuales vamos a analizar su evolución.

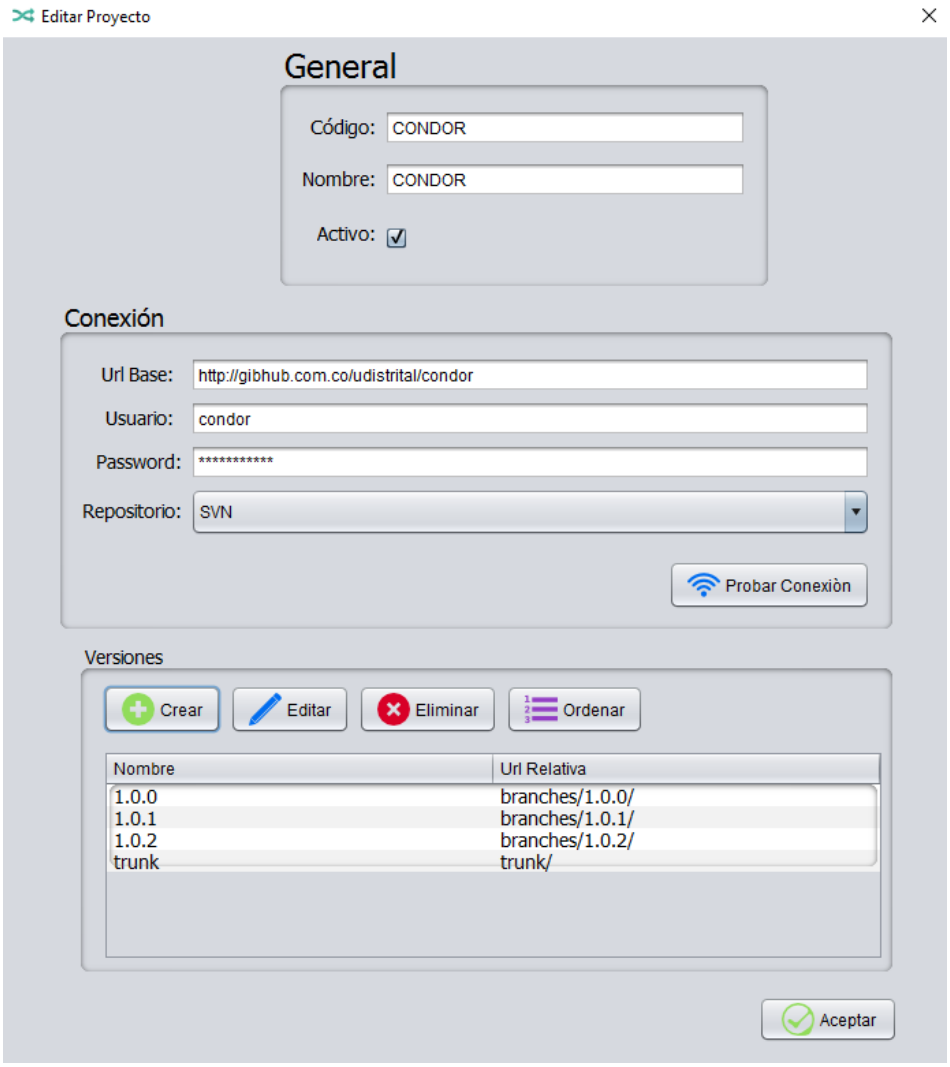


Figura 3.35: Editar proyecto

Con la administración de las versiones se busca tener control más claro sobre las versiones de la aplicación y tener el control de dichas versiones en los clientes. De igual forma, es necesario tener en cuenta que varios clientes pueden tener diferentes versiones de una aplicación, por lo tanto se hace necesario poder controlar la versiones para facilitar las actualizaciones de las aplicaciones en los clientes desde un punto inicial hacia un punto final.
Adicional a esto es necesario dejar en claro que las versiones tienen un orden específico en el tiempo y ese orden debemos determinarlo y definirlo en la aplicación AppEvolution a través de la opción ordenar que me permitirá definir la secuencia de las versiones de las distintas aplicaciones.



Figura 3.36: Ordenar versiones de proyecto

c) Con la opción de "Eliminar" se podrá eliminar un proyecto registrado previamente en el Sistema teniendo que en cuenta que solo se eliminará si no tiene trazabilidad en el Sistema.

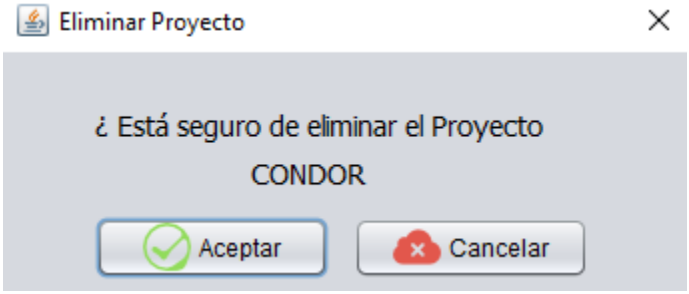


Figura 3.37: Eliminar un proyecto

d) Con la opción de "Control y Evolución" y considerada una de las más importantes características de este módulo ya que permite identificar la evolución de las aplicaciones entre dos versiones y los ítems de configuración que trae consigo.

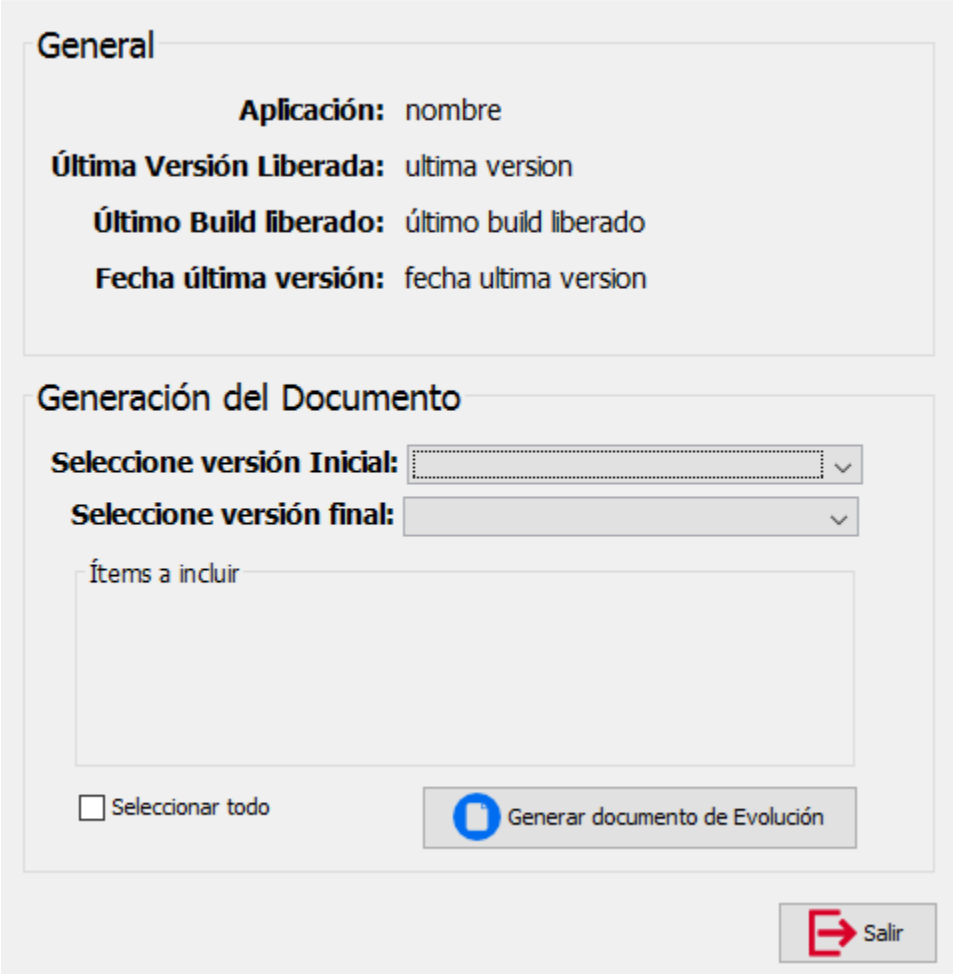
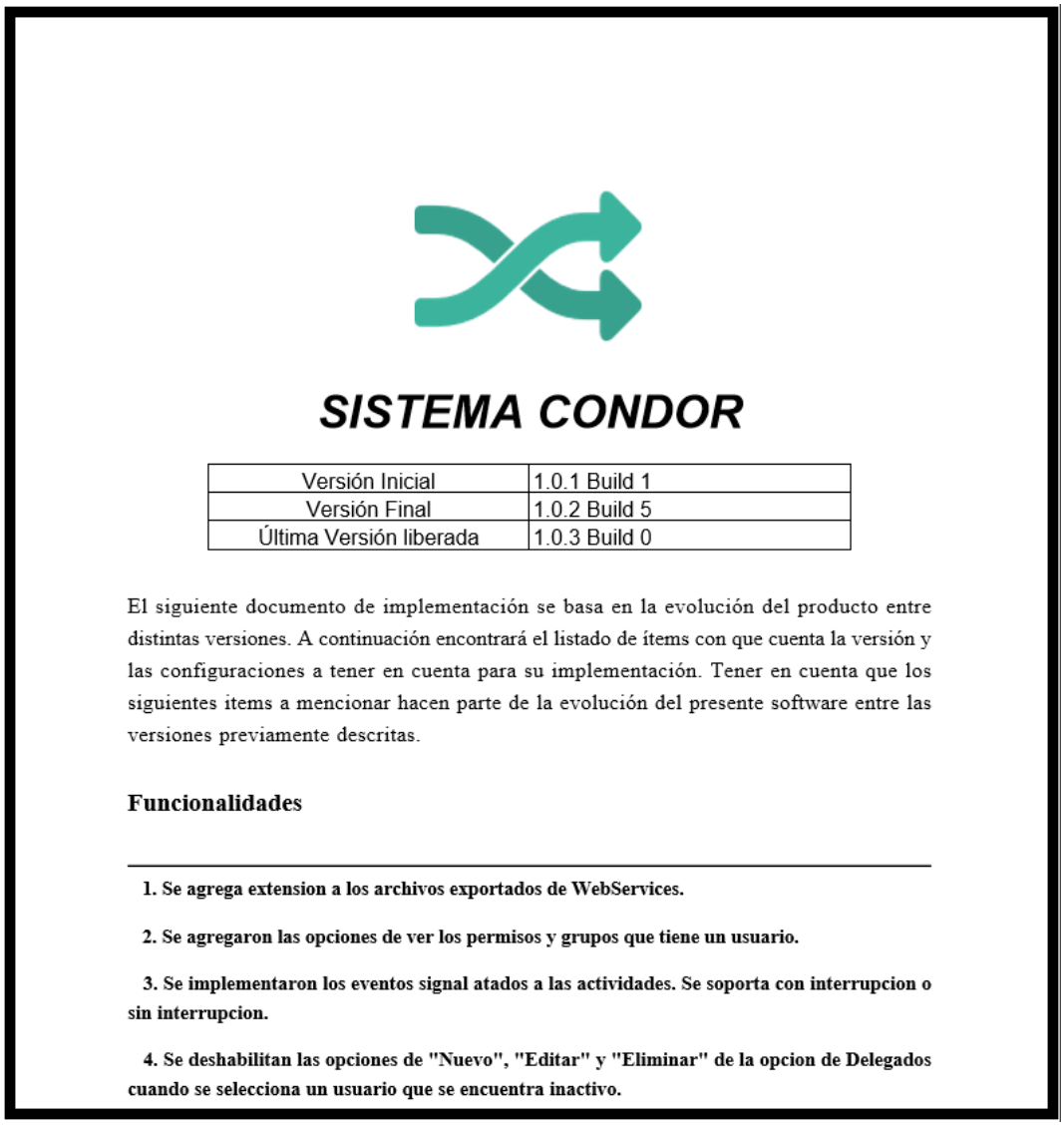


Figura 3.38: Control y evolución de las aplicaciones

La vista de evolución y control nos permitirá consultar y generar un documento de evolución que nos indica que ítems se han generado entre una versión inicial y una versión de la aplicación. Esto quiere decir, que entre dos versiones se pueden identificar los siguientes ítems que igual pueden ser dinámicos dependiendo de lo que el equipo de desarrollo vea necesario. Los ítems iniciales son:

- Mejoras
- Funcionalidades
- Corrección de errores
- Configuraciones
- Cambios en Base de datos

Al generar el documento y de acuerdo a los ítems seleccionados se generará un documento con todas las características nuevas que hay entre una versión y otra. Un ejemplo del documento generado (PDF) podría ser:



SISTEMA CONDOR

Versión Inicial	1.0.1 Build 1
Versión Final	1.0.2 Build 5
Última Versión liberada	1.0.3 Build 0

El siguiente documento de implementación se basa en la evolución del producto entre distintas versiones. A continuación encontrará el listado de ítems con que cuenta la versión y las configuraciones a tener en cuenta para su implementación. Tener en cuenta que los siguientes items a mencionar hacen parte de la evolución del presente software entre las versiones previamente descritas.

Funcionalidades

---

1. Se agrega extension a los archivos exportados de WebServices.

2. Se agregaron las opciones de ver los permisos y grupos que tiene un usuario.

3. Se implementaron los eventos signal atados a las actividades. Se soporta con interrupcion o sin interrupcion.

4. Se deshabilitan las opciones de "Nuevo", "Editar" y "Eliminar" de la opcion de Delegados cuando se selecciona un usuario que se encuentra inactivo.

Figura 3.39: Documento de evolución generado por la aplicación

Parte III

CIERRE DE LA INVESTIGACIÓN

CONCLUSIONES

Con la realización de este trabajo de grado es posible concluir que, en las empresas de desarrollo de software, aunque se cuente con sistemas de información que brindan soporte a las operaciones de construcción de aplicaciones, la documentación sigue siendo una de las principales falencias que se presentan a lo largo del ciclo de vida de un sistema informático. Esto plantea consecuencias como perdida de información y trazabilidad que se mitigan con la implementación de APPEVOLUTION.

Podemos concluir entonces que para garantizar la funcionalidad de un sistema de información es vital llevar correctamente el seguimiento de las operaciones que se le realizan, sin dejar de lado la base de datos, lo anterior es un problema recurrente pues en muchos casos los temas de documentación de base de datos se consideran irrelevantes.

Nuestro aplicativo tiene como plus llevar la traza de los cambios que se realizan a nivel de base de datos pensando en que cada versión del aplicativo sea coherente con su respectiva capa de datos.

Cuando se hace referencia a la trazabilidad en un sistema de información se debe considerar sin lugar a dudas el versionamiento del mismo, esta tarea implica todos los componentes del software y si se piensa desde el punto de vista del mercado este punto puede ser considerado como un objetivo estratégico apuntando hacia la competitividad.

En torno a los sistemas de control de versiones, se evidencia que, aunque la noción que tienen las empresas de estos sistemas es la correcta, no se utilizan como deberían. El sistema que planteamos está diseñado para acompañar el ciclo de vida del sistema informático, organiza el trabajo que realiza el desarrollador e implementador y además pone en marcha las buenas prácticas en construcción de software.

Por otra parte, validamos como nuestra solución se convierte en una herramienta útil para el área comercial de una empresa de desarrollo, al tener evidencia concreta de que versión tiene implementado el cliente se puede determinar que productos, mejoras o actualizaciones necesita y de forma globalmente como atacar los segmentos de clientes.

En conclusión, la realización del trabajo de grado evidencia como la práctica de la Ingeniería de Software describe la forma correcta de construir sistemas de información, es absolutamente necesario dejar de lado la construcción artesanal y empezar a pensar en ingeniería, solo de esta forma se entrega un aporte importante a la disciplina.

4.1. Resultados y discusión

los resultados generales que arroja la investigación son:

- El entendimiento de cómo se debe construir un sistema informático.
- La utilización de una metodología de desarrollo, para dejar de lado la improvisación al momento de entrar a un proyecto de construcción de software.
- La utilización del modelado UML para concebir una herramienta informática.

Los resultados en artefactos se pueden catalogar así:

- Un documento que explica cómo se concibió el sistema de información APPEVOLUTION.
- Un prototipo funcional del sistema de información APPEVOLUTION.

La discusión entonces se debe enfocar en como las empresas que construyen software llevan sus procesos de documentación, en esta parte podemos decir que es absolutamente necesario el control del código y del versionamiento, y es en este punto donde solucionamos la problemática implementando nuestro sistema de información APPEVOLUTION.

4.2. Verificación, contraste y evaluación de los objetivos

Con el desarrollo de este trabajo de grado hemos cumplido con el objetivo principal de la investigación y por ende con los objetivos específicos, esto lo podemos evidenciar así:

- Para dar por cumplido el objetivo principal hemos desarrollado un prototipo funcional que es capaz de llevar la traza del versionamiento de un producto base de desarrollo, permitiendo a las empresas desarrolladoras complementar y soportar la documentación de sus aplicativos. De igual forma les permite evidenciar cómo evolucionan sus productos logrando un control efectivo del ciclo de vida del sistema que producen.
- El aplicativo APPEVOLUTION permite además llevar un control de las estructuras de datos, los scripts de cabios garantizan coherencia entre la versión del aplicativo y su respectiva base de datos.
- Hemos diseñado un aplicativo cuyo fuerte está en el seguimiento de un sistema de información con puntos clave como scripts de cambios al hablar de bases de datos, actualizaciones, mejoras, nuevas funciones y errores. Lo anterior hace sumamente útil nuestra herramienta y la sitúa como un apoyo a los procesos de desarrollo de software.

4.3. Síntesis del modelo propuesto

Para el desarrollo del proyecto utilizamos:

- Archimate, para diseñar las vistas organizacionales donde se posiciona nuestra solución. Además, pretende evidenciar que componentes conforman el sistema.
- Un modelo de proceso de software, en este caso tomamos ciertas características del Proceso Unificado para la construcción del aplicativo. Por cada una de las fases de esta metodología se desarrolló un capitulo.

- Ingeniería de requerimientos, para ver claramente que solución diseñar y como atacar la problemática que surge al momento de documentar software.
- Diagramas UML, propuesto para el diseño de casos de uso, en estos se puede apreciar que funcionalidades clave tiene nuestro sistema informático.
- Prototipos, para mostrar que funcionalidades integra nuestra herramienta.
- Por ultimo proyectamos un manual de usuario de una parte del sistema que pretende dar a entender como interactuar con el aplicativo APPEVOLUTION.

4.4. PROSPECTIVA DEL TRABAJO DE GRADO

4.4.1. Líneas de investigación futuras

- Documentación completa de un sistema de información, haciendo referencia al ciclo de vida del software.
- Control de aplicaciones bajo distintos ambientes.
- Paralelo entre lo que estamos controlando con el aplicativo y la documentación exigida por los distintos modelos de desarrollo.

4.4.2. Trabajos de Investigación futuros

- Diseñar el aplicativo para entornos WEB.
- App móvil del aplicativo, propuesto por temas de comodidad del usuario.
- Desarrollo de herramientas complementarias.

Bibliografía

[Borrell, 2016] Borrell, G. (2016). El control de versiones.

[Bruegge and Dutoit, 2002] Bruegge, B. and Dutoit, A. (2002). Ingeniería de software orientado a objetos.

[Castro,] Castro, S. J. B. Notas de clase ingeniería de software ii.

[Elmasri and Navathe, 2007] Elmasri, R. and Navathe, S. B. (2007). Fundamentos de sistemas de bases de datos.

[ERIKSSON and PENKER, 2016] ERIKSSON, H.-E. and PENKER, M. (2016). UML Toolkit.

[Garzás, 2016] Garzás, J. (2016). Estrategias para versionar las bases de datos relacionales.

[GIT, 2016a] GIT (2016a). Empezando - acerca del control de versiones.

[GIT, 2016b] GIT (2016b). Empezando - fundamentos de git.

[Group, 2013] Group, T. O. (2013). Togaf versión 9.1 guía de bolsillo.

[Hipertextual, 2014] Hipertextual (2014). Qué es un sistema de control de versiones y por qué es tan importante.

[Jacobson, 2000] Jacobson, I. (2000). *El Proceso Unificado de Desarrollo de Software*.

[Pressman, 2010] Pressman, R. S. (2010). Ingenieria de software un enfoque practico.

[Pressman and Troya, 1988] Pressman, R. S. and Troya, J. M. (1988). Ingeniería del software.

[Pérez, 2014] Pérez, D. M. T. (2014). *Administración Estratégica de la función informática*.

[Schmuller, 2000] Schmuller, J. (2000). Aprendiendo uml en 24 horas.

[Sommerville, 2005] Sommerville, I. (2005). *Ingeniería del Software Septima Edición*.

[Stevens and Pooley, 2007] Stevens, P. and Pooley, R. (2007). Utilización de uml en ingeniería del software con objetos y componentes.

[Studylib, 2016] Studylib (2016). El Lenguaje unificado de modelado.

[TERASWAP, 2016] TERASWAP (2016). Liquibase: Gestionando los cambios de la bbdd.