UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DISEÑO E IMPLEMENTACIÓN DE UN PROCESO DE GESTIÓN DE LA CONFIGURACIÓN PARA UN BANCO

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN TECNOLOGÍAS DE LA INFORMACIÓN

ROBERTO FELIPE RODRÍGUEZ PINO

PROFESOR GUÍA:
SERGIO OCHOA DELORENZI

MIEMBROS DE LA COMISION:
ALEXANDRE BERGEL
JOCELYN SIMMONDS WAGEMANN
JUAN VIDAL ROJAS

SANTIAGO DE CHILE
2021

Resumen

La organización en la que se desarrolló la presente tesis es un holding, con presencia en varios países de América Latina y que abarca distintas industrias (*retail* y banca, entre otros). El área de "banca" de esta organización opera involucrando el uso de sistemas grandes y complejos, que son adaptados de manera periódica o, en su defecto, extendidos para responder a las diferentes necesidades de las filiales de la empresa en cada país. En esencia, la estrategia es diseñar una solución general, y luego implementarla de manera local en cada país.

Por otro lado, la gestión del código fuente asociado a las diferentes implementaciones cumple un rol fundamental, puesto que impacta los tiempos de respuesta a las solicitudes de las filiales, así como la calidad y la previsibilidad del resultado de una actualización. Actualmente, los programas fuentes de los sistemas de banca se vinculan al servidor en el que se encuentran instalados, por lo tanto, no existen registros de los cambios, ni relación entre ellos. La falta de registros de los cambios que se realizan sobre el sistema constituye un problema operacional, puesto que no existe certeza de las versiones con las que se está trabajando, generando sobretrabajo y ocultando errores difíciles de detectar.

Para mejorar la gestión de los componentes de los sistemas de la organización, específicamente para el ámbito de "banca", este trabajo de tesis buscó disociar los ambientes donde se ejecuta el software, estableciendo una relación entre los distintos ambientes o versiones de un software, facilitando la instalación o despliegue de las diversas versiones del mismo, permitiendo el desarrollo en paralelo de más de un componente.

Con el fin de lograr lo antes mencionado, esta tesis propone un proceso de gestión de la configuración que busca establecer relaciones entre los distintos ambientes, para instaurar así versiones de un sistema (y de sus componentes) independiente del ambiente en el que se encuentren instalados. Además, se busca reducir el trabajo manual y la incertidumbre durante el proceso de actualización de componentes de un sistema.

El proceso propuesto se aplicó a seis proyectos de software. En cada uno de ellos se establecieron líneas bases, y se identificaron los componentes modificados, la cantidad de actualizaciones realizadas y las puestas en producción completadas. Esto permitió contrastar los errores cometidos y analizar cómo fueron abordados.

Los resultados obtenidos estuvieron dentro de lo esperado. Aunque ocurrieron algunas situaciones que no estaban contempladas, éstas pudieron ser corregidas prontamente con una salida alternativa. Las contribuciones de esta solución apuntan principalmente a facilitar la gestión de los componentes y poder ofrecer certezas respecto de las versiones o funcionalidades disponibles, que en un futuro deberían ser desplegadas en sistemas o servicios dinámicos. El trabajo a futuro considera el uso de este proceso en otros sistemas de la organización.

A mis padres y Lucas.

Agradecimientos

Rodrigo, Felipe, Eduardo y Juan Pablo, muchas gracias por compartir toda su experiencia.
Daniela, Susana y Juan Carlos, por todo el apoyo en los momentos difíciles.
Daniela, por haberme acompañado en este camino.
Al profesor Sergio Ochoa por todas las revisiones y sugerencias realizadas.

Tabla de Contenido

Índice de Tablas

Índice de Figuras

1 Introducción

La organización en la que se aplica este trabajo es un holding que tiene participación en distintas industrias (*retail*, mejoras del hogar, banca y otras), y está compuesta por 12.422 empleados y 492 sucursales, presentes en 5 países de América Latina relacionadas. En particular, para las actividades de negocio financiero asociado a la banca la empresa maneja 6,5 millones de clientes, y desarrolla permanentemente soluciones de software para dicho negocio, fomentando el modelo de atención no presencial.

De acuerdo con la localización, cada negocio requiere tener su propia infraestructura en lo que a servidores se refiere, lo mismo ocurre con las plataformas que apoyan la operación, puesto que existen diferentes normativas y regulaciones específicas en cada país. Estas plataformas acumulan todas las personalizaciones requeridas por cada negocio-país, por lo que se hace necesario contar con una gestión estricta sobre los componentes de software de las mismas.

La administración y gestión, tanto del software como de la infraestructura, se encuentra centralizada en varios equipos en Chile, que atienden las distintas necesidades asociadas a la localización del negocio y a la operación misma de cada plataforma de software. Una de las tareas más críticas y frecuentes es la gestión de componentes de software, tanto para la actualización de reglas de negocio paramétricas, como para el apoyo en la ejecución de proyectos evolutivos o correctivos.

1.1 Problema a Abordar

La inexistencia de registros de cambios realizados sobre un sistema puede traer problemas desde el punto de vista operacional, provocando que se desconozca cuál es la funcionalidad que se está ejecutando actualmente en algún determinado ambiente. En etapas de desarrollo, esto impide tener certeza acerca de si se está trabajando con la última versión de los componentes disponibles para el sistema.

En la organización, algunos equipos encargados de la mantención y operación del software manejan las distintas versiones de éste directamente en los entornos de ejecución, estableciendo una relación entre el servidor propiamente tal (donde se ejecuta el software) y la versión del software en sí, sin mantener un control de los cambios que se realizan. Tampoco existen líneas base respecto a lo que hay instalado en los múltiples ambientes, ni controles o alertas que pudieran gatillarse en la eventualidad de que dos equipos modificaran los mismos componentes de forma simultánea.

Actualmente, los equipos de mantenimiento reciben múltiples peticiones de los equipos de negocio, quienes solicitan instalar componentes de software en un determinado ambiente. Por ejemplo, un usuario de negocio puede pedir la instalación de un componente en el ambiente

de desarrollo. Luego, un desarrollador podría pedir la instalación de otros componentes en el ambiente de QA (*Quality Assurance*), para que finalmente un jefe de proyecto instale nuevas piezas de software en el ambiente de producción.

Siguiendo esa dinámica de trabajo, es sumamente frecuente que se produzcan inconsistencias respecto de los cambios aplicados en los distintos entornos (o ambientes) de ejecución. Tal como muestra la Figura 1, cada servidor representa un entorno de ejecución, que tiene distintos cambios aplicados (círculos de la figura), sin que exista relación alguna entre los distintos entornos, lo que, habitualmente, genera dos problemas importantes:

- En tiempo de desarrollo: Provoca incertidumbre respecto de la modificación del componente, es decir, si este corresponde a la última versión y si se comportará igual en todos los ambientes.

- En tiempo de operación: Causa dudas respecto de la correcta instalación de todos los componentes, de acuerdo a lo que se está desarrollando.

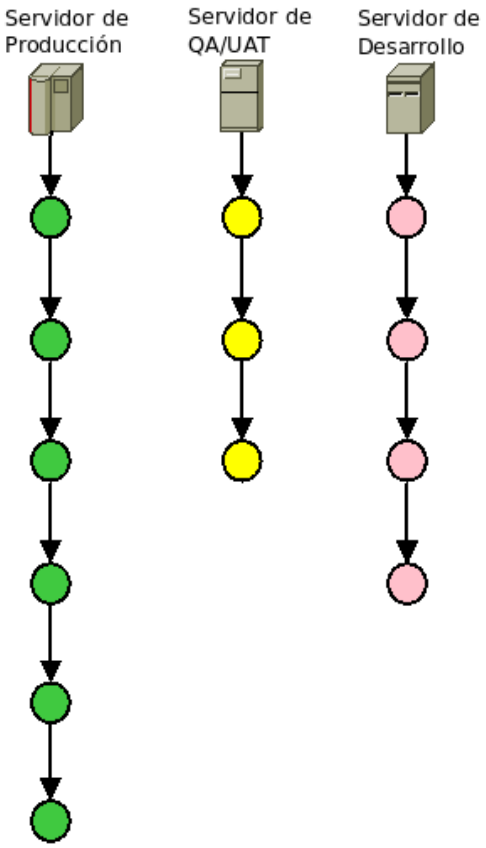

Figura 1: Relación entre cambios en los distintos servidores. Cada círculo representa una serie de componentes modificados en dicho entorno de ejecución.

Lo descrito anteriormente implica además otros problemas, como por ejemplo:

- Las certificaciones de calidad solamente se pueden hacer en un único servidor.

- Al tener más de un proyecto en marcha sobre el mismo sistema, resulta muy fácil sobrescribir erróneamente algún componente.

- Si se realizan correcciones sobre la versión de certificación, no existirían registros de los componentes que se han modificado y que, eventualmente, se deban re-certificar.

De acuerdo con lo anterior, es posible establecer que, debido a la forma en la que opera la unidad analizada:

- No existe trazabilidad del software: resulta imposible establecer una relación entre los distintos componentes de software que pasan de un ambiente a otro; por ejemplo, de desarrollo a QA, o de QA a producción. Esta falta de trazabilidad podría tener como consecuencia la instalación, en el ambiente de producción, de componentes que no han sido probados, los que, a su vez, podrían generar incompatibilidades con el software existente y provocarían un impacto en servicios productivos de procesamiento de datos.

- No existe posibilidad de tener más de un proyecto de desarrollo en paralelo: actualmente, en la organización, está en boga el concepto de desarrollo mediante el uso de metodologías ágiles, lo que podría requerir el desarrollo de varios proyectos en paralelo, sobre el mismo sistema. Eventualmente, estos tendrían la necesidad de interactuar entre sí en etapas muy tempranas, aunque no necesariamente terminarían juntos, por lo que resulta necesario contar con una correcta gestión de los componentes modificados o requeridos por cada uno de los proyectos.

- Los problemas fuera de los entornos (tanto en ambientes previos, como productivos) son irreproducibles: como las distintas versiones del software se encuentran en los servidores, existe un impedimento para detectar o reproducir bugs o errores de manera fácil.

- El software instalado en un determinado entorno no cuenta con su correspondiente registro de actualizaciones: el comportamiento del software varía de acuerdo al servidor en el que esté instalado, por lo que no existe un registro de los cambios que en él se realizan. De existir, dicho registro debiese incluir qué, quién y cuándo se actualizó el software; dado que se realizan actualizaciones manuales a los servidores, es muy común que se actualice solo a nivel productivo, por lo que los otros ambientes quedan

desactualizados y, de esta manera, algunos componentes quedan desfasados frente a producción (o al pasar de un ambiente a otro), generando incertidumbre respecto a cuál es el componente correcto o más actualizado.

- No existe una biblioteca con las compilaciones del software: la existencia de esta no solo facilitaría el acceso al registro, sino que también posibilitaría la instalación de una versión determinada del software, simplificando los pasos a producción. Si se requiere volver atrás, el proceso es el mismo, pero recurriendo a la versión anterior. Actualmente se solicita la búsqueda manual y la restauración de los componentes anteriores, e incluso se recurre a la búsqueda en respaldos.

- Operación manual para el traspaso de versiones: la mayor parte del tiempo, el traspaso de versiones de un ambiente a otro implica la copia manual de archivos, dependiendo del tipo de cambio, por lo que la posibilidad de cometer errores siempre queda abierta.

1.2 Objetivos de la Tesis

El objetivo general de este trabajo de tesis fue definir un proceso de gestión de la configuración que permita el seguimiento de componentes de un sistema de software, durante el desarrollo y el mantenimiento del mismo. Los objetivos específicos que se desprenden del objetivo general son:

1. Definir un ciclo de vida para el software desplegado como parte de un sistema.
2. Desacoplar (independizar) la gestión de versiones del software de los servidores en donde estas se están ejecutando.
3. Automatizar las actividades de despliegue de aplicaciones en servidores productivos, de QA y de desarrollo, para no depender de acciones manuales de los administradores de los sistemas.

Logrando los objetivos se pretende eliminar los problemas asociados a la gestión de los componentes asociados a las manualidades que existen en relación a la instalación de los mismos. Con esto cumplido, se puede:

- Flexibilizar el uso de servidores (principalmente los de desarrollo y certificación).
- Analizar algún problema detectado en un ambiente productivo (donde los accesos siempre están restringidos).
- Desarrollar proyectos en paralelo.
- Formalización de la forma de trabajo, independiente de las personas que compongan el equipo.

En conclusión, la operación del software se verá robustecida desde el punto de vista de la gestión de los componentes, siendo un gran apoyo para el desarrollo de los proyectos eliminando uno de los factores de falla que se presentan habitualmente en la completitud de los mismos.

Este proceso se evaluó a través de una prueba piloto con el equipo encargado de manejar los sistemas de riesgo crediticio de la organización. Los resultados obtenidos fueron muy prometedores. En lo que resta del documento se presenta el trabajo realizado, así como la evaluación del proceso de gestión de la configuración propuesto y los resultados obtenidos.

1.3 Estructura del Documento

Este documento de tesis está compuesto de 6 capítulos. En el capítulo 2 se describe la situación actual de la organización y se presenta una revisión general de cómo distintas empresas y rubros abordan la gestión de componentes del software que desarrollan. En el capítulo 3 se presenta el proceso de gestión de la configuración propuesto; en el 4 describe la evaluación de las herramientas para llevar a cabo la implementación, mientras que en el capítulo 5 presenta la evaluación del proceso propuesto. Finalmente, el capítulo 6 da a conocer las conclusiones de esta tesis y el trabajo a futuro.

2 Marco Teórico

A continuación se describe la situación de la organización antes de este proyecto, considerando la herramienta utilizada para abordar el problema planteado. Luego, en base a una investigación hecha por el tesista, se indica cómo (con qué herramientas) otras organizaciones han tratado de resolver dicho problema. Finalmente, se describen y comparan brevemente las herramientas de apoyo mencionadas en las secciones previas.

2.1 Uso de herramientas de control de versiones en la organización

En la organización donde se enmarca esta tesis existen diversas herramientas con las que se ha intentado resolver la problemática de la gestión de la configuración de componentes. La principal (y actualmente en uso) es SVN. Al día de hoy, esta herramienta solo ha logrado facilitar los despliegues en los múltiples ambientes. Dado que la organización requiere desplegar los sistemas en los distintos ambientes, a partir del uso de esta herramienta aparecen múltiples limitaciones, por ejemplo:

- Existen archivos binarios en repositorios de código, sin tener trazas que indiquen desde donde han sido generados.
- Existen múltiples repositorios para múltiples ambientes, los cuales no están enlazados de una manera sistémica u ordenada.
- Pérdidas de funcionalidades por errores al reemplazar un archivo binario por otro.

El uso de SVN en la organización se ha desvirtuado, llegando incluso a cambiar el significado de los términos asignados por la herramienta. La tabla 1 muestra los significados de los términos que maneja la herramienta según la empresa desarrolladora de SVN en comparación con los asignados por la organización que se describe en este trabajo de tesis.

Por otra parte, en los últimos meses se han ido incorporando nuevas herramientas a la pila disponible, de la mano de una forma de operar más ágil por parte de algunos equipos de trabajo. Desde el punto de vista operativo, el uso de estas herramientas apunta principalmente a la autogestión e independencia del equipo al momento de instalar software en ambientes de desarrollo, QA y productivos. Esto entrega la libertad para que cada equipo defina cómo prefiere trabajar.

Dentro del listado de herramientas, se han habilitado gestores de repositorio y artefactos para automatizar algunas actividades, además de incorporar nuevas aplicaciones que permiten desplegar contenedores con el software ya instalado y configurado.

Tabla 1: Comparación de términos originales en SVN contra el uso dado en la organización

Término en SVN	Significado original	Significado en la organización
Branch	Línea de desarrollo que comparte origen con otra línea, pero siendo ambas independientes.	Es un directorio dentro del repositorio.
Tag	Copia de una de las líneas de desarrollo asociado a un hito.	Es el número de revisión.
SVN	Herramienta de versionado.	Herramienta que se instala en distintos ambientes.
Repositorio	Lugar donde se deja una configuración relacionada con SVN para un proyecto de software.	Un directorio donde cada subdirectorio, a través de permisos, se convierte en un "repositorio" para cada proyecto de software.
Revisión	Identificador único de una transacción en el repositorio.	No es usado.

Debido a que algunas de estas herramientas han sido contratadas como servicio, y al alto nivel de adopción que han tenido por parte de algunos equipos, el costo de mantenerlas se ha vuelto importante. Por lo tanto, la organización ha cambiado la estrategia de disponibilización de las herramientas; en ese sentido, pasó de usarlas como servicio, a tenerlas *on-premise*. Esto último ha llevado a que se deban cambiar algunos productos, como por ejemplo, el gestor de repositorios (de Bitbucket a GitLab).

Para la organización es claro que debe poder adaptarse a los nuevos escenarios, sin dejar de lado la metodología de trabajo, y haciendo que ésta sea consistente en todo momento.

2.2 Experiencia en otras organizaciones

El autor de esta tesis realizó una investigación complementaria para conocer cómo otras organizaciones habían abordado la problemática planteada, y sacar lecciones aprendidas y potenciales soluciones a dicho problema. Para ello se revisó la realidad de cinco empresas, las cuales cumplían con los siguientes criterios: 1) realizaban el desarrollo y mantención de sistemas grandes que estaban en producción, 2) sufrían los mismos problemas (o similares) a los aquí planteado, y 3) estaban disponibles para compartir la información y lecciones aprendidas con este tesista. Este análisis permitió identificar que, independientemente del negocio que aborden estas empresas, la problemática de cómo gestionar los cambios y llevarlos a un entregable final es similar en todas ellas.

En las siguientes secciones se explica brevemente la realidad recabada en cinco organizaciones. La información fue provista por ingenieros de software que trabajaron en distintos roles (jefe de proyecto, desarrollador, arquitecto, asegurador de calidad) dentro de las organizaciones.

2.2.1 Organización 1

La primera organización analizada fue una entidad bancaria la cual, para solucionar el problema planteado, adaptó el ciclo de vida de sus componentes para que se ajuste al funcionamiento de la herramienta de apoyo que ellos utilizaban. En esta adaptación se involucraron a todos los equipos de TI considerando, jefes de proyecto, analistas, desarrolladores y equipos de control de calidad que aproximadamente son 170 personas. Estos números no consideran a los equipos de proveedores externos, los cuales también tenían que involucrarse en el proceso, y que podrían fácilmente aumentar la cifra anterior en 100 personas más.

Para ilustrar la forma de gestión de versiones de los componentes se tomó como guía la forma de operar de la herramienta CA Harvest. Esta herramienta, es del tipo *world class* y teóricamente reúne las mejores prácticas de la industria en cuanto a la gestión de componentes de software.

En CA Harvest cada uno de los componentes tiene su propio identificador asociado a una versión. Si bien esto puede permitir el desarrollo en paralelo, y en parte evitar conflictos por reemplazo de componentes, la gestión de los sistemas se vuelve compleja ya que las instalaciones de los mismos se hacen a partir de lo especificado en documentos. Esto hace que dicha labor se traduzca en tareas sumamente engorrosas cuando se modifican muchos componentes.

Diariamente la persona encargada de la gestión de componentes a nivel de desarrollo, compila e instala lo que los desarrolladores solicitan, pero estos cambios no necesariamente pasan por un repositorio de versiones. Esto desencadena que a veces se estuviera trabajando con componentes desactualizados, situación que era alertada en el momento en que el componente era instalado en un ambiente de QA o de certificación, con la consiguiente pérdida de tiempo.

Finalmente, después de haber terminado el proceso de QA o de certificación, el destino final del componente era un ambiente productivo, el cual es compilado a partir de lo obtenido del repositorio. Este nuevo software compilado es directamente instalado y desplegado en los ambientes productivos. En caso de que se presentaran problemas, el componente podía ser modificado manualmente o reemplazado por la versión anterior. Para el primer caso, quedaba pendiente la regularización del componente en el repositorio, lo que no siempre ocurría, produciendo inconsistencias entre los distintos ambientes.

Adicionalmente existen tablas de parámetros que son modificadas directamente en los servidores, y que no están sometidas a un control exhaustivo en el repositorio. Esto puede

producir comportamientos inesperados, dadas las diferencias de los parámetros entre los ambientes, e incluso entre servidores de un mismo ambiente.

2.2.2 Organización 2

La segunda organización analizada fue también era una entidad bancaria, la cual tenía dos formas de trabajar: una para aplicaciones nuevas y otra para las aplicaciones legadas. En estas últimas se han definido tres áreas para gestionar el flujo de trabajo que tienen los múltiples equipos de desarrollo. En total, este flujo coordina aproximadamente 150 personas, entre desarrolladores, analistas, jefes de proyectos, testers e ingenieros de calidad.

La primera área es la de control de versiones, que está encargada de configurar los *pipelines*, *streamlines* de trabajo, y dar las directrices generales respecto a cómo se deben usar con las herramientas Dimensions CM y Serena Dimensions (ambas de Microfocus)[1] que apoyan esas labores.

La segunda área se encarga de llevar el control y la gestión del proyecto, asegurando que el desarrollo se vaya completando en tiempo y forma, y facilitando la comunicación entre las áreas solicitantes y el equipo de desarrollo.

La tercera área corresponde a los equipos de aseguramiento de calidad, que además tienen la tarea de ambientar los artefactos que el equipo de desarrollo va generando, tanto en ambientes de desarrollo como de *testing*.

El proceso que sigue un equipo de desarrollo corresponde a realizar todas las modificaciones necesarias de acuerdo al requerimiento, y que éstas queden reflejadas en un streamline del repositorio de código, para que un pipeline ad hoc genere un artefacto. Este artefacto está asociado a un número correlativo que permite trazar su origen en la construcción.

Por otro lado, la ambientación de dicho artefacto en un ambiente de desarrollo, es coordinada vía correo electrónico con el equipo de QA, para que estos hagan un despliegue (manual o semiautomático) en los distintos servidores de acuerdo a la naturaleza del software modificado. Cuando el equipo de desarrollo valida con el equipo de control de cambios que el requerimiento está satisfecho, que se han seguido los lineamientos de desarrollo, y que el proyecto está cumpliendo con las fechas planificadas, entonces se formaliza una entrega para el equipo de QA. A partir de ahí, este equipo realizará pruebas funcionales, de calidad de código y de cobertura de código. Para las dos últimas actividades, existen herramientas semiautomáticas que realizan la revisión.

1 DimensionsCM y Serena Dimensions, son herramientas que permiten gestionar los cambios sobre el código fuente y programar tareas automatizadas, como por ejemplo la ejecución de pruebas unitarias sobre el software desarrollado.

El despliegue o instalación en los servidores de QA se hace a partir del pipeline de despliegue. Cuando el software alcanza un cierto nivel de madurez, esta área asegura que cumple con los estándares requeridos por la organización a través de un informe. Luego, el despliegue y/o instalación en servidores productivos es revisado y aprobado por un comité, el cual revisa la documentación generada en el proceso (*streamline*, *pipelines*, informe de QA, identificador de artefactos generados, pauta de instalación y procedimiento de vuelta atrás o *rollback*).

Finalmente, toda información es consolidada manualmente en un sistema construido especialmente para la organización, que resume en información todo lo necesario para que un operador pueda seguir con la gestión en ambientes productivos. El procedimiento de vuelta atrás es volviendo al artefacto a su versión anterior, pero que debe ser explicitado para que el operador no tenga capacidad para confundirse.

No existe una forma explícita de determinar qué artefacto está instalado en cada ambiente, salvo mirando la documentación que tiene el sistema ad hoc, por ejemplo, la información que pueda proveer el equipo de QA, y los registros propios que se puedan tener respecto a los ambientes. Esto a veces complica el análisis y la identificación de los problemas (o de bugs) que pueda tener el software.

Por otra parte, tampoco es posible realizar desarrollos en paralelo, ya que la herramienta que gestiona el código genera un bloqueo de los componentes para que no se pueda modificar. Salvo excepciones, cuando hay problemas urgentes que corregir en ambientes productivos, y mediante una coordinación con el equipo de control de versiones, se puede forzar un cambio, pero con la obvia desactualización que provocará sobre el equipo que está haciendo la actualización.

Esta solución tiene varias limitaciones, por lo cual este flujo de trabajo ha sido declarado como obsoleto para la organización. Debido a eso, dicho flujo es utilizado solo en software desarrollado hace más de 5 años. Algunas de las principales limitaciones de esta solución son:

- La herramienta sólo tiene clientes disponibles para la plataforma Windows. Hoy en la organización los equipos de desarrollo utilizan principalmente Linux y MacOS.
- Los pipelines son extremadamente deficientes respecto a la funcionalidad propuesta por la herramienta, y no generan valor por sí mismos.
- Si bien en un principio, el objetivo de la herramienta de pipelines era tener centralizado la gestión y despliegue de artefactos, la heterogeneidad de los sistemas hacía que se requieran distintas funcionalidades, las cuales no eran provistas por la herramienta, o bien su implementación era deficiente, teniendo que recurrir al uso de otras herramientas.
- Las herramientas de gestión de proyecto, registro de incidencias y monitoreo del nivel de madurez del software son casi imposibles de integrar con las herramientas para

versionar y desplegar el software. Esto produce una sobrecarga respecto al uso de sistemas por parte de los equipos de desarrollo y gestión del software.

2.2.3 Organización 3

La tercera organización era una compañía de software cuyo núcleo de negocio era la creación, mantención y soporte de múltiples sistemas, los cuales tienen como destino final tanto clientes como repositorios públicos de software, por ende, no mantienen los productos en función de las necesidades de cada uno de los clientes, sino que el producto de software base es distribuido directamente a cada cliente. Por lo tanto, cuando hay correcciones a partir de incidencias, o hay nuevas funcionalidades que incluir, éstas son puestas a disposición de todos.

Dentro de la organización existen aproximadamente 20 proyectos, los cuales son mantenidos directamente en la organización, pero existen múltiples contribuciones a distintos proyectos de software libre. Cada equipo está principalmente conformado por equipos pequeños, compuestos por 3 a 5 personas, además hay equipos grandes que llegan a un máximo de 15 integrantes. Adicionalmente, existe un equipo de QA compuesto por aproximadamente 8 personas que se encargan de realizar pruebas sobre todos los proyectos.

En términos generales, todos los equipos llevan a cabo las mismas actividades, las cuales se indican a continuación:

- Ejecución de pruebas unitarias: Las funcionalidades nuevas a incorporar a un producto de software deben tener una gran cobertura usando pruebas unitarias, y aprobar las pruebas respecto a la calidad del código (a través de automatizaciones).
- Liberación de versión de desarrollo: Cuando el equipo de desarrollo considera que se ha alcanzado un nivel de madurez aceptable, se hace un release para que el equipo de QA lo libere a partir de un mínimo de aprobaciones por parte del mismo equipo.
- Ejecución pruebas de integración: El equipo de QA ejecuta pruebas de integración sobre todo el ecosistema, incluyendo el software completo que la compañía pública.
- Liberación y empaquetado versión final: Al pasar las pruebas anteriores el software es identificado a través de una numeración, y es publicado en los repositorios de la compañía para que quede disponible para los clientes que lo requieran o necesiten actualizar.

Todas estas actividades no están formalizadas en un proceso escrito, por lo tanto, el proceso informal utilizado se va adaptando a las nuevas prácticas que cada equipo va incorporando. Adicionalmente, no hay herramientas establecidas, por lo que cada equipo tiene la libertad de escoger su propio conjunto de acuerdo a las necesidades propias del software desarrollado. Las combinaciones de herramientas más usadas en esta organización son las siguientes:

Tabla 2: Herramientas utilizadas por la Organización 3

Gestor de repositorios	Herramienta para versionar	Herramienta para automatizar tareas
GitHub	Git	Travis CI
GitHub	Git	Circle CI
Launchpad	Git	Jenkins

Uno de los criterios para escoger las herramientas es el nivel de automatización requerido, y la rapidez con que se quiera efectuar las pruebas unitarias. A veces las pruebas pueden requerir mayor poder de cómputo, y la herramienta a usar debe escalar adecuadamente, pudiendo ofrecer la flexibilidad de agregar/quitar nodos para poder realizar dichas pruebas.

Respecto a la estrategia para identificar las versiones, si bien no existe una formalidad de manera explícita, la más común es utilizar una nomenclatura que permite identificar cambios que rompen la compatibilidad con versiones anteriores, funcionalidades nuevas y correcciones. Esta nomenclatura es representada a través de números enteros en **major.minor.fix**, donde:

- Major indica el identificador principal de la versión. Una versión mayor puede romper compatibilidades con una menor.
- Minor indica que agrega nuevas funcionalidades dentro del software, y éstas son compatibles con las versiones minor anteriores.
- Fix indica que agrega correcciones sobre la versión minor. Una versión minor superior incluye el fix de una versión minor inferior.

Además de desarrollar software de propósito general, la compañía ofrece servicios de soporte y mantención de servidores. En situaciones muy puntuales se han realizado acciones correctivas del software directamente en entornos productivos, donde lo primordial es restablecer los servicios interrumpidos. Sin embargo, estos cambios son regularizados posteriormente, agregando las correcciones al sistema afectado, y son sometidos a todo el flujo descrito anteriormente.

2.2.4 Organización 4

La cuarta organización se dedicaba al desarrollo, adaptación y extensión de un software que ellos desarrollan, y que presta servicios a distintas empresas generadoras de electricidad del país. En cada nueva funcionalidad que agregan a su software, trabajan hasta tres personas por proyecto. La tabla 3 muestra las herramientas que utiliza la organización para apoyar la gestión de sus programas fuentes.

Tabla 3: Herramientas utilizadas por Organización 4

Gestor de repositorios	Herramienta para versionar	Herramienta para automatizar tareas
GitHub	Git	Codeship

Apoyándose en las funcionalidades que ofrece la herramienta de versionamiento, existen dos ramas principales: una maestra y una de desarrollo. Cada nueva funcionalidad que se agregue a un producto de software, debe realizarse dentro de una rama que nazca desde la rama de desarrollo. Cuando dicha funcionalidad se encuentra terminada, debe actualizarse contra la misma rama de desarrollo, para luego proceder a mezclarse a través de una funcionalidad llamada *pull request*. En el momento de que esta mezcla se encuentra aprobada y aceptada, en la misma rama de desarrollo es marcada como candidata a ser publicada, lo cual gatilla una serie de revisiones (pruebas unitarias, revisión de la calidad de código, nivel de cobertura).

Si estas pruebas salen exitosas, es sometida a una revisión por parte del equipo de QA. Si el proceso de revisión es terminado con éxito, se procede a mezclar el código en la rama maestra, lo que ejecuta los procesos de empaquetado y construcción de imágenes para los contenedores.

En caso de que existan problemas o errores detectados en ambientes productivos, estos nunca son resueltos directamente en los servidores. La forma de agregar estas correcciones es a través de un *hotfix* (alta prioridad) o un *fix* (baja prioridad). Ambas correcciones se realizan a través de ramas que nacen a partir de la rama maestra, y siempre deben mezclarse tanto en la rama maestra como en la de desarrollo.

2.2.5 Organización 5

La quinta organización es un laboratorio de productos farmacéuticos, el cual desarrolla y mantiene un software destinado a apoyar actividades de gestión y control de la producción, registrando horas de trabajo, cantidad de materiales y componentes químicos utilizados en la producción de medicamentos. Dicho software brinda información sumamente útil para las áreas de negocio y logística de la compañía.

El equipo de trabajo encargado del mantenimiento y extensión de este software está conformado por 12 personas, considerando desarrolladores, un jefe de proyecto y analistas. Este equipo no cuenta con un área de aseguramiento de calidad o con *testers*, por ende, todas las pruebas eran realizadas por los desarrolladores. Particularmente, el jefe de proyecto era el encargado de realizar pruebas para el paso a producción en el ambiente general.

La tecnología usada para el desarrollo de este software es .Net, por lo que es natural para el equipo utilizar la herramienta de control de versiones *Microsoft Visual Source Safe* (VSS).

Ésta viene integrada en la herramienta *Visual Studio* (también de Microsoft) utilizada por el equipo para desarrollar.

El software de la compañía tiene una arquitectura de cliente-servidor, por lo que es necesario distribuir cambios en ambos lados. Para esto, el equipo tiene a una persona encargada de compilar los cambios desde el IDE (*Integrated Development Environment*). Para el caso de un cambio en el cliente, esta persona actualiza metadatos del software compilado, y los registra en una base de datos (incrementar un número). Luego, debe coordinar con los usuarios para que vuelvan a ejecutar el software, donde un lanzador del programa se encarga de verificar si el número de construcción había sido incrementado, y obtener así los nuevos archivos binarios. Para el caso del servidor, el software se actualiza directamente.

Para el caso de una vuelta atrás, manualmente se deshacen los cambios en el repositorio de código, generando una nueva versión. Luego, la distribución del código se hace de la misma forma como si fuera un cambio nuevo. En esta organización, no existen registros de las pruebas, ni mayores automatizaciones del proceso de versionado ni de compilación.

Por otro lado, algunos cambios al software de la compañía requieren actualizar procedimientos almacenados y/o campos en alguna tabla de bases de datos. Estos cambios no eran registrados en ninguna parte, salvo en la BD. Ya que estos cambios no se hacen usando el IDE, estos quedan fuera del alcance de *Source Safe* y no son versionados.

Respecto a la herramienta *Source Safe*, uno de los problemas que existe es que solo se puede usar desde el IDE, lo que puede producir confusión en desarrolladores que tuvieran experiencias con otras herramientas de versionamiento. El otro problema que frecuentemente se presentaba es cuando dos desarrolladores modifican un mismo componente de manera simultánea. La lógica de *Source Safe* es tener un registro de los cambios de un archivo, guardándolos en una carpeta compartida (concepto propio de los entornos de trabajo Windows). En contraste, ambientar el software es sumamente sencillo para el desarrollador, ya que no necesita mayores esfuerzos que configurar el IDE y obtener desde el repositorio todo lo necesario para empezar a trabajar.

2.3 Herramientas de gestión de código

En la sección 2.2 se menciona una serie de herramientas asociadas al versionado de código y automatización de tareas. A continuación se describe brevemente cada una de ellas, destacando sus puntos a favor y en contra.

2.3.1 Subversion

Subversion, también conocido como SVN, es un sistema de control de versiones que permite manejar archivos y directorios, registrando todos los cambios que en ellos se hagan, pudiendo recuperar los datos de un archivo versionado previamente. A partir del detalle de los cambios registrados se puede determinar quién, qué y cuándo se modifica un archivo y/o directorio.

Para los archivos, SVN permite registrar cambios tanto en archivos de texto como binarios. Además, tiene conceptos asociados a ramas y etiquetas, los cuales se ven reflejados como copias completas del proyecto en disco. En la figura 2, se encuentra una representación general de los componentes que apoyan el proceso de gestión del software en SVN.

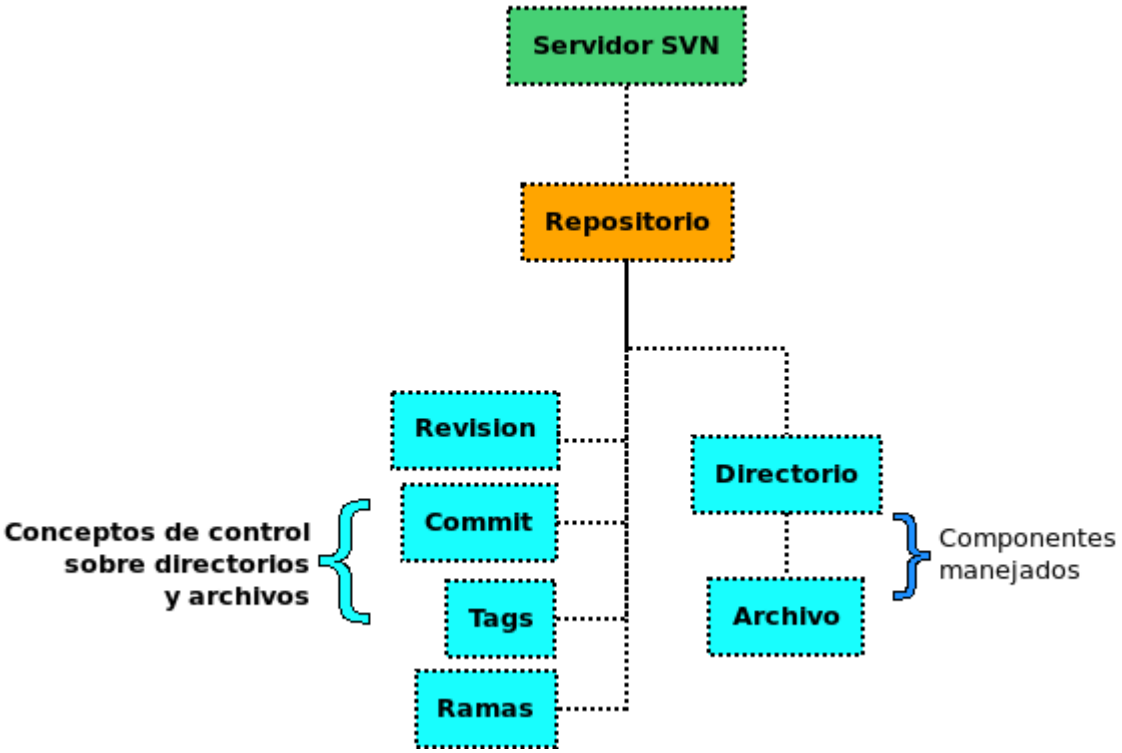

Figura 2: Estructura de apoyo al proceso de gestión de software en SVN.

Cada operación sobre el repositorio es una transacción atómica, donde puede ir uno o más cambios. Todos los cambios son unificados en dicho repositorio, permitiendo enfrentar problemas en el cliente, la red u otros. Cuando la transacción es aceptada, ésta queda asociada a un número de revisión, el cual es un correlativo dentro del mismo repositorio.

SVN funciona usando un repositorio centralizado, el cual puede ser accedido a través de la red usando un protocolo propio o vía HTTP. Esto permite utilizar muchas funcionalidades de

estos protocolos. El producto *Subversion* es distribuido usando el modelo de licencia de software libre.

2.3.2 CA Harvest

Originalmente creada en los años 70 para el Departamento de Defensa de Estados Unidos, la herramienta estaba orientada a gestionar los componentes del motor de un avión. Actualmente permite gestionar y mantener una o varias publicaciones de un software, permitiendo distintos flujos separándolos por *releases*. Esta funcionalidad permite tener proyectos de corta y larga duración en paralelo, pudiendo identificar qué modificaciones de un proyecto afectarán a otros. En esta herramienta los cambios son agrupados en paquetes.

CA Harvest gestiona múltiples proyectos a través de distintos repositorios. Estos repositorios contienen rutas (como si fuera una estructura de directorios) e ítems (como si fueran archivos), y tienen una versión numérica independiente. Por ejemplo, la versión 1 del ítem 1 ubicado en la ruta /ruta/para/llegar/Item1 no tiene relación con la versión 1 del ítem 2 ubicado en la ruta /ruta/para/llegar/Item2.

Cada vez que se requiere realizar alguna modificación, es necesario asociar los ítems a un package, el cual empezará a seguir el ciclo de vida que propone la herramienta. Secuencialmente el package debe pasar por varios estados (TEST → QA → Producción), en donde para cada paso, se requiere contar con múltiples revisiones que son generadas por los usuarios autorizados para hacerlo en cada etapa. La figura 3 muestra una representación general de la estructura usada por CA Harvest para gestionar los componentes de un producto de software.

Esta herramienta funciona usando un repositorio centralizado, accesible desde la red a través de un protocolo propietario. El producto CA Harvest es distribuido usando un modelo de licencias cerrado (licencia comercial).

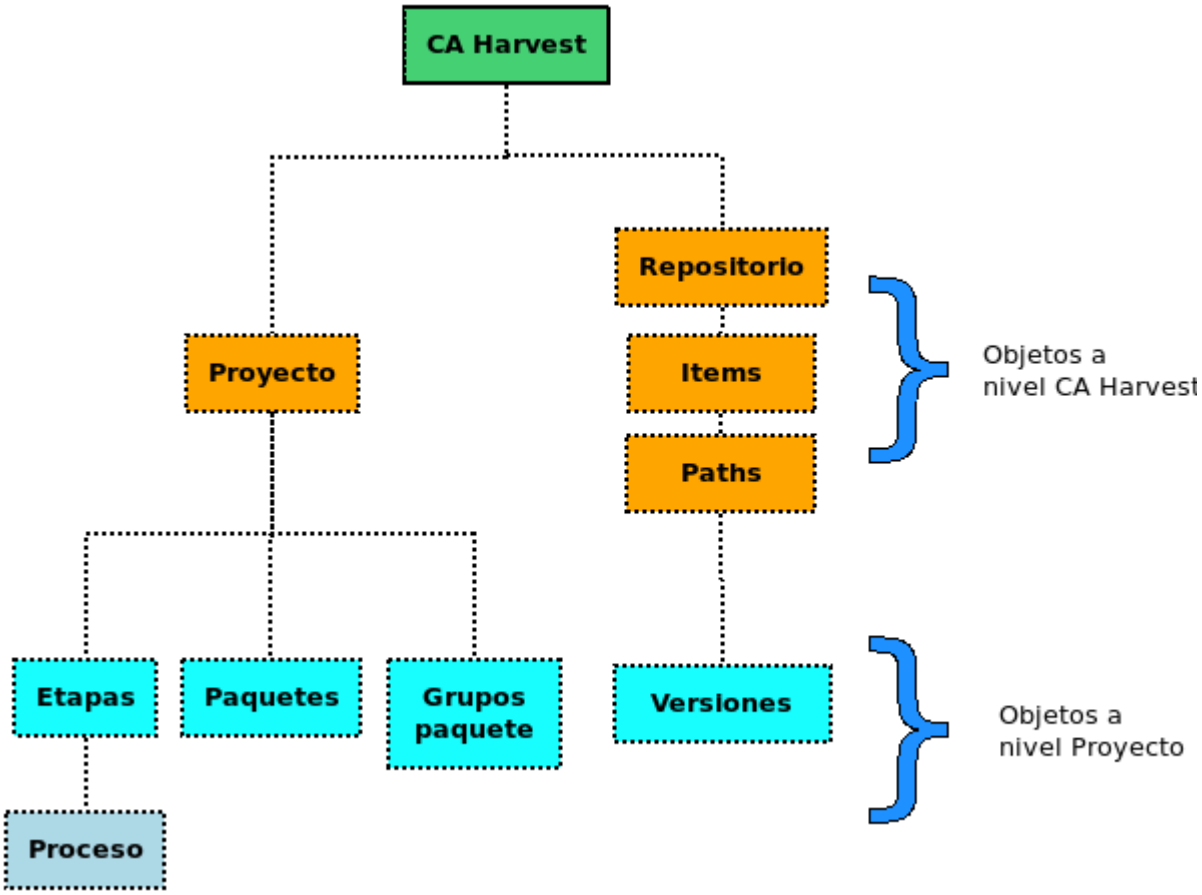

Figura 3: Estructura general de gestión de componentes, implementada por CA Harvest.

2.3.3 Dimensions CM

Dimensions CM es una herramienta que combina la gestión de los componentes del software, con el flujo de trabajo y el ciclo de vida del software. En términos generales este producto está compuesto por una BD básica (o *baseline*), la cual define cómo se gobernará el desarrollo de las aplicaciones. También se describe lo que es un producto, el cual corresponde a una agrupación de funcionalidades que puede tener el sistema (por ejemplo, un producto podría ser la evaluación de créditos).

Este producto estará desglosado en varias partes de diseño (por ejemplo, la interfaz de usuario, la lógica de negocio y los reportes). A su vez, las partes del diseño están compuestas por varios ítems y sus revisiones, donde un ítem podría ser cualquier componente. En general, estos componentes están asociados a un archivo (por ejemplo, código fuente, imágenes, especificación de requerimientos, manuales, etc.).

Finalmente, un *stream* es un contenedor que permite aislar el trabajo sobre una serie de partes del diseño e ítems, reservando dichos componentes a un equipo. Luego, cada vez que se consoliden los cambios en el *stream*, se crea una versión del mismo incluyendo los componentes modificados. La figura 4 muestra un ejemplo de cómo se maneja la gestión de componentes utilizando la herramienta Dimensions CM.

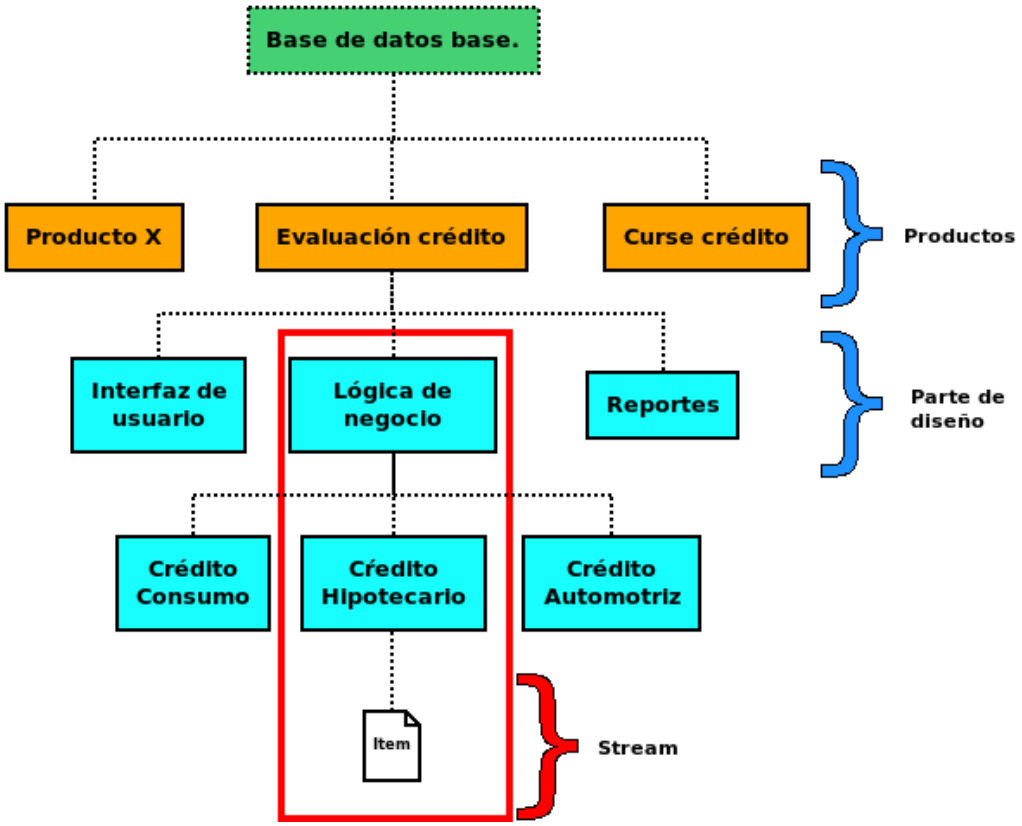

Figura 4: Ejemplo de estructura de la BD de gestión de componentes implementada en Dimensions CM.

Así mismo, la herramienta provee la capacidad de gestionar el flujo que deben seguir cada uno de los streams, cuando pasan desde una etapa de desarrollo a una de QA, y finalmente a una de producción, introduciendo roles para aprobar dichos pasos. Adicionalmente, la herramienta incluye una funcionalidad que permite comunicar a los distintos streams que puedan estar modificando componentes relacionados, generando así la posibilidad de realizar mezclas de los mismos a través de una solicitud.

Finalmente, Dimensions CM, maneja el concepto de línea base la cual consiste en "congelar" algún estado de todos los streams que estén finalizados, para generar un punto de referencia para la creación de nuevos desarrollos a partir de ellos. Al igual que CA Harvest, este producto funciona usando un repositorio centralizado, accesible desde la red a través de un protocolo

propietario. El software Dimensions CM es distribuido usando un modelo de licencia comercial.

2.3.4 Visual Source Safe

La herramienta Visual Source Safe estructura la gestión de componentes a través de una base de datos centralizada, la cual contiene una estructura que debe ser previamente configurada a través de un administrador de acuerdo a lo requerido por un equipo de trabajo. En esta base de datos, se registran los cambios y todas las versiones que puede tener uno o más archivos.

Estos archivos son agrupados en la base de datos en torno a un proyecto que pueden ser manejado visualmente en la herramienta para facilitar el acceso a ellos o a sus componentes. El control de los archivos se lleva a través de un versionado numérico automático sobre cada cambio que se registre en cada archivo.

Visual Source Safe introduce el concepto de hito, cada uno de los cuales es gestionado a través de *labels*. Con los hitos, hace un símil a las etapas que podría tener un proyecto en cascada o iterativo y sus fases. Cuando una de estas etapas termina, se establece el fin o inicio de la etapa a través de un *label*, y se define su correspondiente entrega a nivel de software. Este distintivo es posible de listar a través de la vista de proyectos, ofreciendo así un listado con todos los hitos que ha tenido el desarrollo. En la figura 5 se encuentra una representación general de lo anteriormente descrito.

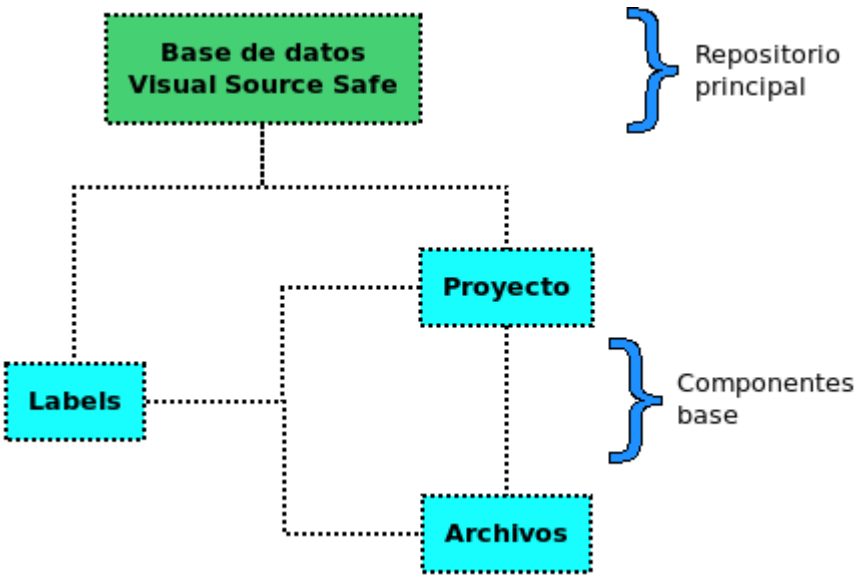

Figura 5: Estructura de apoyo a la gestión de software para Visual Source Safe.

La herramienta funciona como repositorio centralizado y utiliza el protocolo SMB (*Server Message Block*). Éste es el protocolo que usan los sistemas operativos Windows para compartir recursos a través de una red local. El producto Visual Source Safe es distribuido a través de un modelo de licencias cerradas, y es incluido e integrado dentro del paquete de desarrollo *Visual Studio* de Microsoft.

2.3.5 Git

Git es un sistema de control de versiones, que a diferencia de los revisados anteriormente, tiene una naturaleza distribuida y no depende de un repositorio centralizado para funcionar. Es decir, en el momento en que se obtiene una copia del repositorio, ésta tiene la capacidad de convertirse en nuevo repositorio completamente autónomo e independiente del original.

A diferencia de otras herramientas (como SVN) que manejan una lista de cambios en los archivos, Git lo hace a través de copias instantáneas de un sistema de archivos miniaturizados, guardando referencias de los archivos y teniendo en cuenta los archivos modificados. Esto permite realizar todas las operaciones localmente sin tener que usar un repositorio centralizado. Para el control de los archivos, Git maneja tres estados:

1. *Committed*: el cambio está confirmado y registrado en la base de datos local
2. *Modified*: existe algún cambio dentro de los archivos registrados previamente y
3. *Staged*: el cual indica los archivos que serán registrados (incluyendo sus cambios) en una próxima acción.

Esta herramienta permite manejar múltiples ramas y establecer *tags* asociados a cualquier estado *committed*. Las ramas, pueden ser mezcladas tanto para las locales, como para las remotas, pudiendo así incorporar el trabajo de otras personas. No hay una forma preestablecida respecto a cómo debe ser el flujo de trabajo, ofreciendo flexibilidad a cada equipo sobre cómo realizar esta gestión.

Tal como se mencionó anteriormente, Git funciona de manera distribuida y utiliza múltiples protocolos abiertos de comunicación otorgando alta flexibilidad respecto a cómo compartir los cambios. Estos son git://, file://, ssh:// y https://. Este producto es distribuido como software libre. En la figura 6 se encuentra una vista general de cómo Git estructura el soporte de datos para el manejo de versiones.

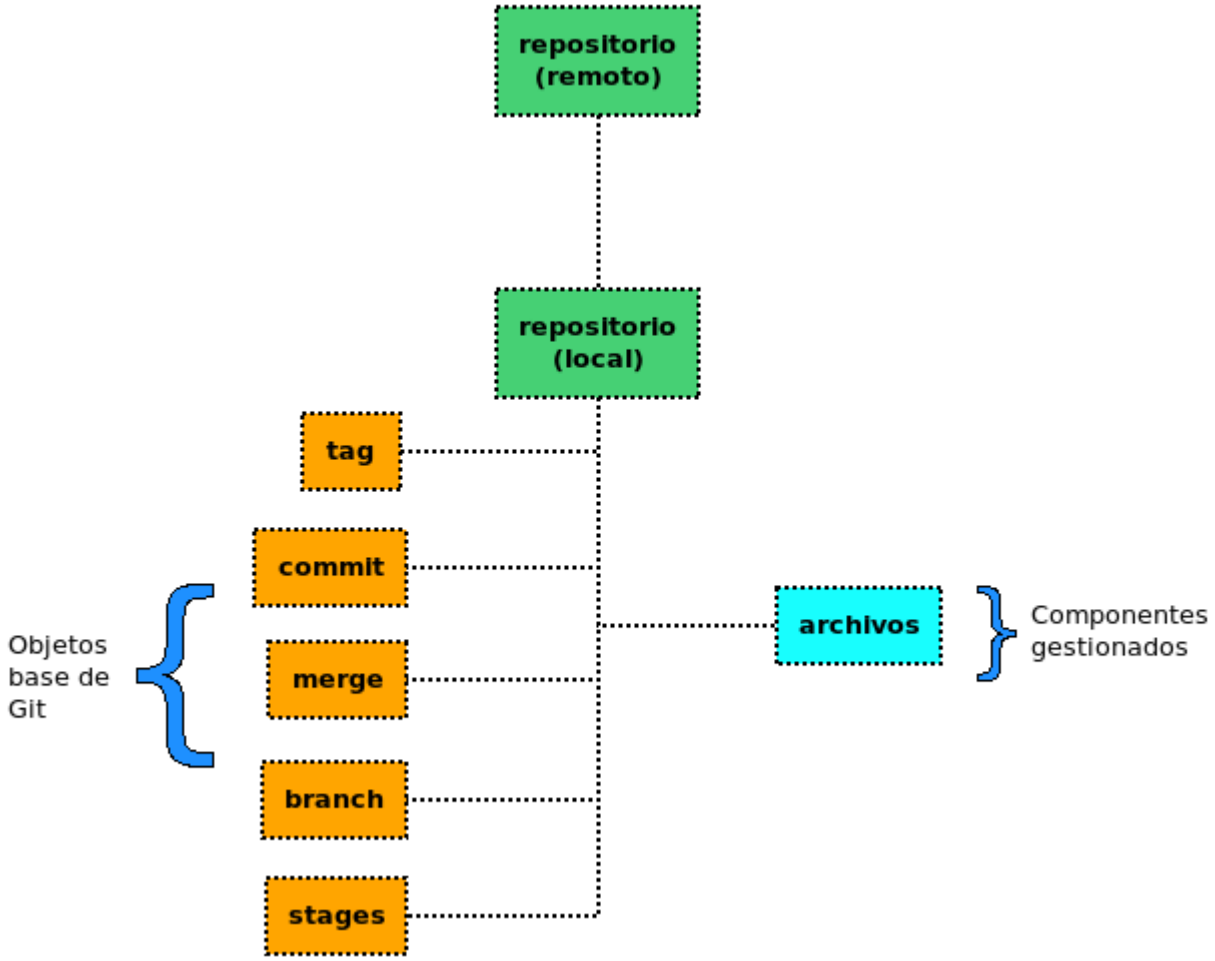

Figura 6: Esquema general de cómo Git estructura el soporte de datos para el manejo de versiones.

2.4 Automatización de tareas

La automatización de tareas que se plantea en este punto corresponde a sistematizar lo máximo posible una serie de tareas, las cuales podrían ser repetitivas y tediosas de realizar. Estas están relacionadas con:

- Compilación de software
- Ejecución de pruebas unitarias
- Empaquetado y disponibilización de software
- Instalación o despliegue en múltiples ambientes (desarrollo, QA, producción)

A continuación se listan y comentan brevemente algunas de las herramientas mencionadas en el punto 2.2.

2.4.1 Travis CI

Esta herramienta permite construir y ejecutar pruebas de manera automatizada. Travis CI existe como servicio pagado (particularmente, para proyectos de desarrollo privados), y tiene un plan gratuito para proyectos de software libre. Adicionalmente, existe también como solución de software libre, que puede ser "armada" a través de varios proyectos. Sin embargo, bajo esta modalidad puede ser difícil llegar a contar con el mismo nivel de prestaciones con las que brinda la herramienta cuando se contrata como servicio. La solución solo se puede integrar con repositorios Git gestionados por Bitbucket y GitHub.

Travis CI funciona "monitoreando" los cambios que pueda tener una o más ramas de un proyecto, y en función de cada una, configurar una o más acciones. Por ejemplo, en un esquema de 3 ramas (producción, QA y desarrollo), al tener cambios en desarrollo, podría disparar la ejecución de pruebas unitarias, pruebas de cobertura y analizadores sintácticos. En ambientes de QA se podrían ejecutar pruebas funcionales y repetir todas las pruebas anteriores. En la rama de producción se podrían generar los empaquetados para ser distribuidos.

Según la configuración que se realice, se pueden disparar distintas notificaciones (correos electrónicos, mensajes instantáneos) a los distintos involucrados, lo que permite que se tomen las acciones requeridas en caso de ser necesario. Las configuraciones para las acciones se realizan usando la especificación provista por Travis CI, y se configura usando el lenguaje YAML[2].

2.4.2 Codeship

Esta herramienta permite automatizar las construcciones del software desde un gestor de repositorio Git (Bitbucket, GitLab o GitHub) y realizar los despliegues o instalaciones de los mismos en alguna plataforma del estilo nube (como Azure o Amazon Web Services). También es posible configurar, crear y ejecutar pruebas unitarias sobre el software.

Codeship funciona como servicio pagado, ofreciendo distintos tipos de planes de acuerdo a las necesidades de cada proyecto o empresa. Éste hace uso de tecnología de contenedores para llevar a cabo la construcción, ejecución de pruebas y despliegue del software.

2.4.3 Circle CI

Circle CI permite definir pipelines, los cuales permiten construir, probar y desplegar el software, al igual que las herramientas antes descritas. Uno de los atributos diferenciadores de esta solución es que permite ejecutar cada tarea del *pipeline* de manera separada, ya sea

2 YAML: YAML Ain't Markup Language. Es un lenguaje para serializar datos con enfoque en la facilidad para ser leída por humanos.

usando contenedores o máquinas virtuales, pudiendo aumentar bajo demanda el poder de cálculo para, por ejemplo, terminar antes la ejecución de un conjunto de pruebas.
También puede desplegar en múltiples servicios de virtualización como nubes privadas (*AWS CodeDeploy, AWS EC2, AWS S3, Google Kubernetes Engine, Microsoft Azure y Heroku)*. Adicionalmente se pueden agregar otras nubes utilizando un conjunto de herramientas disponibles.

2.4.4 Jenkins

Es un servidor que permite automatizar distintas tareas. Uno de los usos más frecuentes es construir el software, ejecutar pruebas unitarias y métricas de calidad sobre el código fuente permitiendo visibilizar problemas tempranamente. También tiene la capacidad de generar notificaciones respecto a las últimas construcciones y el estado de las pruebas.

Dada su naturaleza de software libre, existe un amplio soporte respecto a las herramientas para versionar que soporta, así como la capacidad de ejecutar *scripts* en muchos lenguajes (perl, bash, programas por lotes de Windows, etc.). Además, tiene una cantidad considerable de módulos que permiten agregar funcionalidades.

Es posible, a partir de *scripts* y algunos módulos, realizar despliegues en distintos ambientes, ya sean máquinas físicas, virtuales, contenedores o nubes públicas o privadas. Jenkins es factible de ejecutar localmente o contratar los servicios de terceros para acceder a su uso, lo que otorga flexibilidad a las organizaciones para decidir cómo ejecutar la herramienta.

2.5 Gestores de repositorios de código fuente

En general, un gestor de repositorios de código fuente, permite realizar distintas actividades sobre 1 o más repositorios de código. Dentro de estas actividades se encuentra la gestión de usuarios y los respectivos accesos a los distintos repositorios disponibles (lectura, escritura). También, hay algunos, que extrapolan todas esas funcionalidades a distintas tecnologías para versionar código fuente (Git, Mercurial, bazaar, etc.). Al día de hoy, se ha hecho común agregar otras funcionalidades dentro de la gestión del repositorio y características que permiten gestionar y documentar el proyecto de software en sí, incluyendo *wikis*[3], gestores de incidentes y foros. A continuación se hace una referencia a los sistemas de este tipo, que son los más usados al día de hoy.

3 Wiki: es un sistema que permite mantener de manera colaborativa documentación, pudiendo modificar rápidamente la misma generada por otro usuario.

2.5.1 GitHub

Es un servicio en la nube que permite gestionar repositorios Git. GitHub ofrece una interfaz de usuario que facilita el acceso a personas que están familiarizándose con Git. Al día de hoy, existen muchos usuarios registrados, que al publicar sus desarrollos en esta plataforma, funciona como portafolio de sus trabajos siendo usado como referencia en algunos procesos de selección laboral. También es usado por múltiples proyectos de software libre para alojar el código fuente y la documentación del mismo.

Existe un acceso pagado a los repositorios que está orientado a empresas que necesitan tener sus códigos fuente de manera privada, además de agregar otras funcionalidades asociadas a mayor cantidad de espacio para los proyectos, mayor cantidad de transacciones sobre el repositorio y trazas de auditoría que podrían ser útiles para algunas organizaciones.

Los componentes que conforman GitHub son: un sistema de integración y despliegue continuo, un módulo que permite realizar gestión sobre brechas de seguridad en el código, revisión de pares del código, distintas aplicaciones para integrarse con GitHub (desde *plugins* para los IDE más populares hasta para el *smartphone*), repositorio de código y documentación, gestión de proyectos y gestión de equipos. Grandes empresas usan hoy en día esta herramienta para gestionar sus desarrollos de software, destacando, por ejemplo a Paypal[4], Spotify[5], y Airbnb[6].

2.5.2 Launchpad

Esta plataforma está orientada a los proyectos de software libre y funciona principalmente como un servicio de manera gratuita, aunque también puede ser descargado e instalado localmente.

Los componentes que conforman a Launchpad son: un sistema para gestionar código fuente, uno para hacer seguimiento de *bugs*, un sistema para gestionar requerimientos, uno para realizar traducciones del mismo sistema y uno para poder ayudar a los usuarios del software desarrollado. Uno de los productos de software mantenidos sobre esta plataforma es la distribución de Linux, Ubuntu.

2.5.3 GitLab

Esta plataforma está disponible como servicio en la nube pública y para ser desplegada en una nube privada. Permite realizar la gestión sobre repositorios Git y llevar a cabo distintas

4 PayPal es una empresa que ofrece medios de pago en distintos comercios electrónicos
5 Spotify es una empresa que ofrece el servicio streaming de audio.
6 Airbnb es una empresa que ofrece su plataforma de software para que particulares puedan ofrecer alojamiento a turistas.

integraciones, proveyendo un ecosistema adecuado para definir y soportar un ciclo de desarrollo de software completo.

Los componentes que conforman a GitLab son: un sistema para gestionar accesos a los repositorios, un set de herramientas para gestionar y planificar proyectos integrando los repositorios de código, gestión de repositorios de código usando Git, herramientas para automatizar y verificar la calidad del código y ejecución de pruebas unitarias, un herramienta para gestionar repositorios de paquetes para distintas tecnologías (Conan, Maven, NPM, PyPI, y otros)[7], gestión de incidentes de seguridad sobre el desarrollo, herramientas para el despliegue, gestión de configuración para distintos *stages* de ambientes, métricas para monitorear ambientes productivos y cómo se relacionan con los cambios en el código y herramientas para proteger infraestructura (cortafuegos de aplicación y de aplicación web[8]).

7 Paquetes de dependencias para C/C++, Java, NodeJS y Python respectivamente
8 Web application Firewall (WAF)

3 Proceso de Gestión de la Configuración

Este trabajo de tesis buscó diseñar e implementar el proceso de gestión de la configuración del software, para que resolviera algunos de los problemas que la organización enfrentaba a nivel de proyecto y de cómo este se desarrollaba, sin dejar de lado la operación del sistema en sí, con el objetivo de hacerlos converger.

De acuerdo a lo que se referenció a nivel general, y en específico en el capítulo 1, la organización cuenta con tres etapas para el desarrollo de software:

- Una de desarrollo, en la que los desarrolladores tienen la facultad de realizar múltiples pruebas con bajo impacto.
- Una de certificación de calidad, que consiste en la prueba, en condiciones similares a un entorno real, del software.
- Una etapa productiva, en la que el software es ejecutado y utilizado en un entorno real.

Adicionalmente, y desde un punto de vista operacional, la organización instala el software de manera controlada en los diferentes servidores disponibles, para luego ser usado en cada una de las etapas de desarrollo del mismo. Por lo tanto, el proceso en sí debiese mantener registros considerando las modificaciones que se realizan entre una etapa de desarrollo y otra, a fin de establecer una relación entre los cambios y los despliegues o instalaciones en los servidores.

El ciclo de gestión en cuestión, debiese ser acompañado de herramientas que permitan, al gestor de la configuración, automatizar algunas tareas asociadas a la construcción/compilación del software y generar una biblioteca[9] de paquetes con las versiones ejecutables del software a partir de un punto conocido de las fuentes de este. Esta biblioteca permitiría otorgar independencia respecto de las versiones o estados del software y el servidor en el que se está siendo ejecutado, para evitar la relación en la que la última –y única– versión de cada etapa es la que se instala en los servidores en un momento determinado.

Para este proceso, se plantean cuatro actividades o pasos que deben ser completados para gestionar los cambios que se introduzcan al código y el proceso en el que este se convierte en un equivalente ejecutable consolidado en un paquete, tal como lo resume la figura 7.

9 Una biblioteca se refiere a tener un listado ordenado con todas las versiones de un software.

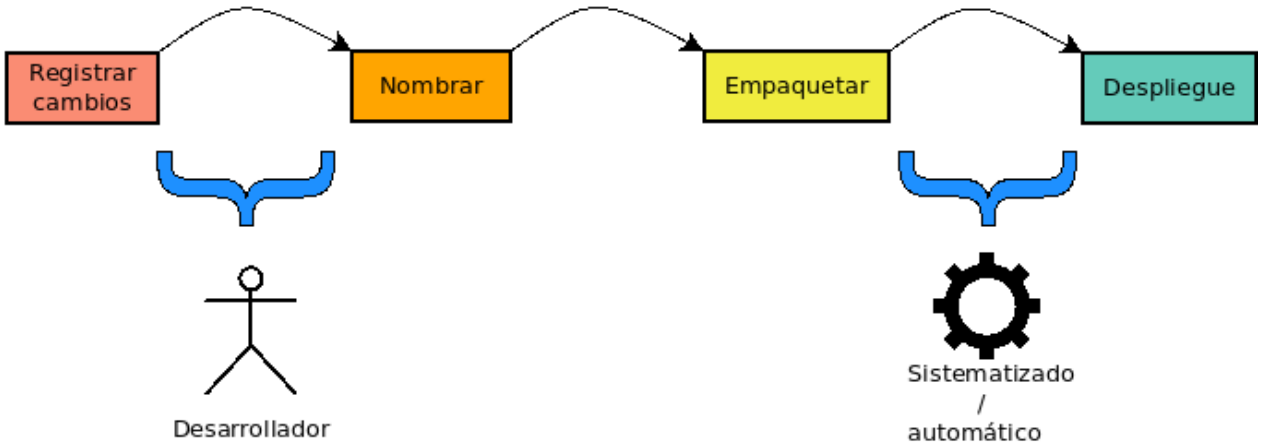

Figura 7: Visión general del proceso de gestión de la configuración.

Las dos primeras actividades, **registrar cambios** y **nombrar**, se orientan a los usuarios que desarrollan y/o hacen la gestión de los componentes. La primera, hace referencia a la identificación de los cambios puntuales que se introduzcan en un nuevo proyecto; además, busca resolver tempranamente los conflictos con otros proyectos que se estén desarrollando de forma paralela y con los que se compartan componentes. Cuando dichos cambios alcancen el nivel de madurez requerido, desde el punto de vista del desarrollo, estos se relacionarán con un nombre que identificará correctamente sobre qué componentes específicos se han realizado pruebas. Cuando el proceso de calidad termina, una modificación del mismo nombre se utiliza para consolidar e identificar la versión final del desarrollo, que es la que termina siendo ejecutada en el ambiente productivo.

En cada una de las iteraciones –incluyendo la final–, para la actividad **empaquetar** se deben construir y generar paquetes, los cuales son publicados y quedan disponibles en una biblioteca para que sean instalados en los distintos ambientes. En la actividad de **despliegue**, el objetivo es que el software sea instalado en los distintos servidores, debiendo estar sistematizado y/o automatizado para evitar la intervención humana.

La figura 8 explica el detalle para cada una de las fases del proceso, donde los puntos con números en círculos están asociados a:

1. Registro de cambios.
2. Nombrado de cambios.
3. Empaquetamiento y archivado de cambios de acuerdo al nombre.
4. Distribución o despliegue de cambios desde la biblioteca de paquetes.

Los círculos de colores naranjo, azul, rosado, amarillo y verde representan distintos cambios dentro del software. Ellos están agrupados en tres bloques importantes, representado un ambiente de desarrollo, QA/UAT y producción. Los cambios que están agrupados en nuevas

funcionalidades corresponden a los nuevos cambios que deben realizarse. Siguiendo el sentido de las flechas representa el tiempo y orden en que se han ido incorporando los cambios. Las flechas que van desde los tres bloques hacia la biblioteca de paquetes, corresponden al lugar donde se consolidan o almacenan los paquetes.

Finalmente, la caja de despliegue corresponde al proceso que se encarga de leer desde la biblioteca de paquetes, y se encarga de ponerlos en algún servidor.

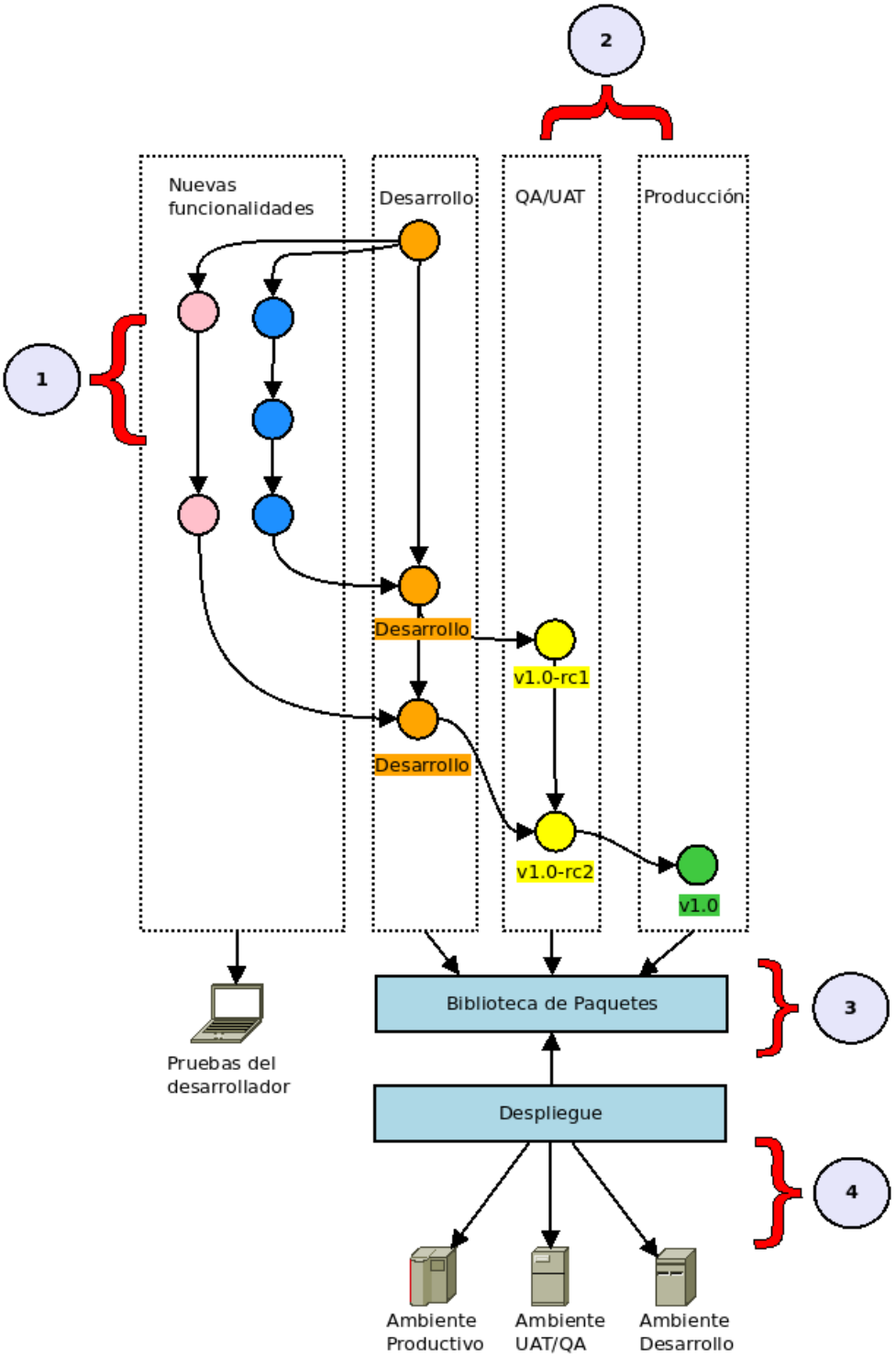

Figura 8: Vista en detalle del proceso para cada una de las actividades.

A continuación, se detalla una especificación para cada paso.

3.1 Registro de cambios

La primera actividad del proceso de gestión de componentes corresponde al registro de todos los cambios realizados; por lo tanto, ésta consiste en identificar todos los elementos existentes que se modificarán o eliminarán, así como también los nuevos que serán agregados. Esto permite ir registrando todos los cambios que se ejecuten sobre ellos, incluso para las piezas nuevas que se agreguen al software. Este registro debe especificar, en concreto:

- Qué se ha modificado o qué se ha agregado.
- Cuándo se realizaron las modificaciones.
- Quién ejecutó los cambios.
- El detalle de las modificaciones.

Este detalle debe tener una gestión, para poder mantener así un inventario histórico de los cambios. Tal como se puede ver en la figura 9, cada registro puede contemplar la modificación de uno o varios componentes y un mismo proyecto puede realizar esta actividad cuantas veces sea necesario; esto se representa con los nodos azules y rosados para dos proyectos separados e independientes, que incluso pueden estar modificando los mismos componentes entre sí.

De manera adicional, en esta etapa de desarrollo, todas las funcionalidades tienen que iniciar con la referencia de la etapa de desarrollo de los componentes que se han gestionado. De esa manera, se puede identificar fácilmente la última versión de los componentes a modificar y, eventualmente, considerar los cambios realizados por otro proyecto.

Al terminar el desarrollo de un proyecto, el registro de los cambios debe ser incorporado a la etapa de desarrollo, pues acá quedará establecida la base para la siguiente etapa del proceso de gestión de la configuración.

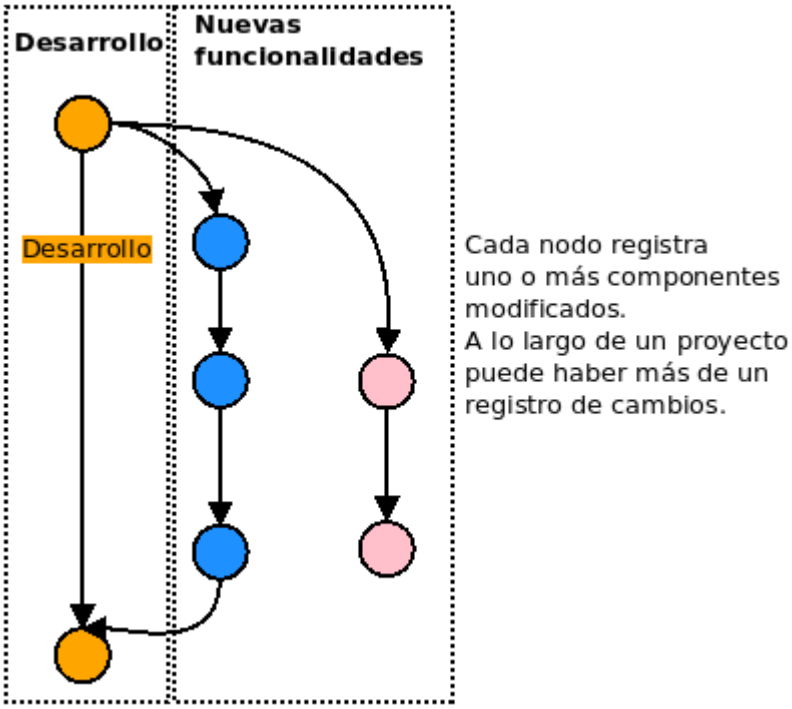

Figura 9: Detalle para registro de cambios.

3.2 Nombrado de cambios

La segunda tarea del proceso de gestión de componentes está asociado a los tres momentos en el ciclo de vida del software, los cuales son los usados por la organización:

- Al *inicio del proyecto*, cuando se está en la etapa de registrar los cambios, todas las modificaciones deben estar asociadas a un nombre que describe la funcionalidad y/o el nombre del proyecto que requiere el cambio. Todos estos cambios corresponden a la **etapa de desarrollo**.
- En la *certificación de calidad del proyecto*, cuando el proyecto ya tiene algún grado de madurez a nivel de desarrollo, o bien cuando se toma la decisión de certificar la calidad del proyecto, es necesario tomar la mezcla que ocurre con el proyecto a nivel de desarrollo y fusionarla con los cambios que se están certificando. En este punto de unión, se identifica con un número de versión superior a la última versión productiva y un iterador asociado a la revisión. En caso de que se deba volver a revisar y aplicar alguna corrección, esta debe ser resuelta desde la funcionalidad que está realizando el proyecto, registrar los cambios, fusionar los que existan a nivel de desarrollo –si es que aplica– y volver a mezclar. En este punto, se debe incrementar el número asociado a la revisión. Esto corresponde a la **etapa de certificación de calidad** del desarrollo.
- Al *final del proyecto*, cuando se alcanza el nivel de calidad requerido y/o se decide terminar las pruebas; se toma el punto identificado con la versión superior a la

existente y la revisión superior y se identifica como final, conservando solo la versión superior. Esta es la **etapa de puesta en producción** del proyecto o evolutivo.

La figura 10 ejemplifica lo anteriormente descrito, donde cada nodo representa una acumulación de cambios registrados en el paso anterior, y por ende, es necesario identificarlo con un nombre. Para el caso de los cambios que se encuentren en etapa de desarrollo, estos siempre usan el mismo nombre.

A los cambios que se encuentran en etapa de certificación se les asocia el nombre de la versión, con el número de iteración/revisión en que se encuentre. Por ejemplo, en la figura 10 se aprecia que en la etapa de certificación de calidad (QA/UAT) existen dos nodos amarillos (v1.0-rc1 y v1.0-rc2), los cuales provienen de dos nodos naranjos (desarrollo, ambos con distintos cambios involucrados).

Finalmente, cuando se ha alcanzado un nivel de calidad esperado, esta versión se consolida con el número de versión (por ejemplo, v1.0), sin considerar la revisión (es decir, sin rc1 y rc2).

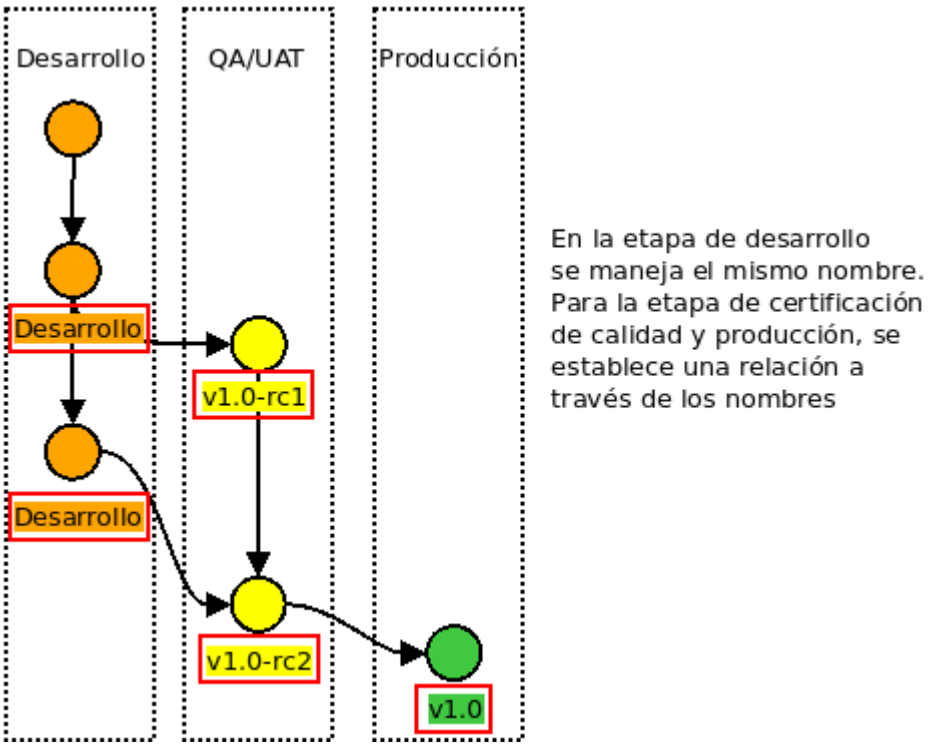

Figura 10: Ejemplo de asignación de nombres a los cambios.

3.2.1 Generación de nombres

En todo momento del desarrollo es necesario identificar el cambio, pero también se requiere el establecimiento de una relación entre lo que existe en el repositorio de código y lo que está instalado en los diferentes ambientes. Si bien se puede usar cualquier estrategia, se sugiere una nomenclatura, la que se ha puesto en marcha en la organización y es explicada en la tabla 4.

Tabla 4: Propuesta de asignación de nombres para cada etapa.

Etapa	Descripción	Propuesta	Ejemplo
Desarrollo de funcionalidad	Nombre asociado a la funcionalidad desarrollada o a algún proyecto.	Debe corresponder al nombre de una rama nueva a partir, de la de desarrollo. No identifica un punto en particular, si no que la rama completa. Debe incluir un identificador de proyecto o evolutivo y el nombre de la funcionalidad separado por guiones para los espacios.	AMDR-666-Agrega-cálculo-de-renta
Desarrollo	Sin nombres.	Solo tiene puntos de mezclas desde la etapa anterior.	No aplica.
Certificación/ QA/UAT	Nombre con un nivel de madurez asociado: un número mayor, un número menor y un nivel de madurez.	El punto de mezcla debe identificarse por el prefijo. **[versión-mayor].[versión-menor]-rc[número-de-iteración]** A diferencia de la etapa "Desarrollo de funcionalidad" corresponde a un punto en particular del proyecto.	v1.1-rc1 v1.1-rc2 v1.1-rc3 v1.1-rc4 v1.2-rc1 v1.3-rc1 v1.3-rc2
Producción	Nombre cerrado asociado a un número mayor y un número menor	El punto de mezcla debe identificarse por el prefijo **v[versión-mayor].[versión-menor]** También corresponde al punto de mezcla desde la etapa anterior, pero establece un hito de cierre de la versión. A partir de este punto, no se pueden	v1.1 v1.2 v1.3

		generar nuevos candidatos de versión en la etapa anterior.	

La figura 11 muestra un ejemplo de la aplicación de la tabla antes descrita. Al seguir la secuencia mostrada en dicha figura, se tiene que:

1. Existen dos funcionalidades que inician en el mismo punto de la etapa de desarrollo. El desarrollo de ambos productos de software sigue caminos independientes. En la versión 1.a se considera registrar 3 cambios en su línea de desarrollo, y un cambio en la 1.b.

2. Ambas funcionalidades terminadas deben mezclarse para crear una nueva versión, que debe consolidarse en la etapa de desarrollo.

3. Después de haber realizado algunas pruebas en conjunto, la funcionalidad 1.b requiere hacer modificaciones, y si bien ya no hay funcionalidades en paralelo, requiere mezclarse con lo que ya existe en desarrollo y que incluye al cambio 1.a.

4. La nueva versión contiene los cambios que hizo 1.a y las correcciones que recibió 1.b en 3, mezclando nuevamente ambas funcionalidades.

5. En la etapa de QA/certificación, se puede ver que hay dos revisiones para la versión 1. De esto se puede sugerir que los problemas fueron detectados por el control de calidad, y no por las pruebas que se puedan haber realizado en la etapa de desarrollo.

6. Finalmente, en la etapa productiva, se obtiene un nombre finalizado. Al revisar los nombres candidatos del punto 5, se concluye que esta versión tuvo dos revisiones formales.

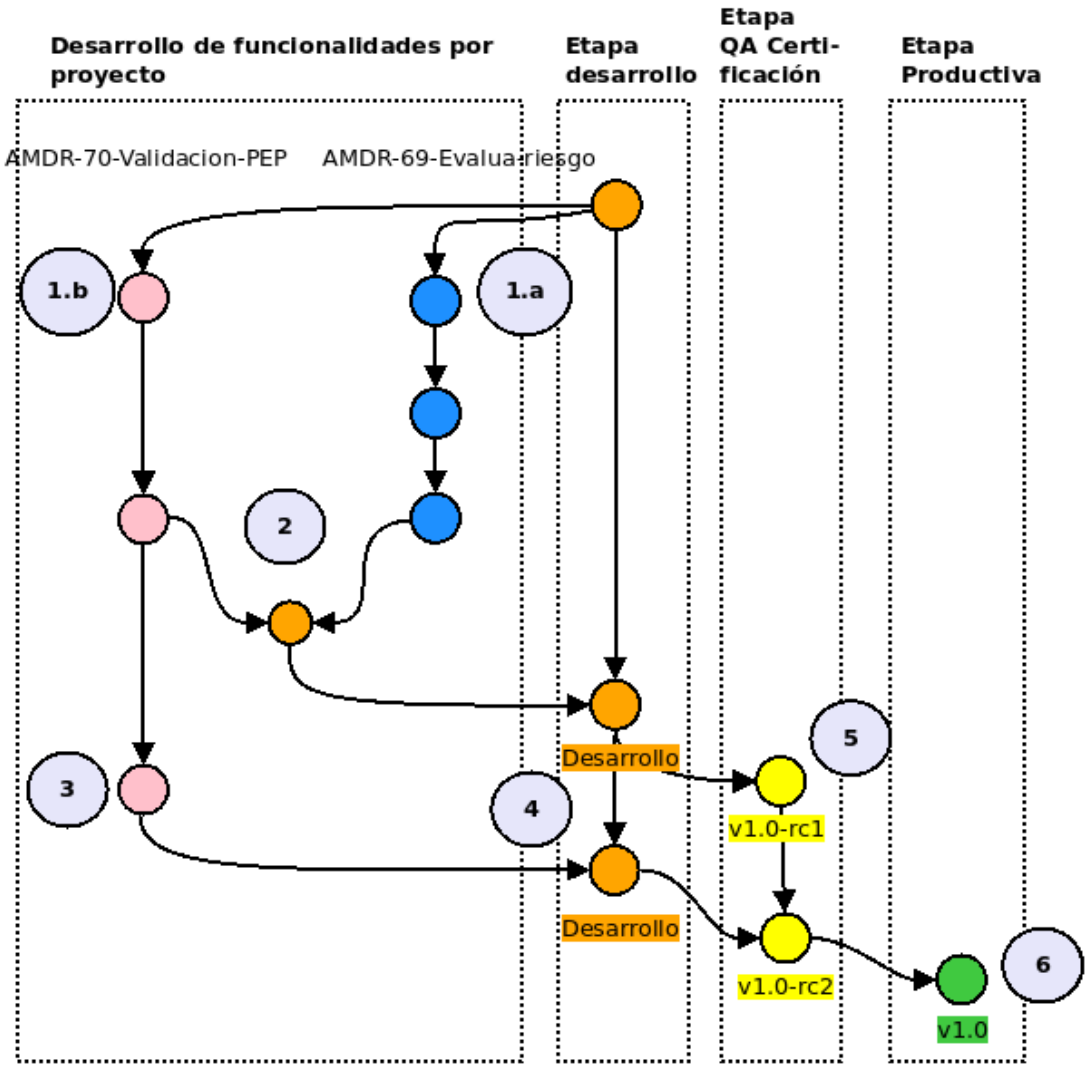

Figura 11: Nombrado y nomenclatura de uso en las distintas etapas.

3.3 Empaquetamiento y archivado de cambios

La tercera etapa del proceso, se refiere a tomar algún punto de los identificados en la etapa de nombrado, y realizar las acciones necesarias para que el software sea ejecutable con todos los componentes asociados a ese punto. Además, que el identificador quede asociado a un instalador que permita replicar el comportamiento del software, independientemente de dónde éste sea ejecutado.

En esta etapa, por ejemplo, se deben compilar todos los programas fuentes asociados a la versión, y los binarios generados de ellas. El resultado debe quedar en un archivo comprimido que permite que esta versión sea distribuida fácilmente. Idealmente, se debe incluir algún identificador que permita al software en su ejecución, indicar una referencia a la versión de éste; es decir, que el software informe con qué versión de sus fuentes fue compilado. A lo largo

del tiempo se tendrán múltiples versiones compiladas en distintos momentos del desarrollo, las cuales en conjunto formarán una biblioteca con versiones para producción, QA y eventualmente desarrollo.

Esta parte del proceso, requiere ser realizada de manera sistemática y ojalá automatizada, evitando así la intervención humana. Esto con el fin de mantener consistencia entre los registros de los cambios y los nombres dados, y evitar introducir nuevos cambios de manera manual. En el ejemplo de la figura 12 se puede apreciar que para los 5 puntos nombrados (producto de la etapa anterior), se tiene un total de 4 paquetes:

- Un solo paquete de desarrollo[10], el cual siempre considera la última versión de desarrollo.
- Dos paquetes producto de las iteraciones en la revisión de calidad: v1.0-rc1 y v1.0-rc2.
- Un paquete con fines productivos: v1.0.

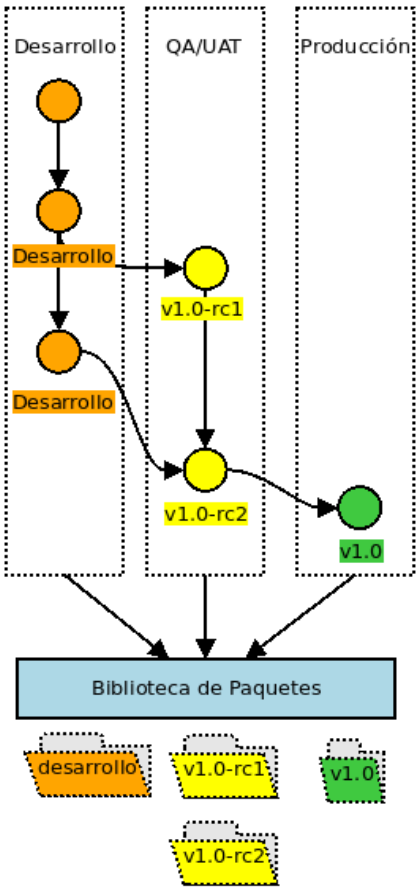

Figura 12: Biblioteca de paquetes a partir de los nombres dados.

10 Esto se definió así, porque a nivel de desarrollo podrían generarse muchos paquetes y ser contraproducente. Esto está respaldado en la experticia del equipo de desarrollo, el cual está constantemente agregando nuevas funcionalidades y/o trabajando sobre el software.

3.4 Despliegue de cambios

La última parte del proceso (es decir, el despliegue) es la encargada de realizar la instalación del sistema en los distintos servidores. Dado que en el paso anterior se ha generado una biblioteca o colección con paquetes, los cuales están asociados a su nombre, es necesario recurrir a ésta para seleccionar la versión origen de la instalación. Esta actividad debe ser realizada de manera automática, y de esa forma disminuir al máximo la intervención humana, ya que en general esta actividad está asociada a tareas repetitivas y tediosas donde podría ser fácil cometer errores.

Idealmente, en este despliegue se deben poner como objetivos los distintos servidores que están asociados a las distintas etapas que se tienen en el desarrollo de software (por ejemplo, desarrollo, QA y producción). Así los componentes de software solo son ejecutados en un servidor determinado, pero la gestión de los mismos se realiza de una manera controlada e independiente. Por ejemplo, en la figura 13 se observa que, a partir del proceso de despliegue, se han obtenido desde la biblioteca los nombrados con la versión:

- v1.0 del servidor productivo.
- v1.0-rc2 del servidor donde se hacen las pruebas.

El único paquete de desarrollo corresponde a los últimos componentes modificados existentes en esa etapa.

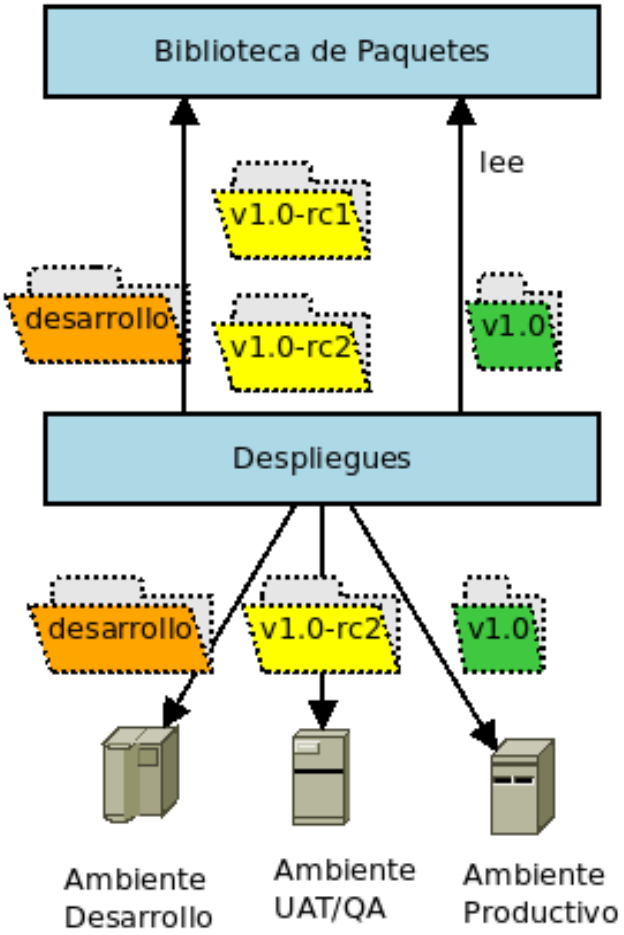

Figura 13: Despliegue de versiones particulares de un paquete, obtenido desde la biblioteca.

Si el despliegue cuenta con alguna flexibilidad, se podrían dar escenarios nuevos. Por ejemplo, la figura 14 muestra la instalación de una versión de desarrollo en un servidor de certificación, con el fin de aprovechar el entorno similar a uno productivo (disponibilidad de datos o accesos de seguridad).

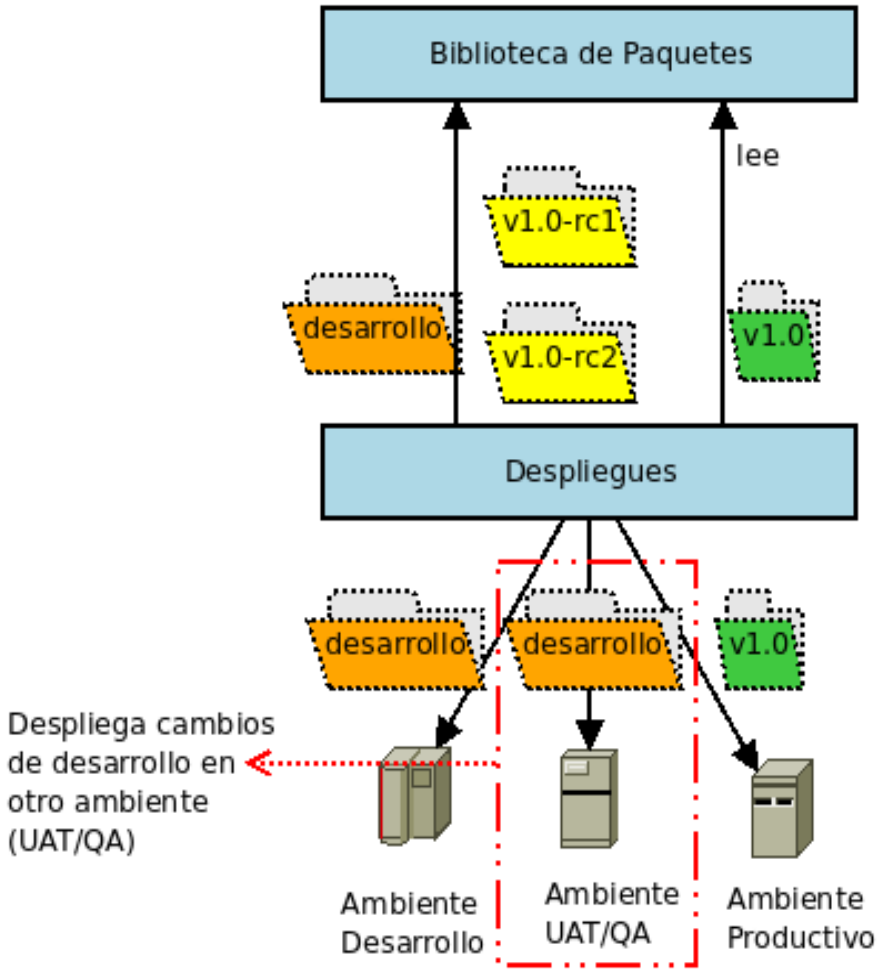

Figura 14: Despliegue de una versión de desarrollo en un servidor distinto a ese fin.

3.4.1 Problemas en los despliegues

En general, modificar las versiones en ambientes de desarrollo UAT/QA es una señal de que se ha encontrado algún problema y debe corregirse, forzando a los equipos a crear una nueva versión. Por otro lado, cuando se detecta un error en un ambiente productivo durante una actividad programada, el equipo que supervisó la actividad, debe recabar todos los antecedentes asociados al problema presentado e instalar la versión inmediatamente anterior.

3.5 Migración de Componentes

Este proceso requiere establecer una línea base o versión inicial de los componentes a gestionar, y convertirlos en referencia para los cambios que se comiencen a hacer. Para el caso

donde no hay ninguna formalidad respecto a la gestión de componentes, se deben tomar los componentes que están instalados en el ambiente productivo, y desde ahí empezar a trabajar.

En caso de que exista algún proceso de gestión de componentes, es necesario revisar en detalle qué se puede homologar contra esta propuesta, y analizar que lo que se descarte no sea relevante para el proceso nuevo.

Respecto a la historia que pudiera existir, es necesario revisar si es que ésta puede ser rescatada e incorporada a la nueva herramienta pero, y tal como ocurre con el caso anterior, se la puede incorporar como línea base, y sobre ella seguir trabajando. Para esta tarea existen distintas alternativas que permiten mover la historia de una herramienta a otra.

4 Evaluación de Herramientas

A partir de las herramientas mencionadas en el capítulo 2, se realizó una evaluación comparativa con el objetivo de identificar qué herramienta satisface de mejor forma la instrumentalización del proceso propuesto. Para realizar esta evaluación se utilizó un instrumento propuesto por una consultora con presencia internacional, el cual fue aprendido por el autor.

Para el caso de esta evaluación el instrumento fue simplificado y se eliminaron dimensiones que no eran relevantes para el proyecto analizado; sólo se dejaron tres dimensiones (funcional, técnica y experiencia), donde cada una considera una serie de atributos propios de la dimensión y un factor asociado a la misma. Finalmente, se consideró la cobertura de cada atributo por parte de la herramienta; es decir, si la herramienta provee el atributo evaluado, si se requiere configuración, o si hay que hacer algún desarrollo extra. Al ponderar y sumar todos los valores, obtenemos la calificación final de la herramienta que va a ir entre 0 y 1, donde 1 indica que la herramienta cumple con todos los requerimientos, y 0 indica que no cumple con ninguno de los requerimientos. La puntuación asignada fue discutida con el equipo de trabajo, para llegar a un valor de consenso que podía tomarse como referencia. A continuación se detalla el procedimiento de evaluación de herramientas, y luego se presentan las dimensiones, los atributos y los niveles de cobertura de las mismas.

4.1 Procedimiento de evaluación de herramientas

El proceso de evaluación de herramientas de software tiene principalmente dos objetivos: 1) identificar una solución de software autocontenida, o bien la combinación de varias soluciones con el fin de satisfacer los requerimientos asociados a alguna actividad de negocio en particular (gestión de ventas, gestión de inventarios, evaluación de riesgo, gestión de carteras, etc.), y 2) aumentar los beneficios comerciales, al aplicar dicha solución a la actividad de negocios seleccionada, y ver cómo se puede lograr este beneficio en el menor tiempo posible (*time-to-market*). Para llevar a cabo esta evaluación se realizan las siguientes actividades:

- Levantar requerimientos y funcionalidades
- Seleccionar proveedores (y su software) a través de una especie de licitación. La idea es que, en base a lo relevado, ver qué tan bien lo puede cubrir la herramienta de cada proveedor.
- Evaluar cada producto (y acá es donde entra una parte de lo que yo consideré). En este trabajo de tesis no se abordaron los aspectos económicos y las demostraciones (o pruebas de concepto) que se puedan ofrecer los proveedores de dichos productos.

4.1.1 Levantar requerimientos y funcionalidades

En términos generales, se deben listar todas las funcionalidades deseadas y complementarlas con un diagrama de procesos. Esto permitirá al equipo de consultores identificar:

- Funcionalidades que deben ser estándar y provistas por las herramientas disponibles en el mercado.
- Adaptaciones o integraciones que debieran ser provistas por las herramientas disponibles en el mercado.
- Correcciones o modificaciones que se deben hacer a los procesos internos.

Teniendo en consideración los objetivos mencionados anteriormente y las funcionales listadas, se debe ubicar en el mapa de capacidades que tiene la organización, y si la capacidad ya existe o se debe crear.

Esta actividad puede sentar las primeras restricciones técnicas que se deben satisfacer, o excepciones que se deben considerar. También podría develar si es que hay alguna herramienta que no esté siendo utilizada en su totalidad y podría satisfacer las necesidades y objetivos anteriormente planteados.

Con todos esos antecedentes recolectados, se debe realizar una búsqueda de proveedores, y ver (a alto a nivel) cómo los productos de software de ellos pueden satisfacer los requerimientos y las brechas que pueden existir respecto a las necesidades de la organización.

4.1.2 Seleccionar proveedor

Para seleccionar un proveedor, a todas las empresas se les genera un RFI (*request for information*), de los cuales se espera una propuesta. Con todas las propuestas recibidas, se realiza una comparación de las alternativas de los distintos proveedores, y del respaldo que brindan dichas empresas. Luego se *rankean* las propuestas, y se comienzan las negociaciones, para finalmente tomar una decisión; es decir, la selección de una propuesta.

4.1.3 Evaluar las propuestas

La evaluación de la propuesta considera las siguientes dimensiones:

- Funcional: busca evaluar si es que los requerimientos funcionales declarados están completamente satisfechos por el producto ofrecido.

- Técnica: considera distintos aspectos, como puede ser:
  - arquitectura

    - seguridad
    - mantención
    - soporte
    - reportes
    - extensibilidad
    - documentación del sistema
    - tecnología con que está implementada la solución
    - capacidad de integrarse con otros sistemas, etc.
- Proveedor: evalúa la experiencia y capacidades del proveedor, teniendo en cuenta:
    - respaldo económico y financiero
    - nivel de presencia y antigüedad en el mercado
    - historia del producto
    - certificaciones y alianzas con otros productos
- Proyecto: busca evaluar la forma de abordar un proyecto, y cómo éste puede plantear una mejor estrategia frente a otra alternativa. Para ello se analiza cómo se abordan las distintas fases del proyecto, considerando por ejemplo:
    - gestión del proyecto
    - plan de trabajo
    - equipo de proyecto
    - calidad, soporte y evolutivos
    - compromiso con el proyecto.
    - experiencia en proyectos similares
- Oferta económica: considera los precios informados por cada proveedor, los costos recurrentes que se puedan requerir como parte de la operación usando el producto, y cómo estos montos impactan la inversión y costo del proyecto en sí.
- Capacidad y calidad de las pruebas de concepto (o demos) por parte del proveedor: se considera la disposición del proveedor para mostrar las distintas funcionalidades del software, y en general, las más críticas de la solución. Estas sesiones retroalimentan la evaluación de las dimensiones anteriormente planteadas.

Las dimensiones de evaluación se pueden agrupar en tres categorías:

- Evaluación técnica. Esta aborda aspectos funcionales, técnicos, del proveedor y del proyecto. Ésta corresponde a una ponderación de la priorización de las distintas dimensiones, bajo el criterio que el usuario final o dueño del proyecto determine.
- Evaluación de las pruebas de concepto. Se realiza teniendo en cuenta la percepción de los usuarios respecto a la cobertura que el proveedor y herramienta pueda ofrecer, y cómo el proveedor se desempeña en dicha prueba.
- Evaluación económica. Se hace teniendo en cuenta costos recurrentes y proyecciones en un determinado tiempo.

4.1.4 Elaboración de plan e Implementación

Conociendo las fortalezas y limitaciones de la solución seleccionada, se procede a hacer una estimación de esfuerzo (tiempos y costos) que se requerirá para llevar a cabo el proyecto, y generar un plan de trabajo de alto nivel. Finalmente, el plan es implementado generando todos los proyectos requeridos para lograr el objetivo final, que es poner en marcha la solución encontrada.

4.2 Dimensiones de evaluación

Tal como se mencionó antes, se consideraron tres dimensiones (o aspectos) para la evaluación: funcional, técnica y experiencia. Para la dimensión funcional se consideraron las ideas planteadas en el capítulo anterior, por ejemplo, la capacidad de crear ramas, identificar versiones, etc. En la dimensión técnica, se tuvo en cuenta diversos temas asociados al software en sí (por ejemplo, documentación, seguridad, etc.). Finalmente, se sopesó el aspecto de experiencia, donde se consideraron la experticia que el autor tiene en el uso de las herramientas, y la disponibilidad de una red de contactos para abordar los eventuales problemas que pudieran aparecer durante la adopción. En la tabla 5, quedó reflejado la ponderación y/o importancia que los equipos le dieron a cada una de las dimensiones de evaluación. Como se mencionó antes, la puntuación asignada fue discutida y acordada con el equipo de trabajo.

Tabla 5: Ponderación de cada dimensión de evaluación.

Dimensión	**Ponderación**
Funcional	40%
Técnica	30%
Experiencia	30%

4.3 Atributos por dimensión

Para cada una de las dimensiones indicadas en el punto anterior, se definieron atributos. A cada uno de ellos se le asignó un porcentaje de relevancia en la dimensión a la que pertenecen. Los siguientes puntos muestran el detalle de cada uno de estos atributos, y el porcentaje de importancia.

4.3.1 Atributos de la dimensión técnica

En este ámbito se consideraron los siguientes atributos técnicos, los cuales fueron ponderados en cuanto a importancia de acuerdo a la tabla 6:

1. *Adaptabilidad*: Capacidad de ser adaptado de forma efectiva y eficiente a diferentes entornos, es decir, si puede ser ejecutado en un *cloud* público o privado o como servicio.
2. *Seguridad*: Provee diversos aspectos relacionados con seguridad como control de acceso, doble factor de autenticación, uso de llaves SSH, filtros de accesos por IP, etc.
3. *Documentación*: Cuenta con documentación adecuada, explicación en detalle de las funcionalidades, ejemplos y foros públicos de discusión de problemas asociados.
4. *Soporte*: En caso de tener problemas con la herramienta, es posible acceder a ayuda por parte del fabricante de la herramienta.
5. *Extensibilidad*: Puede agregar nuevas funcionalidades sin afectar a las existentes.
6. *Portabilidad*: Hay aplicaciones cliente compatibles con los sistemas operativos existentes en la organización (Windows, Linux y MacOS).

Tabla 6: Porcentaje de importancia para atributos de la dimensión técnica.

Dimensión	**Ponderación**
Adaptabilidad	20%
Seguridad	10%
Documentación	20%
Soporte	20%
Extensibilidad	20%
Portabilidad	10%

4.3.2 Atributos de la dimensión funcional

Funcionalmente, se consideran los atributos descritos en la sección 1.1, los cuales se describen brevemente a continuación:

1. *Flujo definido*: Permite seguir el flujo planteado en la solución.

2. *Identificación de versiones*: Permite identificar versiones a nivel de repositorio de código.
3. *Empaquetamiento de la solución*: Permite gestionar paquetes y construir una biblioteca con los mismos.
4. *Desarrollo paralelo*: Permite tener más de un desarrollo en el repositorio de código.
5. *Automatización del despliegue*: Permite automatizar los despliegues.

En la tabla 7, se establecen los porcentajes de importancia (relevancia) para cada atributo en esta dimensión.

Tabla 7: Porcentaje de importancia para atributos de la dimensión funcional.

Atributo	**Porcentaje de importancia**
Flujo definido	30%
Identificación de versiones	20%
Empaquetamiento de la solución	15%
Desarrollo paralelo	15%
Automatización del despliegue	20%

4.3.3 Atributos dimensión experiencia

Esta dimensión considera atributos asociados a la experiencia del equipo que implementará la solución con la herramienta en cuestión. Los atributos a considerar son los siguientes:

1. Experiencia previa: Evalúa si el equipo de trabajo ha utilizado previamente la herramienta.
2. Curva de aprendizaje: Considera la facilidad de poder aprender a usar la herramienta, ya sea siguiendo tutoriales o leyendo la documentación.
3. Soporte interno: Considera si hay personas dentro de la organización (fuera del equipo de trabajo) que conozcan la herramienta y sus problemas habituales. Estas personas podrían actuar como soporte interno ante problemas que pudieran aparecer en la implementación de la solución.

La ponderación de importancia de dichos atributos se indica en la tabla 8.

Tabla 8: Porcentaje de importancia para atributos de la dimensión experiencia.

Atributo	**Porcentaje de importancia**
Experiencia previa	40%
Curva de aprendizaje	25%

Soporte interno	35%

4.4 Cobertura de cada atributo

Para cada atributo evaluado, se debe considerar el nivel de cobertura que brinda la herramienta. Para ello se consideraron los siguientes niveles de cobertura:

1. *Completa*: El software implementa nativamente la funcionalidad/capacidad requerida.
2. *Casi completa*: El software requiere ser configurado para lograr cumplir con lo requerido.
3. *Incompleta*: El software requiere un desarrollo inferior a 4 semanas para cumplir.
4. *Mínimo*: El software requiere un desarrollo entre 4 y 12 semanas para cumplir.
5. *Deficiente*: El software requiere un desarrollo superior a 12 semanas.
6. *Ausencia*: El software no tiene la funcionalidad/capacidad requerida, ni es factible agregarla por parte del usuario.

La tabla 9 muestra el factor asignado a cada nivel de cobertura, y el porcentaje de funcionalidad cubierta.

Tabla 9: Porcentaje de cobertura.

Nivel de cobertura	Porcentaje de cobertura	Factor asignado
Completa	95-100%	1
Casi completa	85-94% configuración	0,85
Incompleta	75-84% desarrollo menor	0,75
Mínimo	50-74% desarrollo medio	0,5
Deficiente	25-49% desarrollo mayor	0,25
Ausencia	0-24%	0

4.5 Uso de parámetros

Al obtener la ponderación de la dimensión del atributo, considerando la importancia (relevancia) del mismo y la cobertura que éste entrega, se podrá determinar cuál es la herramienta que obtiene una mayor valorización. La fórmula asociada a este cálculo es la siguiente:

$$\Sigma\ dimension_i \ * atributo_i * cobertura_i$$

En la tabla 10 se muestra el resumen de la evaluación de cada producto (los detalles se encuentran en el anexo 1), y de ésta se puede concluir que el producto que brinda mayor

soporte es GitLab. Por lo tanto, este producto fue seleccionado para dar soporte al proceso propuesto.

Tabla 10: Resumen de la evaluación de productos.

Producto	**Resultado de Evaluación**
GitLab	0,93
GitHub	0,73
Jenkins + Bitbucket	0,85
Codeship + Bitbucket	0,56
Solución inhouse (SVN)	0,46

5 Evaluación de la Solución Propuesta

La solución propuesta fue evaluada considerando el desarrollo de proyectos evolutivos (actualizaciones) de los sistemas actuales de la organización, y también el desarrollo de un proyecto completo desde cero. La diferencia entre ambos tipos de proyecto radica principalmente en su duración, donde un proyecto evolutivo puede tomar entre 3 y 10 semanas, mientras que un proyecto desde cero contempla alrededor de 18 meses (dependiendo del proyecto).

Para cuantificar los cambios en los proyectos involucrados en esta evaluación, se tabularon los datos asociados a ellos, considerando los siguientes aspectos:

- *Código del proyecto:* Este es el id usado para identificar el proyecto evolutivo o el proyecto desde cero que se está evaluando.
- *Cantidad de entregas realizadas*: Cada entrega involucra actualizaciones al sistema, realizadas por un desarrollador. Cada vez que una nueva versión del sistema pasa del ambiente de desarrollo al de QA, se computa como una nueva entrega.
- *Cantidad de fuentes intervenidas durante el proyecto*: Se refiere al total de componentes creadas, modificadas o borradas, considerando todas las entregas involucradas en un proyecto.
- *Cantidad de versiones disponibles en QA*: Corresponde a las versiones del sistema que fueron aprobadas por el equipo de aseguramiento de la calidad (QA).
- *Situación final*: Describe en qué estado quedó el proyecto, e incluye información relevante respecto a la experiencia del proceso que se llevó a cabo.
- *Cumplimiento de los objetivos específicos enunciados en esta tesis* (según lo descrito en la sección 1.2), particularmente:
  - *Adherencia al ciclo de vida*: Indica si se siguieron los pasos propuestos por el proceso.
  - *Independización de versiones de los servidores*: Indica si el estado de los componentes se encuentra en un repositorio externo.
  - *Automatización del despliegue*: Indica si el despliegue es realizado de manera automática.

A continuación se presentan los resultados de aplicar el proceso en cinco proyectos evolutivos, y luego se muestra cómo resultó cuando se aplicó en un proyecto desde cero. Los proyectos evolutivos fueron desarrollados de manera lineal y cronológica, pero en paralelo entre ellos. Para todos los evolutivos se estuvo completando el proyecto desde cero, es decir, siempre en la cartera de proyectos hubo dos en ejecución.

Las siguientes secciones muestran los datos tabulados, acompañados de la representación gráfica de la aplicación de la metodología en cuanto a cada entrega, y cómo ésta se iba integrando en los ambientes de desarrollo, QA y producción. Cada nodo representa una entrega, y las flechas van mostrando las mezclas que van ocurriendo entre los distintos ambientes, y los nombres que van tomando a medida que el proyecto va avanzando.

5.1 Proyectos evolutivos

5.1.1 Proyecto 1

El primer proyecto fue una actualización de las definiciones bases para la ejecución de interfaces con diversas fuentes de datos. La actualización tenía una duración prevista de 8 semanas de trabajo efectivo. Tal como muestra la Tabla 11, el proyecto tuvo 8 entregas, que involucraron la modificación del código fuente de 214 archivos.
El equipo ejecutor del proceso era nuevo, lo cual permitió experimentar con confianza el uso del nuevo proceso (no hubo resistencia al cambio); además, permitió contrastar resultados con los de los equipos anteriores. El equipo anterior no dejó trazas de los cambios realizados en el software, lo cual dejó muchas dudas acerca del estado del proyecto en sí. El equipo actual pudo apoyarse en el control de los cambios y no desconfiar de los mismos, tal como ocurrió al inicio. Si bien el proyecto fue desestimado (no se puso en producción), permitió realizar una evaluación preliminar del proceso propuesto y de los cambios introducidos en él.

Tabla 11: Resumen del proyecto 1

Código del proyecto	AMDR-38
Cantidad de entregas realizadas	8
Cantidad de fuentes intervenidas durante el cambio	214
Cantidad de versiones disponibles en QA	0
Situación final	Evolutivo desestimado. Este proyecto se utilizó para realizar pruebas y evaluar el impacto del nuevo proceso.
Adherencia al ciclo de vida	Se pudo seguir sin problemas el flujo definido.
Independización de versiones de los servidores	Se definen líneas bases. Se pudo usar el servidor de QA (mejores prestaciones) con la rama asociadas al proyecto, **sin generar una versión para desarrollo ni QA**.
Automatización del despliegue	Los despliegues se hicieron correctamente.

La figura 15 describe los cambios que ocurrieron a nivel de la rama AMDR-38 pero estos fueron desestimados.

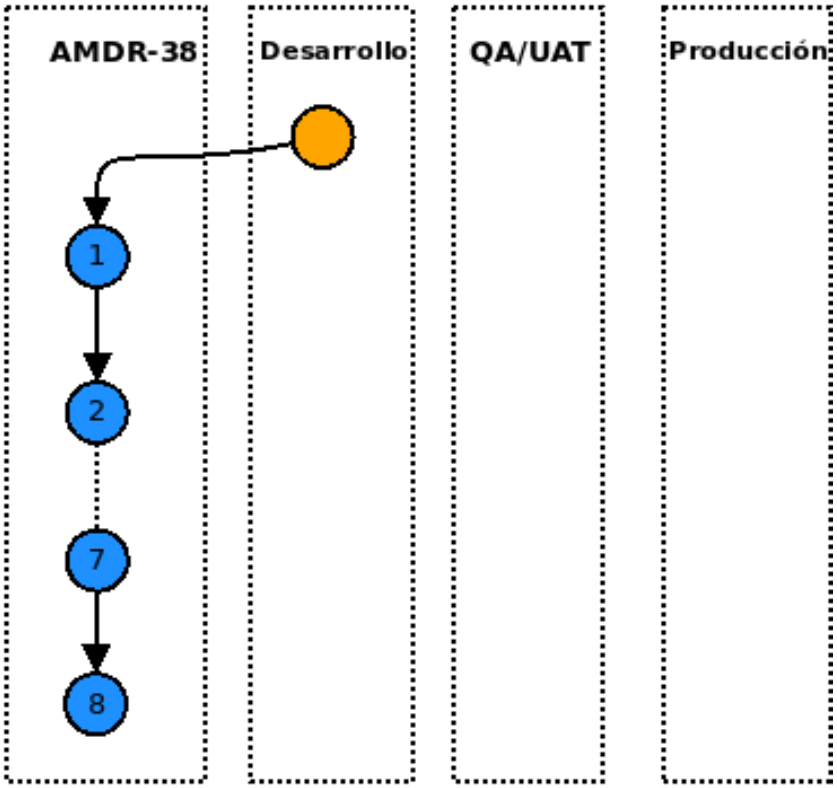

Figura 15: Representación gráfica para el proyecto 1. Este proyecto, tuvo 8 entregas pero fue desestimado.

5.1.2 Proyecto 2

En el caso del segundo proyecto, se trató de una modificación asociada a la forma en que se realizaban los cálculos para determinar si un cliente era prospecto para otorgarle un producto de crédito. En este proyecto se realizaron 125 modificaciones a piezas de software, y si bien los cambios conceptualmente eran sencillos, la implementación requirió cambios más profundos.

Estos cambios fueron sometidos a revisión por el equipo de QA en dos oportunidades, y mientras se revisaba el primer release se seguían haciendo cambios. Tal como se indica en la tabla 12, la instalación del software en el ambiente productivo tomó menos de un minuto, lo cual es muy bueno comparado con el escenario anterior de paso a producción. En el escenario anterior el operador encargado de la actualización debía copiar manualmente cada uno de los componentes, hacia cada uno de los servidores, a partir de una pauta.

La figura 16 describe los cambios que ocurrieron a nivel de la rama AMDR-91 el cual contempló 15 modificaciones, dos mezclas en desarrollo y dos revisiones en QA.

Tabla 12: Resumen del proyecto 2

Código del proyecto	AMDR-91
Cantidad de entregas realizadas	15
Cantidad de fuentes intervenidas durante el cambio	125
Cantidad de versiones disponibles en QA	2
Situación final	Instalación en producción exitosa. La instalación tomó menos de un minuto, lo cual era uno de los objetivos a alcanzar (agilizar la puesta en producción).
Adherencia al ciclo de vida	Se pudo seguir el ciclo de vida sin problemas, generando más de un *release* en QA.
Independización de versiones de los servidores	Se definieron líneas base, y las versiones quedaron independientes de los repositorios de código en los servidores.
Automatización del despliegue	Los despliegues se hicieron correctamente.

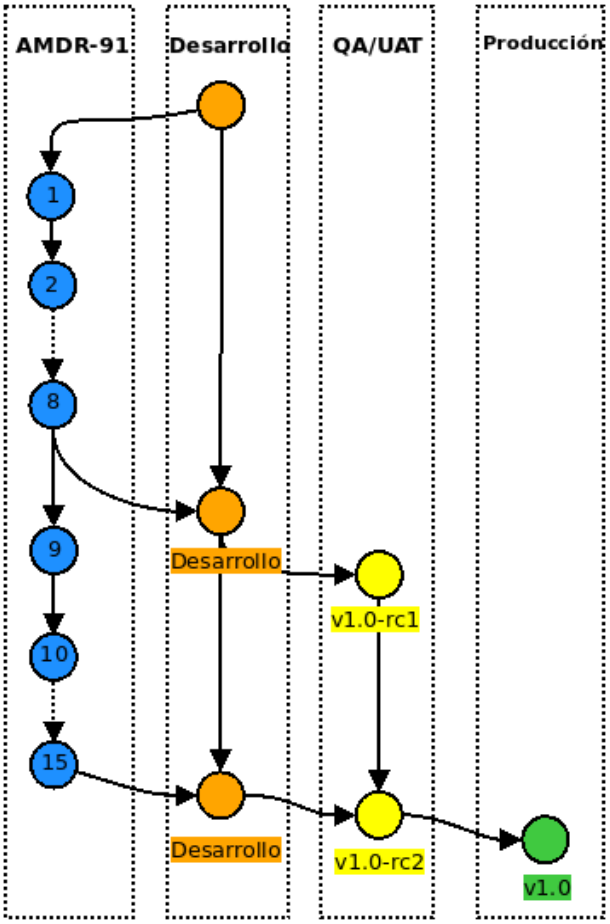

Figura 16: Representación gráfica para el proyecto 2. Este proyecto, tuvo 15 entregas con dos mezclas en desarrollo y dos revisiones en QA.

5.1.3 Proyecto 3

El tercer proyecto evolutivo duró dos semanas, y el cambio buscaba modificar los procesos de aumento y disminución de los cupos para poder generar una oferta a los clientes. Este proyecto evolutivo tuvo la particularidad de que transcurrieron 3 meses entre el término del desarrollo, y su puesta en producción. Esto ocurrió producto de problemas no relacionados directamente con el software en sí.

La puesta en producción fue igualmente sencilla, y bastó con recurrir a la última versión disponible, sin mayores vueltas ni revisiones. El cambio tomó solo unos minutos en completarse. En el escenario anterior, y dado el tiempo transcurrido entre el cierre del desarrollo y su puesta en producción, el equipo habría optado por volver a ejecutar las pruebas en QA, para asegurarse de que no hubieran ocurrido cambios, previo a copiar los componentes. El resumen de este proyecto se muestra en la tabla 13.

Tabla 13: Resumen del proyecto 3.

Código del proyecto	AMDR-108
Cantidad de entregas realizadas	4
Cantidad de fuentes intervenidas durante el cambio	41
Cantidad de versiones disponibles en QA	1
Situación final	Instalación en producción exitosa. La instalación tomó menos de un minuto.
Adherencia al ciclo de vida	Se pudo seguir sin problemas el flujo definido.
Independización de versiones de los servidores	Se definieron líneas base, y las versiones quedaron independientes de los repositorios de código en los servidores.
Automatización del despliegue	Los despliegues se hicieron correctamente.

La figura 17 describe los cambios que ocurrieron a nivel de la rama AMDR-108 el cual contempló 4 entregas, una mezcla en desarrollo y una revisión en QA. A nivel de QA, se puede ver como se incrementa la versión pasando de v1.0-rc2 a v1.1-rc1, dado que previamente ya existía en producción la versión v1.0. Al finalizar la certificación de la versión, la revisión v1.1-rc1 pasa a ser consolidada en la v1.1 en producción.

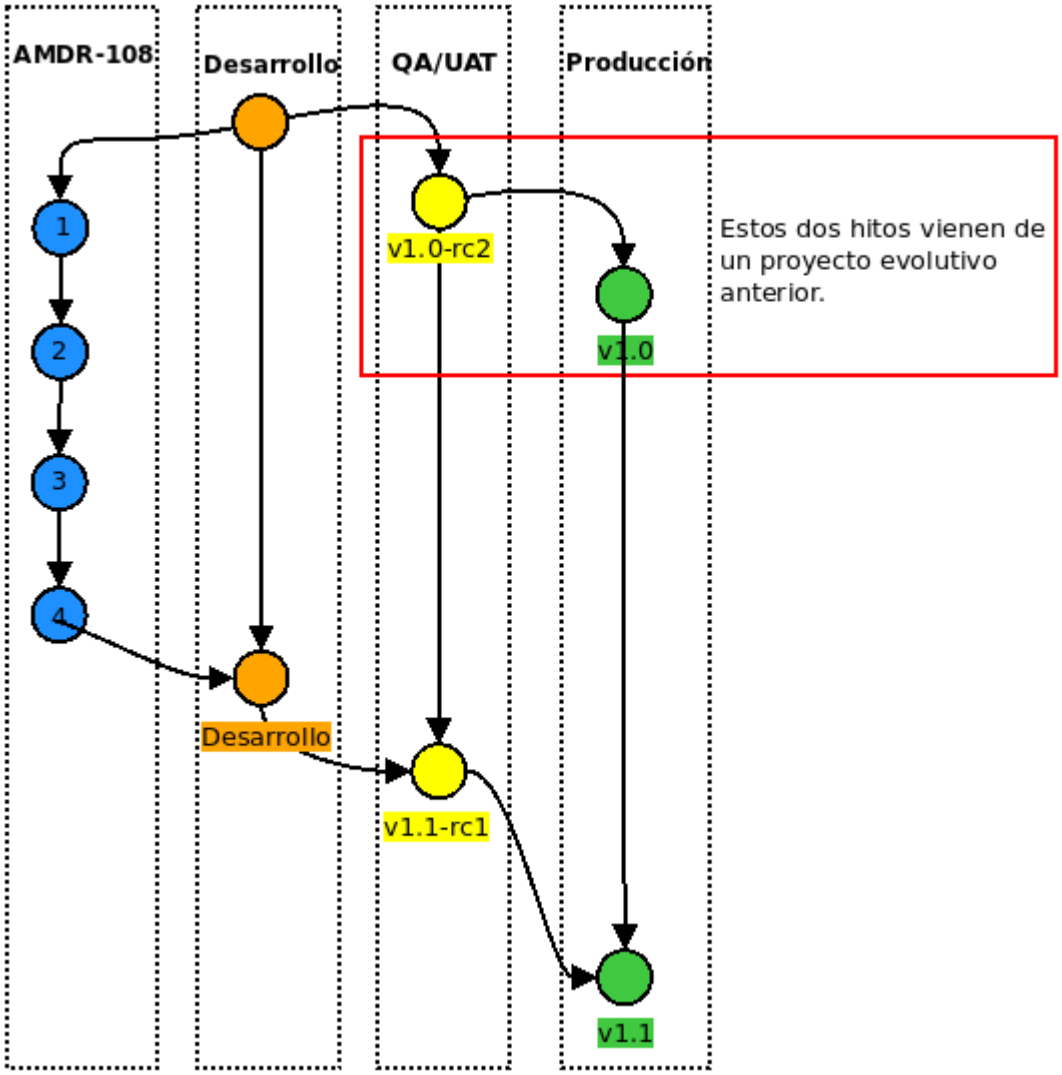

Figura 17: Representación gráfica para el proyecto 3. Este proyecto, tuvo 4 entregas. Notar cómo se incrementa la versión en QA y producción.

5.1.4 Proyecto 4

El cuarto proyecto partió como un evolutivo muy pequeño, que requería agregar una variable adicional a los cálculos que éste realizaba. Sin embargo, por limitaciones de las librerías usadas, se requirió actualizar una buena porción del software base (java, compilador, *runtime* y sistema operativo). Luego se actualizaron algunas definiciones de negocio, agregando nuevas funcionalidades, lo que alargó el proyecto e involucró muchas iteraciones.

Las nuevas definiciones requirieron ejecuciones con un gran volumen de datos, que hicieron necesario utilizar los servidores de certificación como si fueran de desarrollo. Realizar este control con la estrategia anterior habría sido caótico, debido a la gran cantidad de instancias

en las que se podían introducir errores. Al igual que en el proyecto 2, mientras se realizaban pruebas en el ambiente de QA, se seguían desarrollando nuevas funcionalidades en paralelo. El haber seguido trabajando de la forma anterior, habría complicado de sobremanera el haber usado los servidores con un propósito distinto. En ese caso, se debería haber creado respaldos manuales y realizado alguna gestión sobre los mismos. El resumen de este proyecto se presenta en la tabla 14.

Tabla 14: Resumen del proyecto 4.

Código del proyecto	AMDR-43
Cantidad de entregas realizadas	17
Cantidad de fuentes intervenidas durante el cambio	93
Cantidad de versiones disponibles en QA	4
Situación final	Proyecto evolutivo sigue en curso. Se han generado múltiples versiones para realizar la certificación de la calidad del sistema, la cual involucra el uso de un gran volumen de datos (>100 GB). En este proyecto se ha podido realizar comparaciones de resultados entre las distintas versiones, eliminando la incertidumbre relacionada a los componentes usados en el software. También se han podido replicar diversos problemas presentados en certificación de calidad, en ambientes de desarrollo con un volumen recortado de datos, pudiendo descartar problemas del desarrollo y relacionándolos con los datos usados.
Adherencia al ciclo de vida	Se siguió sin problemas el ciclo de vida definido, pero se identificó una situación no prevista; esta es, cómo hacer una corrección desde la rama de producción. El problema pudo ser corregido siguiendo el flujo normal, aunque existe una posibilidad de mejora de este aspecto en el proceso.
Independización de versiones de los servidores	Se definen líneas bases. Se pudo usar el servidor de QA (mejores prestaciones) con las ramas asociadas al proyecto.
Automatización del despliegue	Los despliegues se hicieron correctamente.

La representación gráfica para este proyecto es revisada junto con el proyecto 5, y mostrada en la figura 18.

5.1.5 Proyecto 5

El proyecto 5 está directamente relacionado con el proyecto anterior. Dado los problemas encontrados en una de las librerías, se requirió actualizar las versiones de software base (versión de java, compilador y *runtime*). Esto llevó a separar el proyecto en dos subproyectos: primero modificar el software base, y luego abordar los cambios funcionales.

La estrategia que se planteó para abordar este proyecto fue desarrollar ambos subproyectos en paralelo, pero trabajando de forma independiente uno del otro. El usar el proceso definido permitió ahorrar mucho tiempo, ya que en otras circunstancias, el proveedor habría tenido que desplazarse hasta las dependencias de la organización para solucionar el problema. Cuando el primer subproyecto fue terminado, incorporó naturalmente los cambios introducidos por el segundo subproyecto, permitiendo tener continuidad y a los dos equipos trabajando simultáneamente. El resumen de este proyecto se muestra en la tabla 15.

Tabla 15: Resumen del proyecto 5.

Código del proyecto	AMDR-61
Cantidad de entregas realizadas	17
Cantidad de fuentes intervenidas durante el cambio	642
Cantidad de versiones disponibles en QA	2
Situación final	La cantidad de componentes modificados contabiliza también a los eliminados y renombrados. En este proyecto evolutivo participó por primera vez un proveedor externo. La comunicación habitual fue complementada con los cambios que refleja el repositorio de código, eliminando el envío de componentes a través de correo electrónico. En particular, este proyecto actualizó componentes que requerían actualizar productos base (compiladores y *runtime*), por ende esta gestión minimizó la cantidad de errores.
Adherencia al ciclo de vida	Se pudo seguir sin problemas el flujo definido. Sin embargo, a la hora de mezclar las entregas en el ambiente de desarrollo, hubo un componente que tenía dos versiones distintas. En ese caso hubo que determinar, mediante comunicación con los desarrolladores, con cuál versión quedarse como definitiva.
Independización de versiones de los servidores	Dada la actualización de los servidores, estos tuvieron que ser reinstalados teniendo la tranquilidad de contar con un repositorio que contenía cada una de las versiones.

Automatización del despliegue	Los despliegues se hicieron correctamente.

Como se mencionó antes, luego de realizar la división en subproyectos (es decir, primero modificar el software base, y luego abordar los cambios funcionales), el proyecto 5 (AMDR-61) sienta las bases para que el proyecto 4 (AMDR-43) pueda comenzar a trabajar. La entrega 10 se mezcla con la 4 en desarrollo, y dan paso a la versión v1.0-rc1. Luego, se van produciendo sucesivamente las distintas versiones hasta llegar a la versión de producción final tal como se puede apreciar en la figura 18.

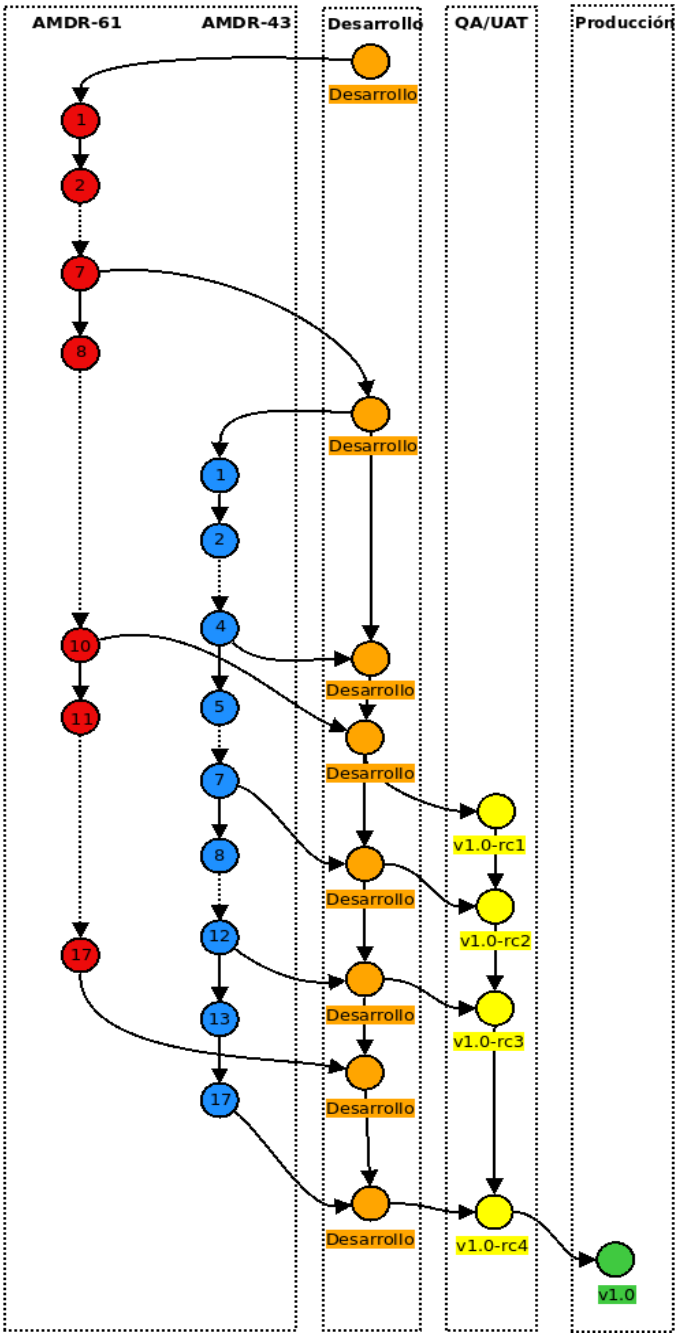

Figura 18: Representación gráfica para proyectos 4 y 5, ejecutado con un alto nivel de superposición.

5.2 Proyecto desde cero

A continuación se resume la ejecución a la fecha de un proyecto iniciando desde cero, en donde se aplicó el proceso propuesto en esta tesis. Dicho proyecto ha sido implementado usando iteraciones de 2 semanas de duración, por lo tanto, al final de cada iteración se tiene una nueva versión para realizar pruebas, y una certificación de calidad más acotada. Se han registrado los cambios para las ocho iteraciones que han generado componentes para este sistema. El resumen de este proyecto está en la tabla 16.

Tabla 16: Resumen del proyecto 6.

Código del proyecto	Se utilizó el nombre del proyecto, más el número del *sprint* para cada set de cambios. No se indica el nombre del proyecto por un tema de confidencialidad, pues este es identificable para un lector externo.
Cantidad de entregas realizadas	56
Cantidad de fuentes intervenidas durante el cambio	1396
Cantidad de versiones disponibles en QA	8
Situación final	En desarrollo
Adherencia al ciclo de vida	Se pudo seguir sin problemas el flujo definido.
Independización de versiones de los servidores	Al ser un proyecto nuevo, éste inició siempre desde los repositorios. Incluso los servidores de certificación y producción, estuvieron disponibles después de haber llegado a etapas de certificación (eran pruebas tempranas y no requerían capacidades especiales). Por ende, los hitos de prueba fueron registrados en los repositorios.
Automatización del despliegue	Los despliegues se hicieron correctamente.

En las figuras 19 y 20 se gráfica como cada *sprint* va complementando el desarrollo generando incrementos parciales sobre las ramas de desarrollo y distintas revisiones sobre la de QA. Para los datos considerados, todavía no se crea una versión final para dejar en producción.

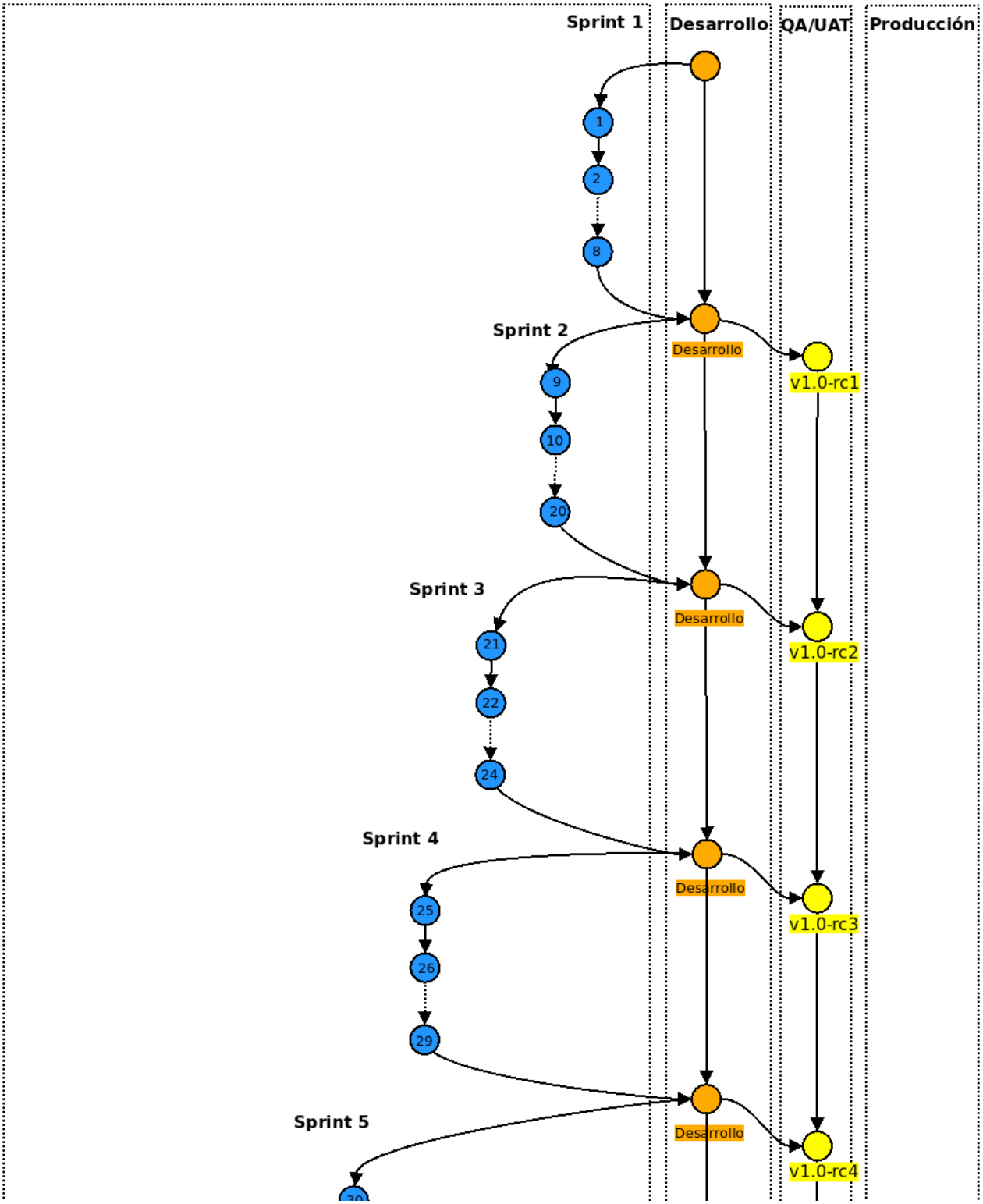

Figura 19: Representación gráfica para proyecto desde cero, parte 1. Cada sprint va complementando el desarrollo.

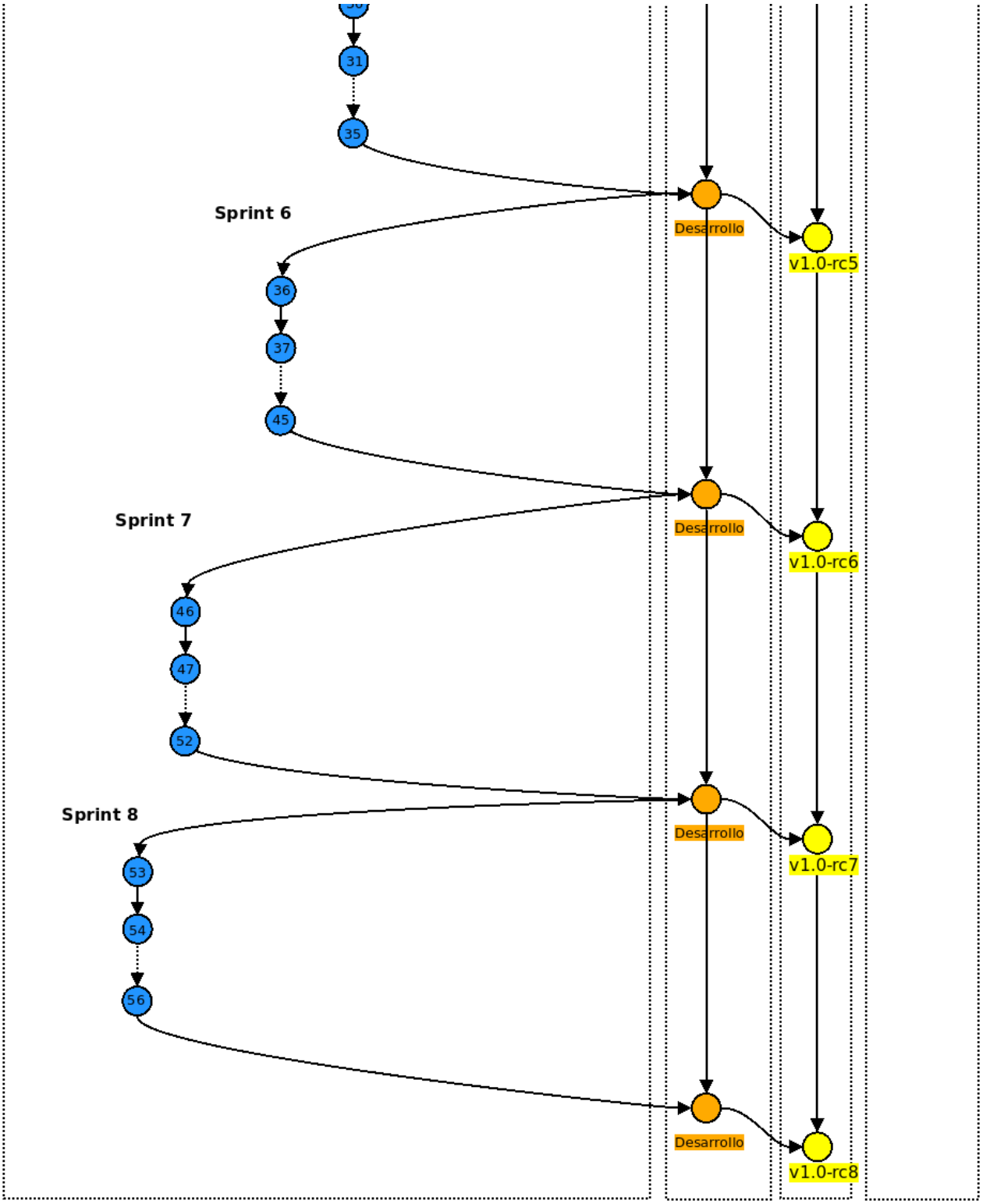

Figura 20: Representación gráfica para proyecto desde cero, parte 2.

Como se indica en la tabla 16, este sistema todavía se encuentra en desarrollo. A pesar de que aún no se ha concluido, el haber estructurado el proyecto considerando el proceso propuesto ha permitido realizar diversas pruebas, tanto a nivel funcional como a nivel técnico. Por ejemplo, para una de las características de este proyecto, fue necesario tener certeza respecto a los tiempos que puede tomar el procesamiento de grandes volúmenes de datos. Para ello, se efectuaron múltiples pruebas de rendimiento en distintos servidores, con diferentes prestaciones y validaciones técnicas, contando con la seguridad de comparar siempre las mismas versiones del software.

Por otra parte, en este proyecto han participado varias personas, y el intercambio de componentes se ha realizado sólo a través de los repositorios, eliminando el envío de componentes a través de correo electrónico, como se hacía antes.

La instalación de versiones en los distintos servidores se ha facilitado, evitando el trabajo manual sobre los mismos. A modo de prueba, el proveedor ha realizado modificaciones en los servidores, y dichos cambios han sido fácilmente advertidos, los cuales prontamente han sido regularizados.

5.3 Discusión

A partir del uso del proceso propuesto en los seis proyectos reportados, se puede ver que dicho proceso permite mantener un control detallado de los componentes modificados, estableciendo una relación entre las versiones al momento de pasar de una etapa a otra. Además, hoy día el sistema es auditable, no en base a la documentación que habitualmente se genera, sino que a través del *tracking* de los componentes modificados. Tal como se puede visualizar en las imágenes, además se cuenta una referencia gráfica y fácil de entender respecto a lo que ocurrió en el desarrollo de un proyecto.

Por otra parte, fue posible trabajar en más de un proyecto (o subproyecto) en paralelo, agilizando así los desarrollos, teniendo primero un proyecto que entrega las bases, para que el segundo pueda avanzar y al final ser el responsable de consolidar los cambios. En este caso hubo que agregar una instancia de mezcla (integración) de componentes. El trabajo de integración fue resuelto en su gran mayoría por la herramienta, pero en las situaciones donde hubo conflictos, los desarrolladores debieron comunicarse para consolidar la mezcla. Esta etapa no estaba contemplada originalmente en la propuesta, pero fue correctamente cubierta por las herramientas y la metodología.

El hecho de independizar el código, de los servidores donde éste se encuentra alojado, fue sumamente útil para realizar pruebas. Esto permitió, por ejemplo, utilizar servidores de certificación (producto de las prestaciones y capacidades del mismo) con versiones de

desarrollo. Luego, al ir cerrando (dejando lista) cada una de las versiones, se obtiene certeza respecto a lo que se está instalando.

Por otro lado, en uno de los proyectos ocurrió una situación puntual, donde se planteó realizar la corrección al sistema directamente en el servidor, y luego regularizar. Sin embargo, se optó por el "camino largo" que implicó crear un proyecto nuevo, y realizar la corrección pasando por todas las etapas, hasta llegar a la final. Esto último abre la opción para agregarle una etapa nueva al proceso, que permitiría incluir estos cambios pequeños de manera expedita sin tener que caer en ninguno de los dos extremos antes mencionados.

La forma de desplegar los sistemas permitió igualar el proceso independiente del ambiente en donde se estaba ejecutando. Al eliminar las tareas manuales, ya no existieron dudas respecto a la instalación de los componentes y se eliminaron los errores a la hora de instalarlos.

En contraste con lo que ocurría antes, el formulario o pauta de instalación fue reducido a un documento que solo quedó para fines de auditoría y hoy solo explicita la versión que se instala, a diferencia del largo y tedioso detalle de componentes modificados que planteaba antes. Apoyado en la herramienta de versionamiento y la visualización que se puede obtener de los componentes desarrollados, los auditores pueden revisar el detalle de los componentes modificados.

6 Conclusiones y Trabajo a Futuro

Este trabajo de tesis buscó abordar el proceso de gestión de la configuración de sistemas del área de "banca" de un holding latinoamericano. Esta empresa utiliza sistemas grandes y complejos, que necesitan adaptarse periódicamente para satisfacer las necesidades de las filiales de cada país. El proceso de gestión de la configuración de estos sistemas fue identificado como crítico, y como plataforma de apoyo a la evolución de la organización de cara al futuro y a la competencia.

Dicho proceso contaba con una alta tasa de actividades manuales, y en consecuencia, era dependiente de las personas que lo llevaban a cabo. Por lo tanto, era lento, costoso, propenso a errores, y frecuentemente generaba sobretrabajo. Además, el proceso funcionaba diferente dependiendo de cada operador.

Para mejorar la gestión de los componentes de estos sistemas se definió un nuevo proceso de gestión de la configuración, más previsible y sistemático. Tener un flujo de proceso conocido facilita mucho la comunicación respecto a qué hacer con los componentes entregados, eliminando pautas de instalación, tareas manuales y diferencias en la gestión de componentes dependiendo de quien aborde el proyecto. Realizando un mismo tipo de gestión, se puede abordar múltiples proyectos descansando en el proceso. De esta forma, todos los cambios se enfrentan de la misma forma, y se deja de depender del orden que un jefe de proyecto pueda ofrecer, o del caos que un desarrollador pueda introducir. Este proceso establece las bases para el ciclo de vida que debe seguir el programador respecto al software que se está desarrollando. Además, establece cómo proceder para que sus cambios lleguen a una versión final.

Al tener flexibilidad en la gestión de los componentes, el usuario final (de negocio) logra plantear distintos escenarios a la hora de realizar pruebas, permitiendo descubrir nuevas funcionalidades y/o errores tempranamente. El involucramiento del usuario ha sido inesperado pero fructífero, y ha despertado la ambición del mismo. Esto último ha sido principalmente por la facilidad y rapidez con que se pueden desplegar los cambios en los distintos servidores, y la certeza que se tiene respecto a cuáles son los componentes a desplegar (ya no se tiene dudas respecto a las versiones de los mismos). Al desacoplar las versiones del software, respecto a los servidores donde ellas se ejecutan, nos ha permitido referirnos a la versión del software y no al servidor como elemento de referencia.

Usando este proceso se completaron 5 proyectos evolutivos, de distinto tamaño y duración, y un desarrollo desde cero. En lo que respecta a la gestión de los componentes, la experiencia en el uso del proceso propuesto fue muy similar; particularmente, en ningún caso el foco de los errores estuvo asociado a confusiones o errores en las versiones de los componentes

instalados en los servidores. Una cosa a destacar en el desarrollo desde cero, es que el proveedor completó todas las entregas a través de los repositorios, cosa que en desarrollos anteriores fue casi imposible de realizar, pues involucraba un trabajo tedioso, tanto para él (a la hora de generar pautas de instalación), como para el equipo (a la hora de seguirlas).

Como resultado del uso del proceso propuesto, la instalación de componentes en ambientes productivos se ha simplificado de manera significativa. La acción de instalar componentes ha eliminado la incertidumbre y la posibilidad de cometer errores propios de estas actividades. Si bien dentro de los proyectos registrados no se presentaron problemas donde se requería volver a la versión anterior, en los ejercicios con los operadores se completó una vuelta atrás, la cual fue igual de sencilla que el paso a producción.

Actualmente se logra hacer muchas más pruebas, en menor tiempo que antes, lo que empieza a generar un poco de desorden, principalmente con las pruebas a nivel de desarrollo. Por lo tanto, como trabajo a futuro se abordará la gestión de las pruebas realizadas a nivel de desarrollo, para ir enlazándolas y registrándolas. De esa manera se podrá entregar una base de pruebas ya realizadas a nivel de desarrollo, que el equipo de QA no deba construir desde cero.

Como parte del trabajo a futuro también es necesario formalizar la comunicación entre los desarrolladores, particularmente cuando existen conflictos con algunos componentes. Si bien la herramienta de gestión de código resuelve la mayoría de estos conflictos, existen casos donde una persona debe decidir qué componente (o mezcla de ellos) se debe mantener.

Aprovechando algunas de las sistematizaciones, se pueden incorporar herramientas que hagan chequeo y revisión de código, para así generar indicadores de calidad sobre las entregas y eventualmente empezar a resolver la deuda técnica asociada al código, como por ejemplo, falta de documentación del mismo, la existencia de rutinas no utilizadas, las variables inicializadas y no usadas, etc. El objetivo de estos indicadores podría ser que, en función de la cobertura de estos, los procesos de instalación se hagan directamente sin intervención de los operadores.

Bibliografía

Preston-Werner, T., 2013. Versionado Semántico 2.0.0-Rc.2. [online] Semantic Versioning. Disponible en: <https://semver.org/lang/es/> [Último acceso: 2 de Julio de 2021].

Fowler, M., 2006. Continuous Integration. [online] martinfowler.com. Disponible en: <https://martinfowler.com/articles/continuousIntegration.html> [Último acceso: 2 de Julio de 2021].

Sommerville, I., 2002. Ingeniería de Software. 6th ed. pp.640-660.

Pressman, R., 2005. Ingeniería de Software. Un Enfoque Práctico. 6th ed. pp.796-814.

DeKrey, M., 2017. Git: How I Use It And Why I Don'T Use Gitflow. [online] Medium. Disponible en: <https://medium.com/@matt.dekrey/git-how-i-use-it-and-why-i-dont-use-gitflow-8688f255fef2> [2 de Julio de 2021].

Collins-Sussman, B., Fitzpatrick, B., & Pilato, C. (2004). Control de versiones con Subversion (1st ed.). pp.1-14.

CA Technologies, 2014. CA Harvest Software Change Manager Workbench User Guide. Release 12.6. pp. 15-23.

Microfocus, 2019. Dimensions CM Getting Started Guide Product version 14.5.1.
Serban A.,2007. Visual SourceSafe 2005 Software Configuration Management in Practice. pp.237-248.

Microsoft, 2007. Visual SourceSafe. Disponible en <https://docs.microsoft.com/en-us/previous-versions/ms181038(v=vs.80)> [Último acceso: 2 de Julio de 2021].

Chacon, S. & Straub, B. Pro Git (2nd ed.). pp 30-33.

Travis CI, Disponible en <https://en.wikipedia.org/wiki/Travis_CI> [Último acceso: 2 de Julio de 2021].

Getting started with codeship pro part 1. Disponible en <https://documentation.codeship.com/pro/quickstart/getting-started/#getting-started-with-codeship-pro-part-1> [Último acceso: 2 de Julio de 2021].

Overview Circle CI. Disponible en <https://circleci.com/docs/2.0/about-circleci/> [Último acceso: 2 de Julio de 2021].

Jenkins. Disponible en <https://es.wikipedia.org/wiki/Jenkins> [Último acceso: 2 de Julio de 2021].
Launchpad. Disponible en <https://es.wikipedia.org/wiki/Launchpad> [Último acceso: 2 de Julio de 2021].

Features – The right tools for the job – GitHub. Disponible en <https://github.com/features> [Último acceso: 2 de Julio de 2021].

Features | GitLab. Disponible en <https://about.gitlab.com/features> [Último acceso: 2 de Julio de 2021].

Anexo A Ponderación de herramientas

A continuación se presentan las herramientas evaluadas para implementar el proceso propuesto, y los valores obtenidos por cada una en los ítems considerados. Cabe recordar que, tal como se indicó en el capítulo 4, la puntuación asignada fue discutida y acordada con el equipo de trabajo, con el objetivo de llegar a un valor de consenso, y de esa manera poder tomarlo como referencia.

Jenkins + Bitbucket

Tabla 17: Ponderación para producto Jenkins + Bitbucket.

Atributo	Factor de cobertura	Dimensión	Peso de la Dimensión	Factor	Valor
Seguridad	1	Técnica	0,3	0,1	0,03
Documentación	1	Técnica	0,3	0,2	0,06
Soporte	0,85	Técnica	0,3	0,2	0,051
Extensibilidad	0,85	Técnica	0,3	0,2	0,051
S.O. soportados	1	Técnica	0,3	0,1	0,03
Adaptabilidad	1	Técnica	0,3	0,2	0,06
Flujo definido	1	Funcional	0,4	0,3	0,12
Identificar versiones	1	Funcional	0,4	0,2	0,08
Empaquetar solución	0,75	Funcional	0,4	0,15	0,045
Desarrollo paralelo	1	Funcional	0,4	0,15	0,06
Automatizar despliegue	0,75	Funcional	0,4	0,2	0,06
Experiencia anterior	0,5	Experiencia	0,3	0,4	0,06
Curva de aprendizaje	0,75	Experiencia	0,3	0,25	0,056
Contactos usando la tecnología	0,85	Experiencia	0,3	0,35	0,089
				Total	**0,852**

Codeship + Bitbucket

Tabla 18: Ponderación para producto Codeship + Bitbucket.

Atributo	Factor cobertura	Dimensión	Peso de la Dimensión	Factor	Valor
Seguridad	1	Técnica	0,3	0,1	0,03
Documentación	0,85	Técnica	0,3	0,2	0,05
Soporte	0,5	Técnica	0,3	0,2	0,03
Extensibilidad	0,75	Técnica	0,3	0,2	0,05
S.O. soportados	1	Técnica	0,3	0,1	0,03
Adaptabilidad	0,5	Técnica	0,3	0,2	0,03
Flujo definido	1	Funcional	0,4	0,3	0,12
Identificar versiones	1	Funcional	0,4	0,2	0,08
Empaquetar solución	0,25	Funcional	0,4	0,15	0,02
Desarrollo paralelo	1	Funcional	0,4	0,15	0,06
Automatizar despliegue	0	Funcional	0,4	0,2	0
Experiencia anterior	0,25	Experiencia	0,3	0,4	0,03
Curva de aprendizaje	0,5	Experiencia	0,3	0,25	0,04
Contactos usando la tecnología	0	Experiencia	0,3	0,35	0
				Total	0,558

GitLab

Tabla 19: Ponderación para producto GitLab.

Atributo	Factor cobertura	Dimensión	Peso de la Dimensión	Factor	Valor
Seguridad	1	Técnica	0,3	0,1	0,03
Documentación	1	Técnica	0,3	0,2	0,06
Soporte	0,85	Técnica	0,3	0,2	0,05
Extensibilidad	0,75	Técnica	0,3	0,2	0,05
S.O. soportados	1	Técnica	0,3	0,1	0,03
Adaptabilidad	1	Técnica	0,3	0,2	0,06
Flujo definido	1	Funcional	0,4	0,3	0,12
Identificar versiones	1	Funcional	0,4	0,2	0,08
Empaquetar solución	0,85	Funcional	0,4	0,15	0,05
Desarrollo paralelo	1	Funcional	0,4	0,15	0,06
Automatizar despliegue	0,85	Funcional	0,4	0,2	0,07
Experiencia anterior	0,85	Experiencia	0,3	0,4	0,1
Curva de aprendizaje	0,85	Experiencia	0,3	0,25	0,06
Contactos usando la tecnología	1	Experiencia	0,3	0,35	0,11
				Total	0,925

GitHub

Tabla 20: Ponderación para producto Jenkins + Bitbucket.

Atributo	Factor cobertura	Dimensión	Peso de la Dimensión	Factor	Valor
Seguridad	1	Técnica	0,3	0,1	0,03
Documentación	1	Técnica	0,3	0,2	0,06
Soporte	0,85	Técnica	0,3	0,2	0,05
Extensibilidad	0,75	Técnica	0,3	0,2	0,05
S.O. soportados	1	Técnica	0,3	0,1	0,03
Adaptabilidad	0	Técnica	0,3	0,2	0
Flujo definido	1	Funcional	0,4	0,3	0,12
Identificar versiones	1	Funcional	0,4	0,2	0,08
Empaquetar solución	0,75	Funcional	0,4	0,15	0,05
Desarrollo paralelo	1	Funcional	0,4	0,15	0,06
Automatizar despliegue	0,75	Funcional	0,4	0,2	0,06
Experiencia anterior	0,25	Experiencia	0,3	0,4	0,03
Curva de aprendizaje	0,85	Experiencia	0,3	0,25	0,06
Contactos usando la tecnología	0,5	Experiencia	0,3	0,35	0,05
				Total	0,727

Solución inhouse (SVN)

Tabla 21: Ponderación para solución inhouse (SVN).

Atributo	Factor cobertura	Dimensión	Peso de la Dimensión	Factor	Valor
Seguridad	0,5	Técnica	0,3	0,1	0,02
Documentación	0	Técnica	0,3	0,2	0
Soporte	0,85	Técnica	0,3	0,2	0,05
Extensibilidad	1	Técnica	0,3	0,2	0,06
S.O. soportados	1	Técnica	0,3	0,1	0,03
Adaptabilidad	0,25	Técnica	0,3	0,2	0,02
Flujo definido	0	Funcional	0,4	0,3	0
Identificar versiones	0,25	Funcional	0,4	0,2	0,02
Empaquetar solución	0	Funcional	0,4	0,15	0
Desarrollo paralelo	0	Funcional	0,4	0,15	0
Automatizar despliegue	0,75	Funcional	0,4	0,2	0,06
Experiencia anterior	0,85	Experiencia	0,3	0,4	0,1
Curva de aprendizaje	0,25	Experiencia	0,3	0,25	0,02
Contactos usando la tecnología	0,85	Experiencia	0,3	0,35	0,09
				Total	0,461