Planificación y gestión de las herramientas de SCM aplicadas a un proyecto de desarrollo

Memoria del proyecto
de Ingeniería Informática
Realizado por:
Robert Garcia Atserias
Y dirigido por:
Xavier Binefa Valls

El sotasignat, ...

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en

I per tal que consti firma la present.

Signat: .....
Bellaterra, ...de.....de 200.....

Agradecimientos:

Teniendo en cuenta que me encuentro al final de un ciclo académico, sería injusto dedicar mis agradecimientos solo a las personas que me han ayudado en la elaboración de este proyecto, dicho esto debo agradecer **especialmente** el haber llegado donde he llegado a **mis padres y hermano**, puesto que sin ellos seguro que todo esto no hubiera sido posible.

Agradezco también a mi primo **Carlos Garcia**, toda la ayuda académica y extra-académica que me ha aportado estos últimos años.

A **Cristina Vélez** le debo los ánimos aportados en la fase final de mi carrera, sin ella tampoco hubiera sido posible.

Nunca podré estar suficientemente agradecido a **Ricard Burriel**, quien me ha aportado todos los conocimientos que tengo en este sector.

Y a muchos otros que no menciono, pero no por ello menos importantes:

Muchas Gracias

INDICE

1. Introducción:

1.1. Introducción y objetivos del proyecto:

El desarrollo de un producto de software, es cada vez más complejo, el tamaño del código es mayor, esto hace que el producto sea por un lado más inestable, puesto que cuantas más líneas de código contenga, más errores habrá en este, por otro lado, el código se hace menos trazable, y más inconsistente. El número de desarrolladores aumenta, hecho que conlleva a un código con redundancias si no hay una buena comunicación entre los desarrolladores, por otro lado, el desarrollo distribuido, es cada vez más usual, y esto requiere a su vez, una correcta sincronización entre todos los centros que desarrollen un mismo proyecto.

Por todo esto, es fundamental tener claro el papel que desempeñará el SCM (Software Configuration Management), dentro del marco de nuestro proyecto.

PRINCIPIOS DE SCM:

SCM es el encargado de controlar las modificaciones efectuadas en el software, con el fin de garantizar una correcta trazabilidad del proyecto, y un fácil mantenimiento del mismo.

Cuando hablamos de controlar los cambios en el desarrollo del software, nos referimos a conocer: quien ha realizado cada modificación, que modificación ha realizado, y porqué la ha realizado. Gracias al conocimiento de todo esto, podemos obtener mejores versiones del producto, y por supuesto, cuantas más versiones tengamos de un producto, mejor será el resultado final del mismo.

Nuestro proyecto tratará de asegurar el correcto cumplimiento del proceso de desarrollo de SW, por tanto, antes de conocer que se puede mejorar, vamos a tener una visión global de las partes que forman este desarrollo.

1.1.1. Fase de diseño

La fase de diseño de un proyecto es aquella en que se toman las principales decisiones en cuanto a diseño del mismo, herramientas a utilizar (Control de versiones del SW, control de versiones de Documentos, bases de datos de actividades y defectos, entorno de desarrollo…), y políticas que definen como usar estas herramientas.

Todas estas decisiones afectarán a todos los desarrolladores, y por tanto, deben estar debidamente documentadas. Ésta documentación a su vez permanecerá almacenada, y debidamente versionada, bajo un control de versiones específico para documentos. Por otro lado, mientras el proyecto avance, y se vaya creando nueva documentación, ésta será adjuntada en el mismo directorio, dentro del dicho control de versiones.

En esta primera fase del proyecto, las tareas a desarrollar serán debidamente definidas, una vez hecho esto, dichas actividades serán insertadas en una base de datos, cuya única funcionalidad será la de tener el estado de las actividades, junto a la información relacionada con éstas, bajo un control debidamente versionado, y teniendo en cualquier momento acceso a la información en cuanto al estado de la actividad se refiere.

A la vez que estas actividades son definidas, se van asignando a los correspondientes desarrolladores, esto nos permitirá fácilmente reasignar actividades en cualquier momento del proyecto, así como consultar el trabajo de cada desarrollador, la sobrecarga de éstos, así como tener una visión global del estado del proyecto.

Para finalizar con las herramientas necesarias para desarrollar un proyecto, no podemos dejar atrás el control de versiones del SW.

En la fase de diseño, no únicamente nos quedaremos definiendo la herramienta a utilizar, sino que también debemos definir las políticas que deberán seguir los desarrolladores a lo largo del período de implementación del proyecto.

1.1.2. Fase de implementación

En esta fase del proyecto, el control de versiones del SW, debe tener claramente definidas las políticas a seguir por parte de los desarrolladores, mientras que la base de datos de actividades y el control de versiones de la documentación seguirán recibiendo nuevas entradas.

Debido a esto, podemos llegar a la conclusión que es en la fase de desarrollo, en la que las aplicaciones anteriormente mencionadas, toman mayor importancia, implicando esto, que es la fase más susceptible a errores humanos, y a su vez, la fase en la que mayor claridad debemos tener.

Es por esto, que para garantizar esta trazabilidad, y evitar los errores humanos, debemos garantizar el correcto procedimiento en la utilización de dichas herramientas.

1.1.3. Fase de corrección de defectos

Una vez el proyecto esta desarrollado, se entrega al equipo de testeo, encargado de detectar todos los errores de código o diseño, que provocan un incorrecto funcionamiento del producto.

Este equipo, una vez detecta un error, lo introduce en la base de datos de defectos, posteriormente, el líder del proyecto, se encarga de asignar el defecto a algún desarrollador, que a su vez lo corregirá para una posterior versión del mismo.

Si tomamos un defecto como si fuera una actividad a desarrollar, nos encontramos de nuevo en la fase de desarrollo, y por tanto, debemos tener en cuenta la importancia que tiene que éstas correcciones sean de nuevo debidamente versionadas.

Por tanto, en esta fase, es también sumamente importante la utilización de una serie de herramientas que nos garanticen trazabilidad en el código, y nos minimicen al máximo la introducción de nuevos errores.

1.2. Objetivos y plan de la memoria

Una vez tenemos una visión global del proceso de creación de una aplicación, vamos a definir los objetivos de nuestro proyecto.

Dada la importancia que hemos observado que tiene mantener el código estable, trazable, y por tanto rigurosamente versionado y documentado, vamos a definir una serie de políticas que nos lo garanticen, y una serie de herramientas que nos asegurarán el cumplimiento del proceso.

El primer punto a tener claro es, por tanto, decidir que herramientas deberemos utilizar, y según estas herramientas, definiremos las diferentes políticas a seguir dentro de dichas herramientas. Esto lo veremos en el punto 2 de esta memoria.

Una vez definamos las herramientas a utilizar, lo siguiente que deberíamos hacer és asegurar el cumplimiento de estas políticas.

En un proceso donde el número de desarrolladores es elevado, y su ubicación dispar, cualquier error puede tener graves repercusiones, de ahí que debamos enfatizar en este punto.

Con la finalidad de garantizar que el proceso es robusto, definiremos una serie de herramientas, con las que el desarrollador deberá interactuar, de este modo, al no actuar directamente sobre las bases de datos, controles de versiones, etc. se reducen notablemente los posibles errores humanos introducidos por los desarrolladores.

Éste apartado constituirá el grueso de nuestro proyecto, el diseño de las herramientas así como su información desarrollada, serán ampliamente explicados en el tercer apartado de esta misma memoria.

Finalmente, y no por ello menos importante, nos encontramos con la información relacionada con el desarrollo de un proyecto, y con los informes de la evolución del mismo, en ellos, la información debe ser clara, concisa, y por supuesto, sin errores, para ello también se ha desarrollado una herramienta específica, que será explicada también en el tercer apartado de esta memoria, junto con las demás herramientas de automatización.

Dicho esto, podemos observar que nuestro objetivo es realizar una herramienta, que haciendo de nexo, tal y como se muestra en la siguiente figura, entre todas estas aplicaciones y el desarrollador, realice de forma automática las operaciones que debería realizar manualmente el desarrollador.

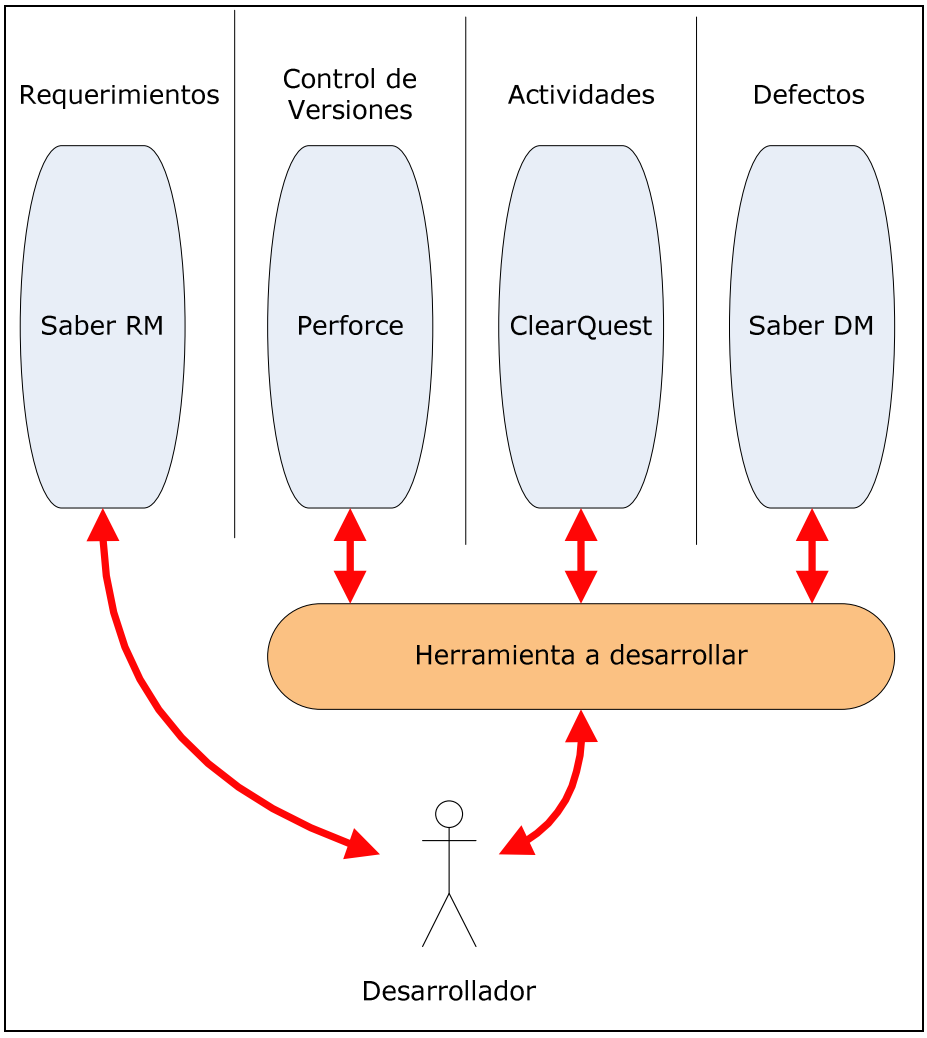

Figura 1

Entorno de la aplicación

Como podemos observar en la figura anterior, no hemos considerado el hecho que la herramienta opere directamente con la base de datos de Requerimientos, pues no se puede definir implícitamente un automatismo para ella, considerando que la frecuencia de acceso a esta base de datos no es la misma para todas las actividades, ni en todos los momentos, es decir, no podemos determinar en ningún caso, los documentos que debe consultar el desarrollador para implementar una determinada actividad, es por esto que las consultas a esta base de datos las hará el usuario de forma manual.

1.3. Definiciones y abreviaciones:

Término	Descripción
SCM	Software Configuration Management
UCM	Unified Change Management
Saber DM	Base de datos de Defectos
Saber RM	Base de datos de Requerimientos
ClearQuest	Base de datos de Actividades
Perforce	Control de versiones del proyecto
Acti Time	Base de datos para controlar el tiempo de las actividades
Share Point	Base de datos de documentos
Baseline	Versión de código etiquetada.
Release	Versión de código lista para entregar
Peer Reviewer	Persona asignada para revisar la implementación de un desarrollador.

2. Fundamentos teóricos y herramientas utilizadas.

En este apartado hablaremos de las herramientas utilizadas en el desarrollo del proyecto y de los requerimientos que debemos tener en cuenta en el momento de su elección, y a su vez, de cómo automatizar los procesos de interacción entre ellas, objetivo principal de este proyecto.

2.1. Base de datos de las actividades:

Una mala comunicación entre los desarrolladores, una mala coordinación, o bien la falta de una debida priorización en las tareas, puede concluir en que se detecten redundancias o bien información inconsistente en la entrega de un proyecto.

Utilizando una base de datos para almacenar las actividades a desarrollar, y su estado, podemos tener acceso a de forma clara y rápida a la información, permitiendo corregir los anteriormente mencionados errores, y facilitándonos además la toma de decisiones de una manera clara y rápida.

Gracias a la utilización de esta herramienta, podemos, en cualquier momento del proyecto conocer el estado de las actividades desarrolladas, y las que están todavía pendientes de implementar, documentar o diseñar.

En definitiva, usar una base de datos específica para las actividades, ayuda a la creación del proyecto eliminando los errores que ocurrirían manejando toda esta información de forma manual, mejora la comunicación entre los desarrolladores, mejora la calidad del software, y se puede tener una visión del proyecto en cualquier estado de tiempo.

En nuestro caso, como base de datos de actividades, hemos utilizado la herramienta de IBM, Rational ClearQuest.

Esta herramienta permite la creación de un esquema asociado a cada proyecto, pues si bien los requerimientos para un proyecto no son los mismos que para otro, la información de sus actividades también varía.

Por otro lado, esta herramienta permite que los desarrolladores hagan sus consultas personales, así como al administrador poner queries predefinidas, que serán vistas por todos los desarrolladores asociados al proyecto.

La aplicación estará almacenada en un servidor, al que todos los desarrolladores tendrán acceso, y las restricciones vendrán dadas por los propios grupos creados dentro de la misma aplicación.

A continuación podemos ver un ejemplo de esta base de datos:

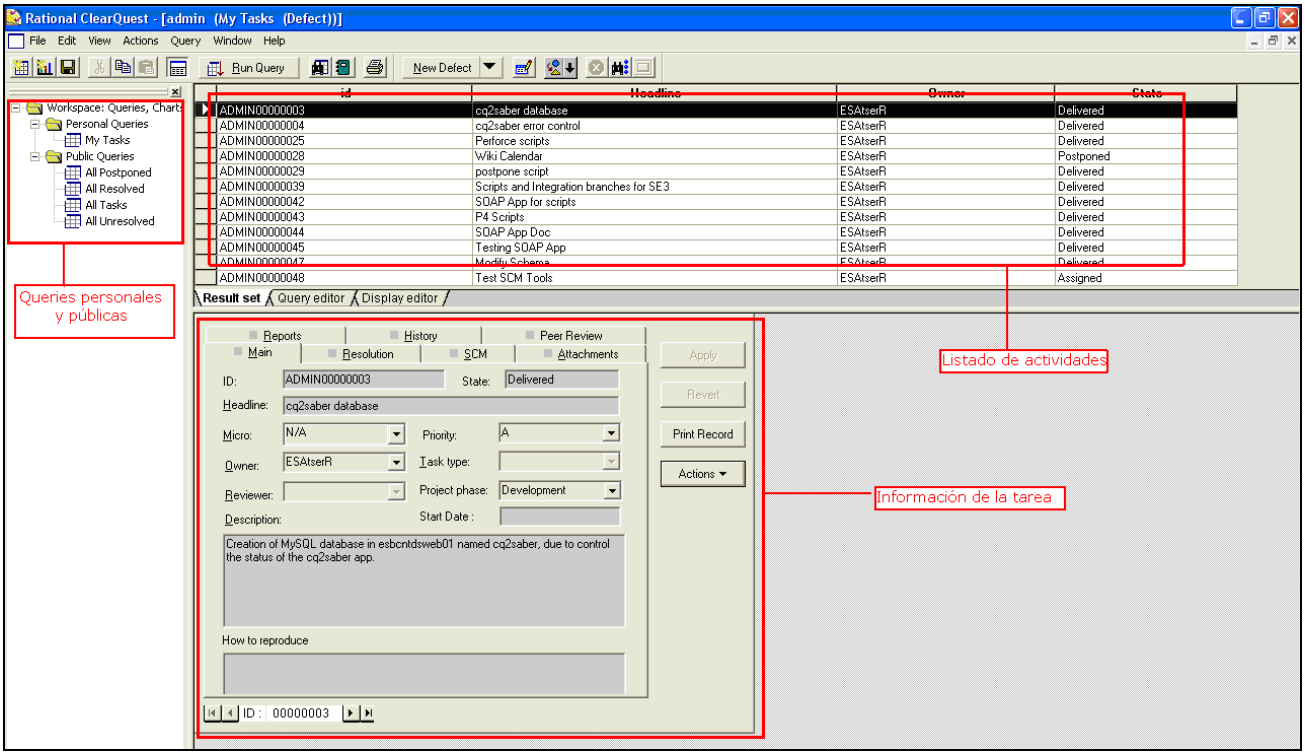

Figura 2

Entorno Rational ClearQuest

Como se puede observar en la figura anterior, esta aplicación nos aporta un robusto seguimiento del estado de las actividades, y de su información asociada.

Toda esta información está claramente ampliada en el anexo 1, correspondiente al datasheet de Rational ClearQuest.

2.2. Control de versiones:

El control de versiones, quizás sea la herramienta más importante de todas las que veamos a lo largo de este documento. En ella se almacenará el software desarrollado, debidamente versionado.

Estudiando las herramientas a fondo, podemos observar claramente que IBM nos ofrece una herramienta muy potente, Rational ClearCase (5000€ por licencia), no obstante, en nuestro caso, y para este proyecto hemos optado por confiar en una herramienta, menos sofisticada, y por otro lado más económica: Perforce (600€ por licencia).

Hemos dicho que IBM nos ofrece un entorno mucho mas potente, con ello nos referimos a que posiblemente disponga del mejor entorno multisite para este tipo de herramientas, mediante la técnica de replicar servidores con unas políticas específicas para su mantenimiento, nos permite realizar un buen desarrollo distribuido. El mantenimiento de este sistema de réplicas, por otro lado castiga el tiempo del desarrollador, debido a que los datos subidos a un servidor, tardan un determinado tiempo en ser replicados.

Por otro lado, otro de los puntos negativos de esta herramienta, es que es sumamente difícil de administrar, hecho que nos hace tener un administrador dedicado por cada 30-40 desarrolladores, coste este también a tener en cuenta cuando queramos decidir que herramienta utilizar.

A parte de este coste de administración, ClearCase no es una herramienta fácil de utilizar para los desarrolladores, requiere de una formación específica, que también conlleva un coste adicional, por el contrario, cuando hemos adquirido los conocimientos necesarios para trabajar con este entorno, las posibilidades que nos da son infinitas en comparación a cualquier otro control de versiones.

A todo esto debemos añadir, que usando este control de versiones, las integraciones no son triviales, y por tanto, toma fuerza la figura de un integrador, persona dedicada a integrar los cambios en una rama "main", de la cual saldrán las versiones del producto final.

Como ya hemos dicho anteriormente, en nuestro caso nos hemos decidido a trabajar bajo el entorno Perforce, a continuación veremos como opera esta herramienta internamente.

La siguiente figura nos muestra la conexión entre un servidor de Perforce y un cliente: este cliente puede tener cualquier sistema operativo, como pueden ser (Windows, Unix o Mac OSX), mientras que por otro lado se puede escoger entre tener un servidor Linux o Windows.

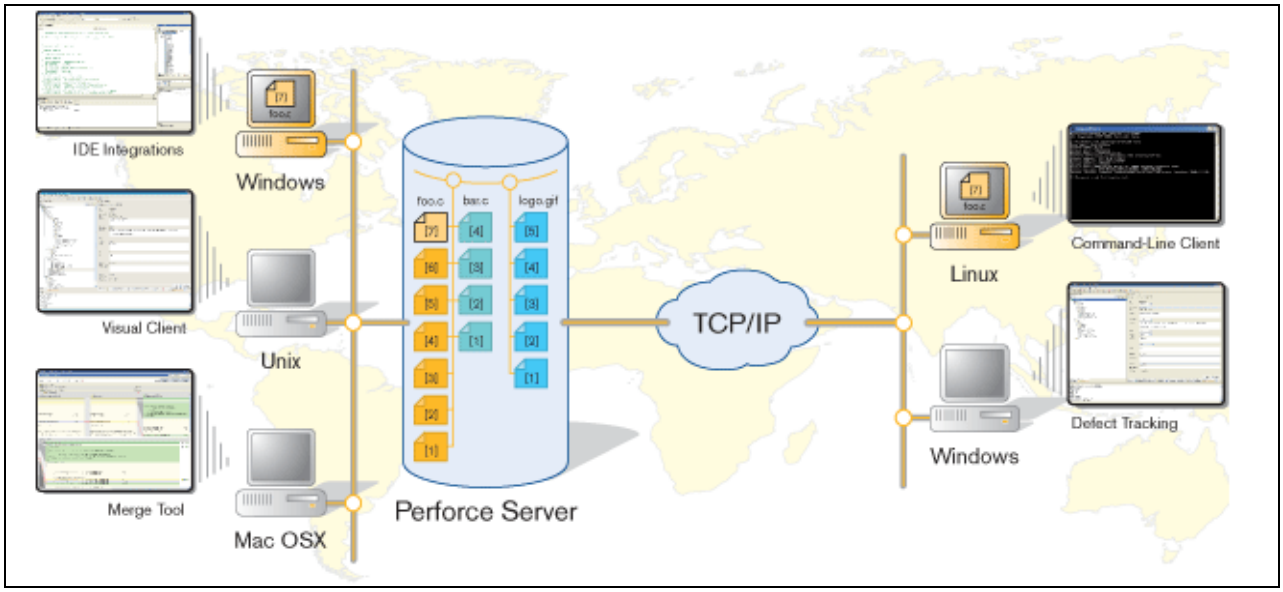

Figura 3

Servidor Perforce

Puesto que debemos considerar el desarrollo distribuido, vamos a ver como internamente maneja Perforce un entorno distribuido.

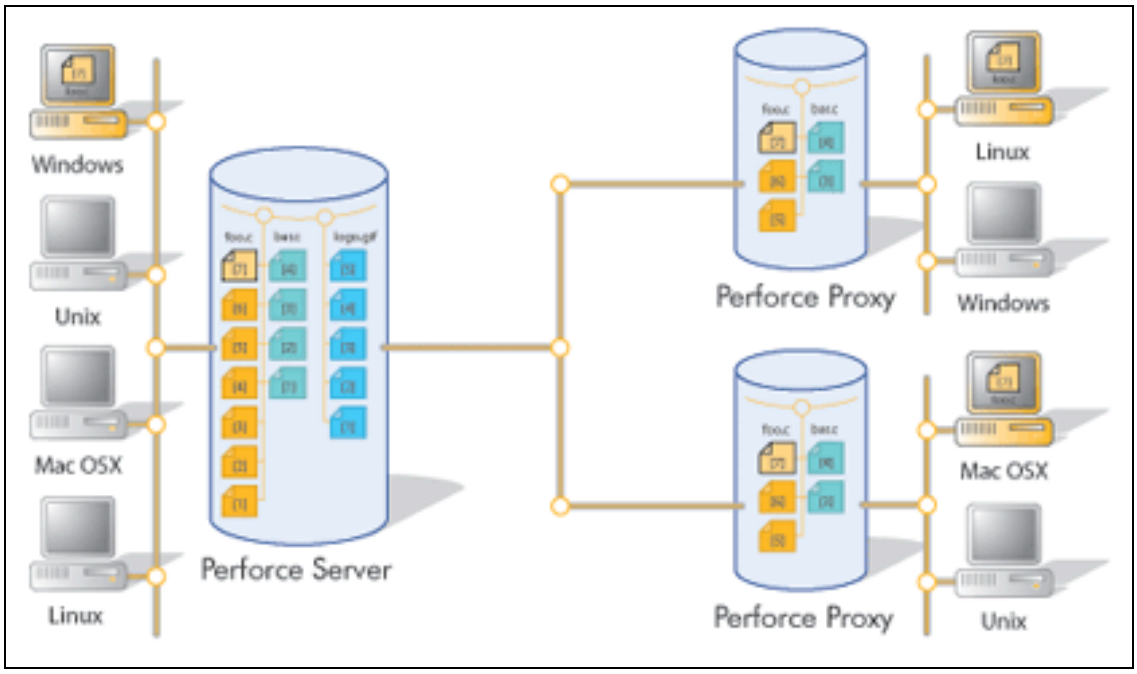

Figura 4

Perforce Multisite

En otros controles de versiones, como en el caso de ClearCase, el sistema distribuido se crea mediante un servidor en cada "site", en la que los datos se replican mediante una sincronización periódica.

En el caso de Perforce, y gracias a su más que excelente Performance, únicamente es posible disponer de un servidor, mientras que en cada "site", dispondremos de un Proxy para mejorar las comunicaciones. De este modo, y a efectos de los usuarios, siempre se está accediendo directamente al servidor central.

Ambos sistemas tienen sus ventajas y sus inconvenientes, en el primer caso, la sincronización es un proceso que requiere tiempo, y por tanto, puede ser que en un "site", se efectúen unos cambios, y en otro tarden un cierto tiempo en verlos, mientras que en el caso de Perforce esto no ocurre, por otro lado, si un servidor falla, en el caso de ClearCase, los datos están replicados en otro y el usuario siempre tiene acceso a él, mientras que en el caso de Perforce, hasta que no se restaurase el sistema, se detendría el desarrollo.

Perforce, a su vez tiene como ventajas principales, su velocidad en las transacciones, punto fuerte de esta herramienta, que permite a los desarrolladores ahorrar mucho tiempo en las operaciones que atacan al control de versiones. A su vez, el entorno de

este control de versiones, es sumamente intuitivo, hecho que hace que la formación a los desarrolladores para el uso de esta herramienta sea de bajo coste temporal y económico.

Llegados a este punto, y habiendo seleccionado ya el control de versiones a utilizar, entramos en la difícil tarea de diseñar las políticas que deberán seguir los desarrolladores al implementar el código bajo un determinado control de versiones.

Debido a que el proyecto, en nuestro caso, cuenta con un número amplio de desarrolladores, debemos garantizar que nuestro desarrollo permitirá cambios de forma concurrente, así como garantizar por otro lado que las integraciones se efectuaran a menudo, esto nos permitirá asegurar la reproducibilidad de las Releases.

Introduciéndonos en el escenario de desarrollo, tendremos un directorio donde se almacenará el código desarrollado, (rama de integración), por otro lado, tendremos las diferentes ramas, donde los desarrolladores implementarán sus actividades, para posteriormente ser integradas en la rama de integración.

Para el control de versiones utilizado, la creación de una rama implica duplicar el código en el servidor, por este motivo, deberemos conseguir un compromiso en la creación de estas, pues la creación de excesivas ramas puede conllevar a un exceso de uso de disco, y a unos tiempos de espera indeseados en el momento de actualizar dichas ramas, mientras que una creación insuficiente de las mismas, nos llevaría a determinados problemas en la trazabilidad del proyecto.

Por otro lado, debemos tener en cuenta, por ejemplo que el proyecto sea desarrollado desde varios emplazamientos, ya que Perforce únicamente puede contar con un servidor centralizado, donde se almacenaría toda la información, gracias a la utilización de un Proxy, veremos reducido en gran medida el tiempo de acceso a los datos.

Una manera muy óptima para el desarrollo sería, que cada desarrollador implementara cada actividad en una rama diferente, y luego se integraran las actividades que se deseen, como muestra el ejemplo siguiente:

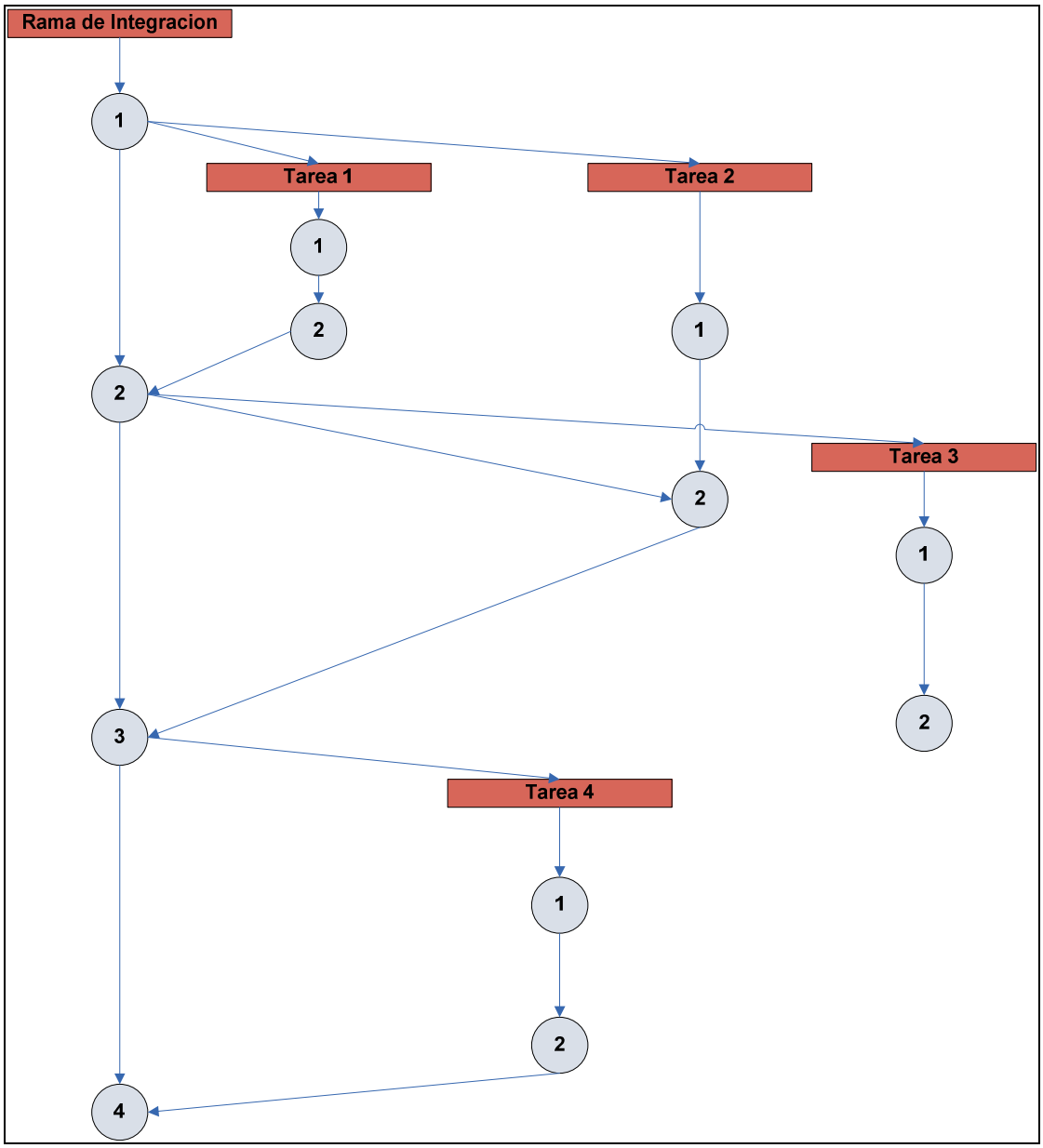

Figura 5

Diagrama: 1 rama por actividad

En este caso se desarrolla cada tarea en una rama diferente, la tarea 1 y 2 parten de la versión uno en la rama de integración.

Al finalizar la primera se integran los cambios en la rama de integración, posteriormente, se desea integrar la segunda, así que para evitar conflictos, se cogen los cambios recientes en la rama de integración, para posteriormente integrar la tarea dos en esta rama.

De la misma versión de la rama de integración, se inicia el desarrollo de la tares 3, como se puede observar, esta tarea esta todavía en desarrollo, o bien se ha finalizado, pero no se ha deseado integrar, así que los cambios se mantiene estables en la rama de desarrollo de la correspondiente tarea.

Del resultado de integrar la tarea 2, se inicia el desarrollo de la tarea 4, tras su finalización, al desear integrar los cambios en la rama de integración, esta no ha sufrido cambios, así que al no haber conflictos, los cambios se pueden integrar directamente.

Debido a los problemas que hemos comentado anteriormente, este tipo de desarrollo no sería posible, puesto que el tamaño se aumenta en exceso, y los tiempos de creación y destrucción de ramas son excesivos, de modo que debemos buscar una alternativa.

Teniendo en cuenta las restricciones de la aplicación de control de versiones utilizada, hemos decidido utilizar la siguiente política de ramas:

- Una Rama de integración
- Una única rama para cada desarrollador.

De este modo, los desarrolladores implementarán sus cambios en su propia rama, y posteriormente, los integrarán en la rama de integración. El siguiente esquema muestra cómo se efectúan éstos cambios.

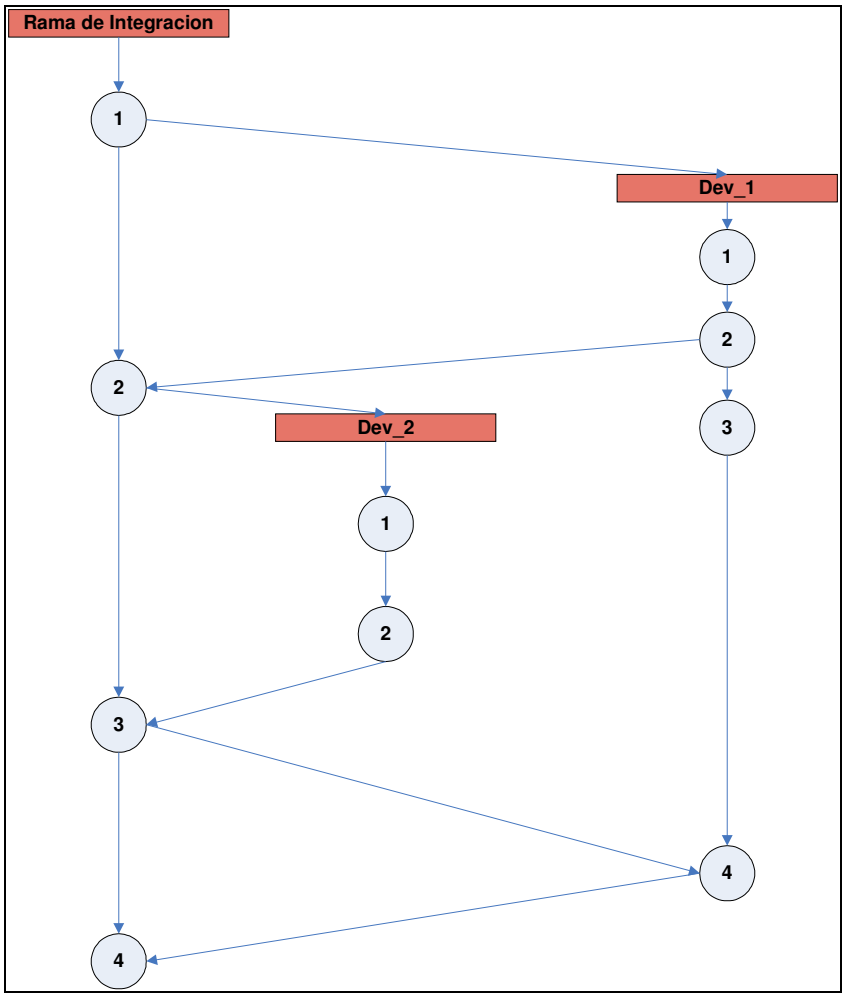

Figura 6

Diagrama: 1 rama por desarrollador

Al crear la rama del desarrollador, todos los archivos existentes en la rama de integración en ese momento se copian en la nueva rama creada. El desarrollador efectúa unos cambios sobre este fichero, creando la versión dos en su propia rama, para posteriormente integrar estos cambios en la rama de integración, y creando la versión dos del fichero en integración.

Posteriormente, se crea la rama de un segundo desarrollador, a partir de la versión dos del fichero, por tanto, este segundo desarrollador ya tiene integrados en su rama los cambios efectuados por el primero. Ambos desarrolladores tienen ahora la misma versión del fichero en sus respectivas ramas, y ambos efectúan cambios sobre los mismos ficheros; el segundo desarrollador termina sus cambios primero, y por tanto puede integrarlos en la rama de integración. El segundo desarrollador, por su lado, una

vez ha finalizado sus modificaciones, debe integrar los cambios efectuados por el segundo en su rama, para posteriormente integrarlos en la rama de integración.

Como se puede apreciar, la primera política nos da una mejor visión del desarrollo de cada actividad, mientras que la segunda política únicamente nos permite hacer un seguimiento del trabajo elaborado por un desarrollador, pero en ningún caso nos permitirá discernir el trabajo efectuado por un desarrollador en diferentes actividades.

Puesto que nuestro sistema no nos permite utilizar la primera propuesta, deberemos realizar unas pequeñas modificaciones en la segunda para conseguir separar el trabajo realizado en una actividad del realizado en otra.

El siguiente esquema muestra el procedimiento a seguir:

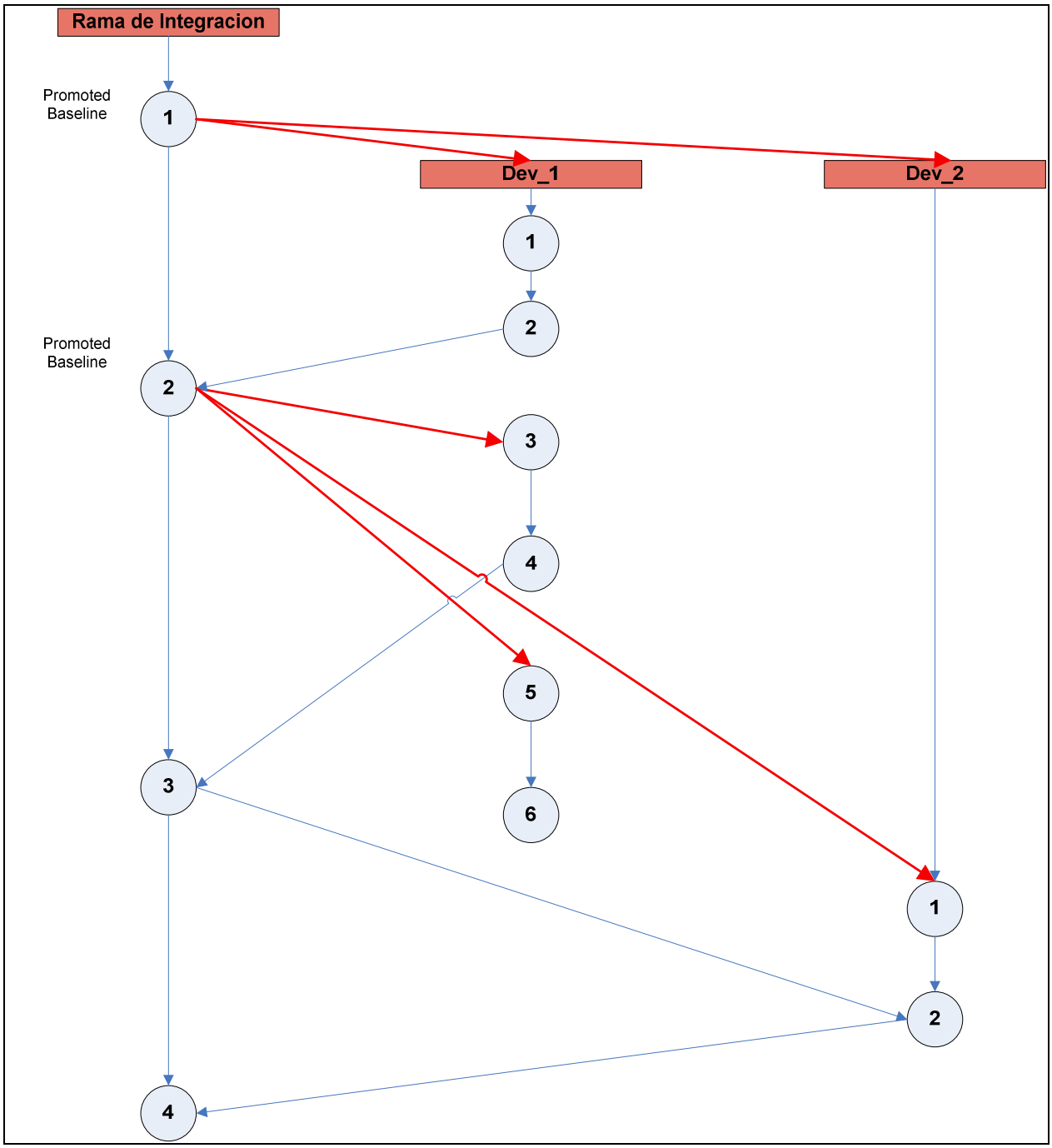

Figura 7

Diagrama: 1 rama por desarrollador utilizando baselines

Mediante el uso de este procedimiento, podemos separar claramente el trabajo efectuado en cada actividad, desarrollándolas todas ellas en una misma rama de desarrollo.

Se crea una rama para cada desarrollador a partir de la última baseline estable (dev_1, Dev_2). El desarrollador 1 termina las modificaciones de la tarea y la integra en la rama de integración. En el ejemplo, en ese momento se crea una baseline estable. Al querer empezar a desarrollar la segunda actividad, la rama de desarrollo toma el mismo estado que la rama de integración en su última baseline estable. Igual que pasa en la versión 5 de esta rama.

De este modo, podemos garantizar exactamente cuál ha sido el desarrollo para cada actividad, realizando todas las actividades en una misma rama.

2.3. Control de versiones de documentos:

Una de las partes más importantes de un proyecto es su documentación, y por tanto, ésta, debe estar almacenada de forma segura, y debidamente versionada.

Por otro lado, no todos los usuarios deben poder acceder a toda la información.

Para ello hemos decidido utilizar la plataforma SharePoint de Microsoft, ya que además de permitirnos lo anteriormente comentado, soporta todo tipo de formatos, y su interfaz es ampliamente familiar para todos aquellos que ya hayan usado cualquiera de las herramientas de Microsoft Office.

3. Escenario a automatizar

En todo proyecto, para que éste tenga una correcta organización de sus actividades, tiene que haber un proceso, claramente definido, entre las tareas a realizar, y sus actores.

En el proceso contemplado, hemos tomado los siguientes roles:

- **Managers**
- **Project leaders**
- **Developers - Integrador**

En el siguiente gráfico se puede contemplar como conviven estos roles, con las herramientas explicadas en el punto anterior. Más adelante explicaremos este gráfico en detalle.

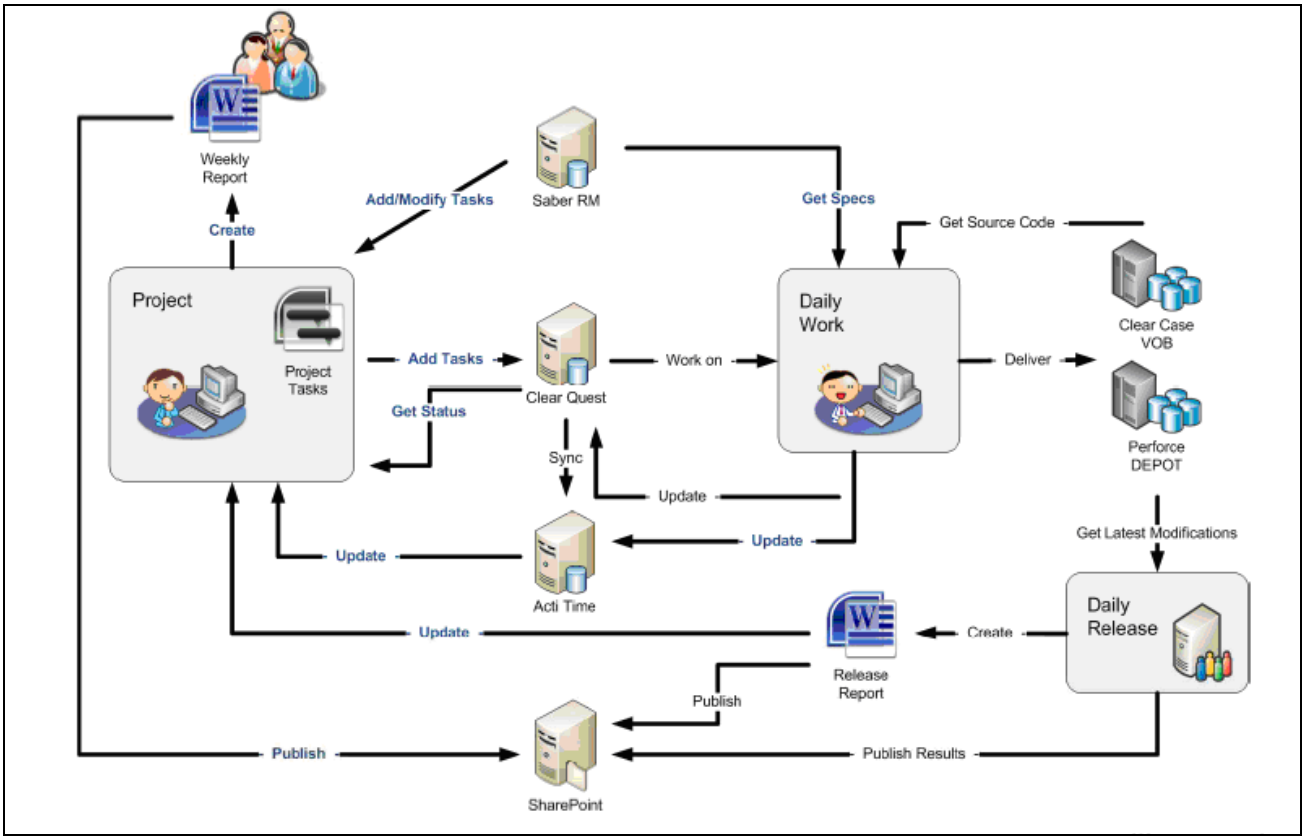

Figura 8

Entorno de un proyecto

Managers

El siguiente gráfico nos muestra el entorno con el que interactúa un "Manager"; básicamente, su función en la fase de desarrollo de un proyecto, es la de tener una visión global del estado de las actividades, y publicar estos estados bajo un control de versiones de documentación, (en nuestro caso SharePoint)

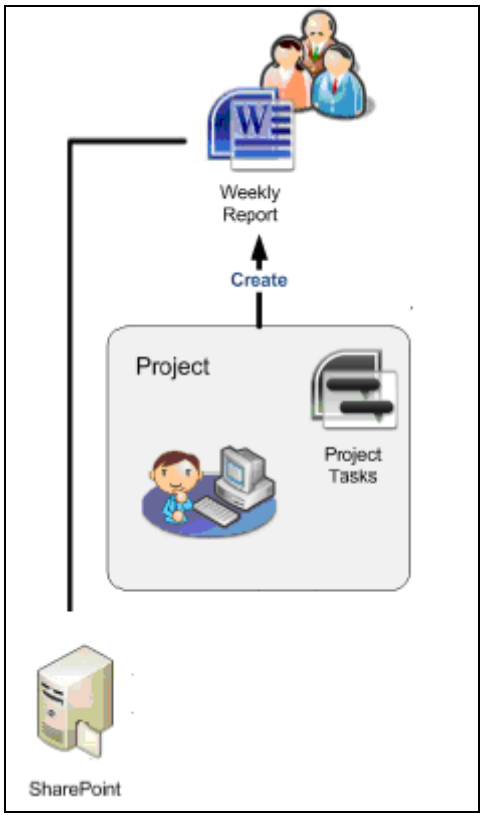

Figura 9

Entorno de actuación de un Manager

A continuación vamos a definir el proceso que deberían seguir los managers antes de publicar la documentación.

En primer lugar, deberían hacer una consulta a la base de datos de actividades, y verificar el estado de las actividades.

Una vez verificado el estado de las mismas, el manager realizaría un documento, y éste sería publicado en SharePoint.

Más adelante será mostrada una herramienta cuya finalidad es la de automatizar este proceso.

Project Leader

El siguiente gráfico muestra el escenario en el que el Project leader se ve involucrado. Su función principal es la de, mediante los documentos obtenidos de la Release, reorganizar las tareas existentes en las diferentes bases de datos.

Según los resultados de estos "reports, el Project leader puede asignar nuevas tareas a algunos desarrolladores, reorganizar las tareas existentes, o reasignar tareas de aquellos desarrolladores más sobrecargados, a otros que no lo sean tanto.

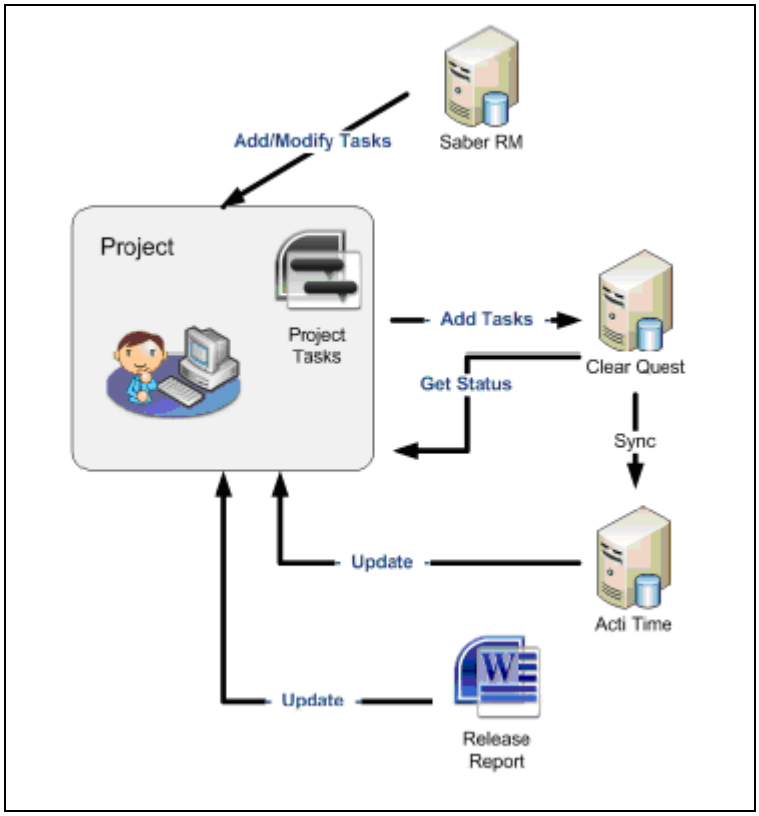

Figura 10

Entorno de actuación de un Project Leader

Releases

Todo proyecto debe tener versiones estables y por tanto, etiquetadas específicamente. Éste proceso consiste en la obtención del código de los debidos repositorios, posteriormente el Project leader o la persona asignada en cualquier caso, realizaría una compilación, revisaría que el proceso fuera correcto, para posteriormente etiquetar la versión debidamente, realizar un report de los resultados, y publicar el mismo en el control de versiones de documentos, como muestra el siguiente gráfico.

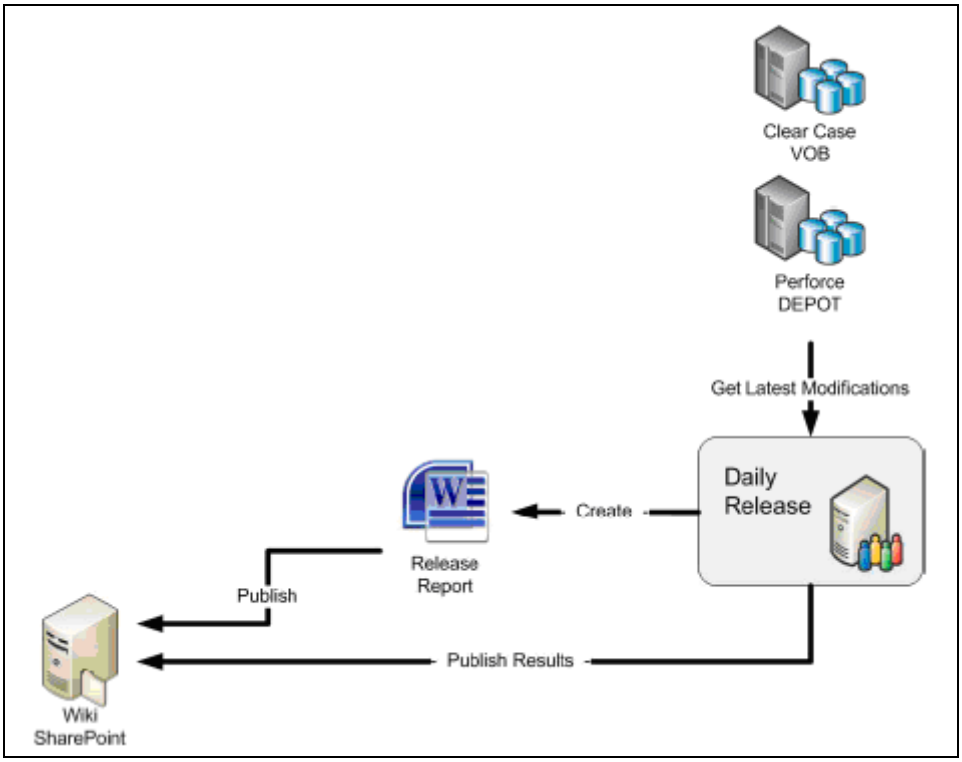

Figura 11

Entorno de actuación de las Releases

Developer

Para finalizar con este apartado, podemos observar un gráfico que muestra el entorno con de acción de un desarrollador.

A grandes rasgos, el desarrollador debe coger las especificaciones de una determinada base de datos (SaberRM); asimismo, debe empezar a trabajar en una actividad que le haya sido asignada en otra base de datos (ClearQuest), además debe también actualizar la base de datos que define el tiempo empleado en cada actividad (ActiTime), para finalmente coger el código del control de versiones (Perforce), y realizar sus modificaciones.

Una vez el desarrollador haya modificado el código, éste debe ser correctamente almacenado en el control de versiones, y de nuevo actualizar todas estas bases de datos.

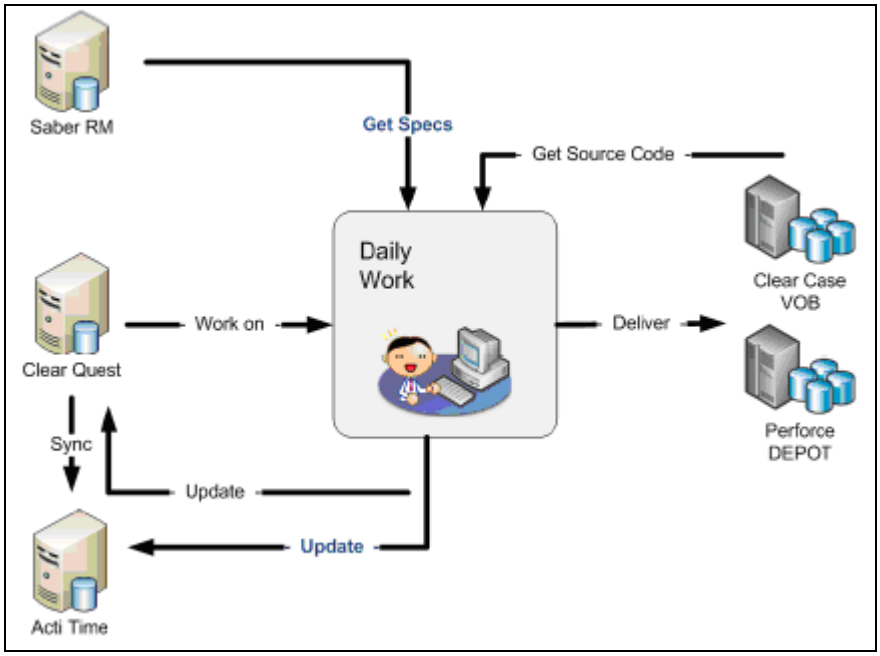

Figura 12

Entorno de actuación de un desarrollador

3.1. Dependencias del proyecto

Antes de desarrollar las herramientas de automatización, debemos hacer un estudio de los requisitos y las dependencias con las demás aplicaciones.

Lo primero que debemos tener en cuenta, es que esta herramienta deberá tener control de múltiples parámetros de estado, puesto que será utilizada por muchos desarrolladores, normalmente, al mismo tiempo, es por eso, que debemos utilizar una base de datos para controlar cada uno de estos parámetros.

La tecnología utilizada para la creación de esta base de datos será MySQL, a continuación mostramos un gráfico con las tablas y sus relaciones.

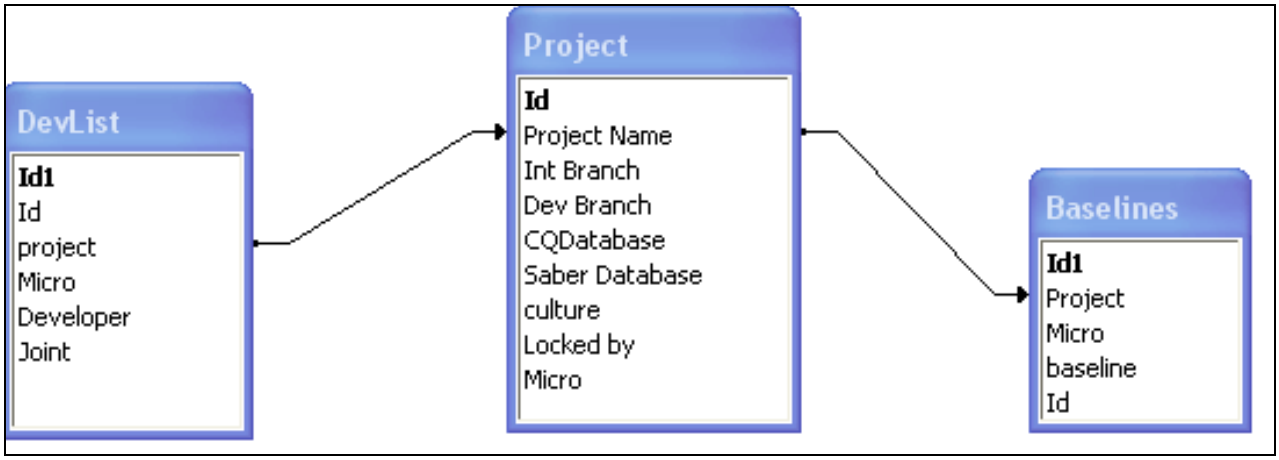

Figura 13

Esquema base de datos para el control de la aplicación

Con el esquema anteriormente mostrado, nos será suficiente para controlar todas nuestras herramientas, esta jamás será modificada por ningún desarrollador directamente, pues el control de nuestras aplicaciones se vería afectado ocasionando pérdidas importantes en cuanto al código se refiere., en caso de error, el administrador del sistema será el encargado de volver a dejarlo en un estado estable.

Una vez tenemos definida la base de datos, tendremos en cuenta también que para conectarnos a la base de datos de defectos (Saber DM), necesitaremos utilizar el protocolo SOAP.

Por otro lado, debemos buscar una herramienta que pueda interactuar tanto con la API de ClearQuest como con la API de Perforce.

Además de todos estos requisitos, necesitamos que nuestra aplicación sea lo más "user friendly" posible.

Por todo esto, nos hemos decidido a desarrollar nuestras herramientas bajo la tecnología de Visual Basic .NET.

3.2. Automatización de las “Releases”

Teniendo en cuenta la política de desarrollo que sigue nuestro control de versiones, en la que todos los desarrolladores integran sus cambios en la rama de integración (push model), es muy importante tener un control sobre dicha rama.

Con este fin, diariamente, se tomará el código que haya en la rama de integración, y se le aplicarán una serie de test para verificar el estado de este. En función del resultado de estos tests, se evaluará si el estado actual de la rama es correcto o no.

En caso de que sea correcto, se etiquetará esta versión con un nombre determinado, por otro lado, si esta versión fallara, se debe etiquetar de una forma distinta, para su posterior corrección.

En caso de que la release sea correcta, se debe crear un documento con los nuevos cambios introducidos en la rama de integración, y publicarlo en SharePoint, para que los desarrolladores sean debidamente informados.

Por supuesto, durante el proceso de release, la rama de integración debe estar bloqueada, consiguiendo así que nuevos cambios no sean integrados mientras se esta compilando una versión, o bien se este etiquetando esta. La duración por tanto de este proceso varía en función del volumen de código que haya en el proyecto, y de los test que se deseen pasar para determinar si la versión es o no válida.

Así pues el estado del código quedaría de la siguiente forma:

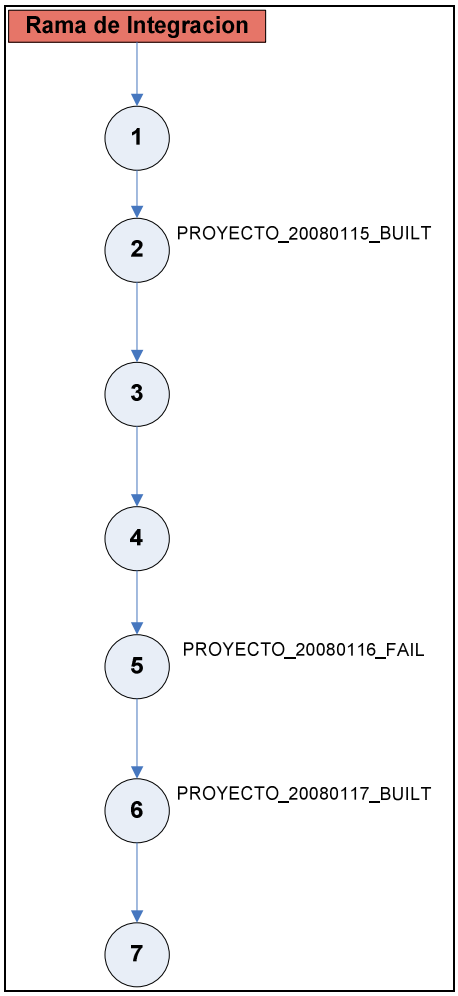

Figura 14
Rama de integración

En el ejemplo anterior, podemos observar, que la versión 2 de la rama de integración fue una release satisfactoria realizada el dia 15/01, asi como que la versión del dia siguiente no pasó todos los test o bien no compiló, mientras que los posibles problemas fueron resueltos en la release del día siguiente.

Con el objetivo de automatizar el proceso de creación de una release, se ha desarrollado una herramienta que genere estas de forma automática. Dicha herramienta se ejecutará como una tarea programada, a las horas que el proyecto tenga menos carga de desarrollo.

Lo primero que hará esta herramienta será bloquear la rama de integración, para que ningún desarrollador pueda modificarla durante este proceso; posteriormente, descargará la última versión de esta rama, y lanzará la compilación, pasará una serie de tests automáticos, y chequeará los resultados, en función de éstos, etiquetará la rama de una forma u otra, y generará un report, por último la rama de integración será desbloqueada dando por finalizado este proceso.

Dado que todo este proceso puede tardar más de una hora en realizarse, es muy importante que éste no necesite interacción alguna con una persona, y dado que requiere que nadie modifique la rama de integración, es muy importante que se lance cuando nadie este trabajando, por ejemplo por la noche.

3.3. Automatización del desarrollo

A continuación vamos a tratar de automatizar el escenario con el que interactúa el desarrollador. El siguiente gráfico nos muestra las transiciones posibles para el desarrollador para llevar a cabo su implementación.

Así que realizaremos un conjunto de aplicaciones para garantizar que cada una de las actividades se desarrolla rigurosamente bajo este escenario, sin errores, y sin excepciones.

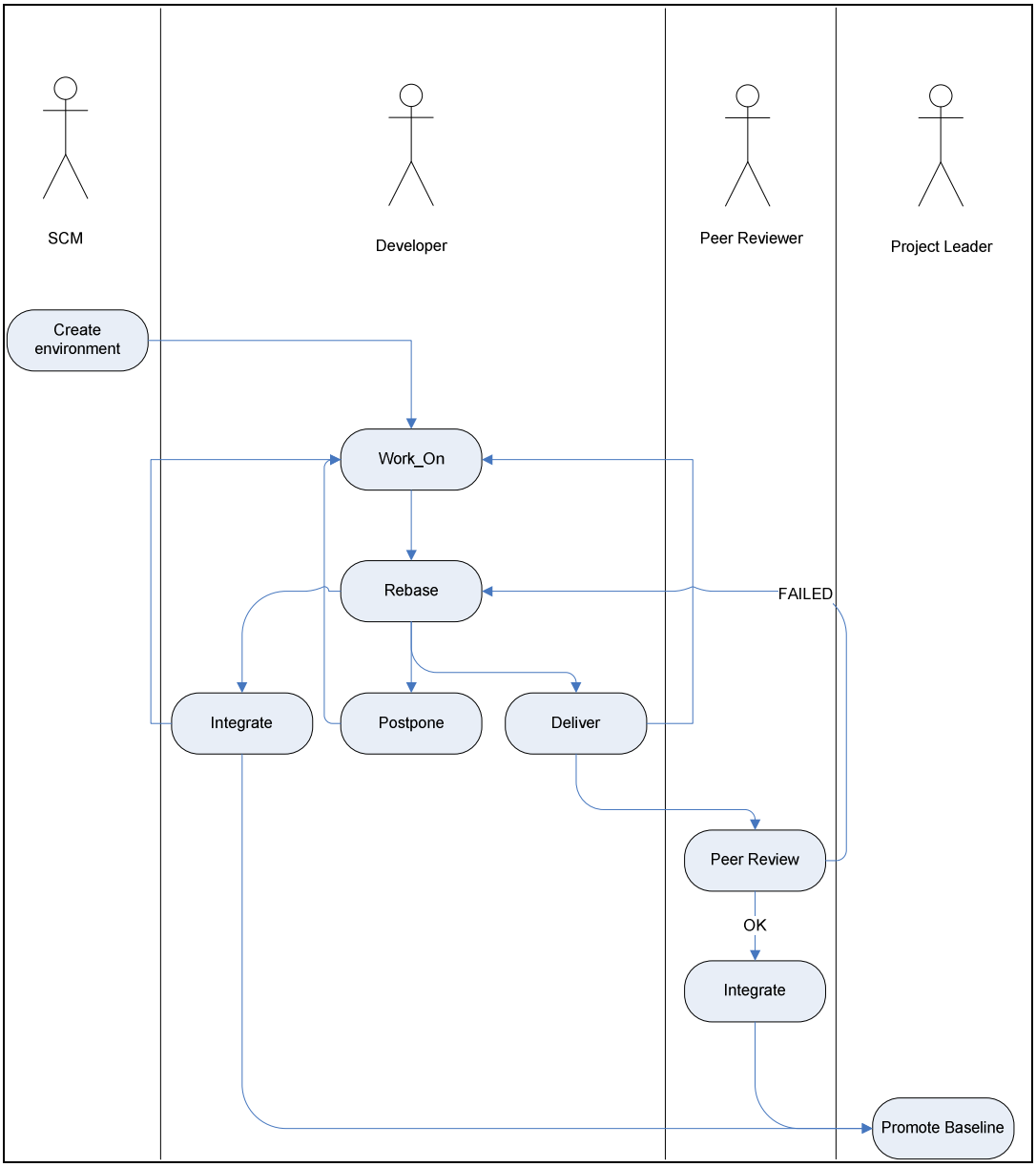

Figura 15

Diagrama de transiciones del desarrollador

Como podemos ver en el anterior gráfico, el desarrollador, una vez ha empezado a trabajar en una actividad (Work_on), realizará todas las modificaciones pertinentes en el código, hasta su posterior finalización (Deliver), cuando la tarea llegue a este estado, la persona asignada para revisarlo, confirmará que los cambios sean correctos, de ser así, dichos cambios se integrarán directamente en la rama de integración, para su posterior promoción en la fase de Release.

3.3.1. Select Project

Debemos contemplar la opción que nuestros desarrolladores puedan estar simultáneamente trabajando en dos proyectos a la vez, así pues, para empezar se ha diseñado una herramienta para permitir seleccionar el proyecto al que pertenece la actividad que se desee implementar.

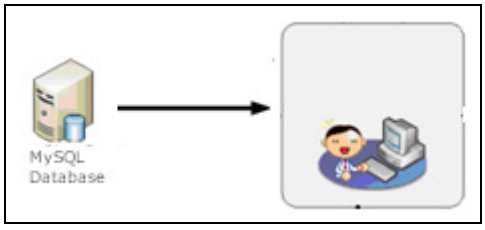

Figura 16
Entorno del SelectProject

Como se muestra en al anterior figura, únicamente actuara con el usuario, y contra la base de datos del control de la aplicación.

Esta aplicación será muy simple, mediante un formulario de Windows, mostrará el listado de proyectos seleccionables para el desarrollador. En caso de que el desarrollador no esté unido a ningún proyecto, éste deberá seleccionar uno para unirse, si ya se encontrara unido a algún proyecto, deberá seleccionar uno para unirse de igual modo que antes, y automáticamente sería desunido del anterior proyecto.

El usuario interactúa con interfaz como el que se muestra a continuación.

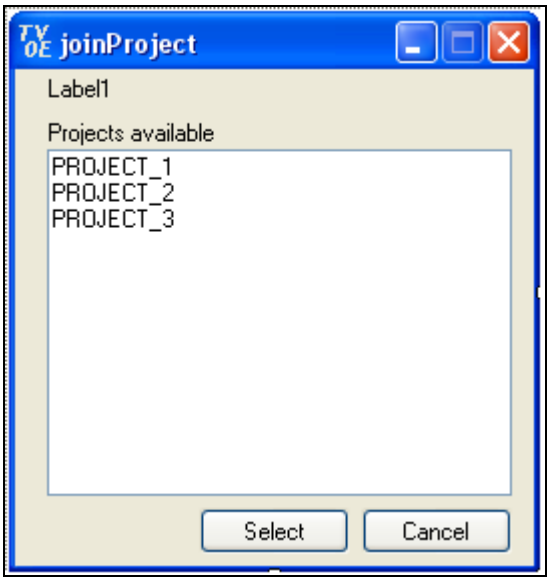

Figura 17

Interfaz del SelectProject

En caso de que el desarrollador ya se encuentre unido a un proyecto, éste sería mostrado donde se ve "Label 1", de otro modo, este campo estaría vacío.

Una vez el desarrollador ya está unido a un proyecto, está listo para empezar a trabajar en una actividad.

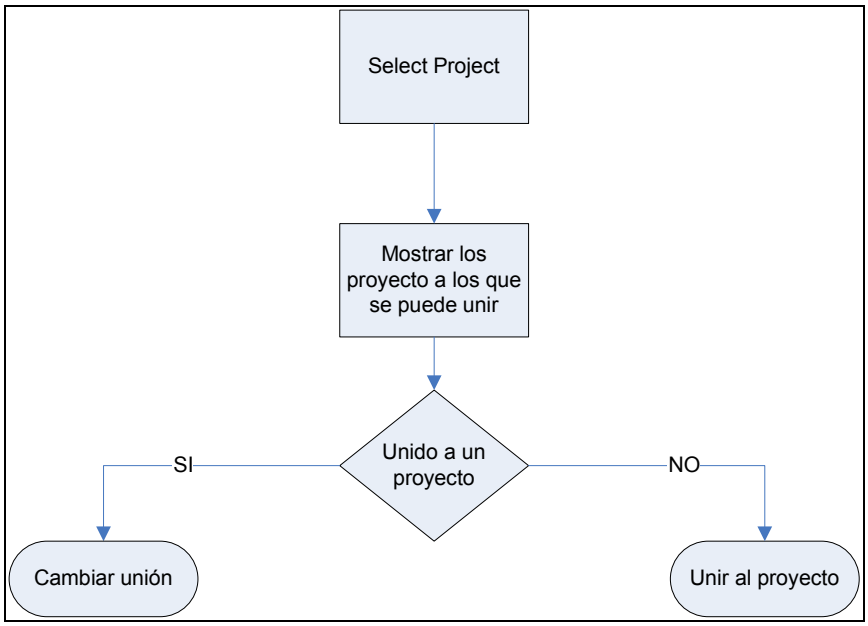

Figura 18

Diagrama de estados del SelectProject

3.3.2. Workon

Como se puede observar en la siguiente figura, el ámbito de actuación de esta aplicación será la obtención de las actividades de un desarrollador, así como el tratamiento de estas y del control de versiones.

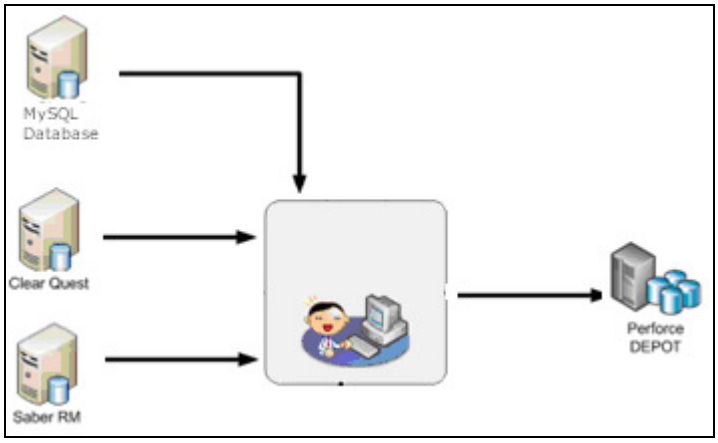

Figura 19

Entorno de actuación del Workon

Esta herramienta se ha diseñado básicamente para configurar el entorno de desarrollo antes de empezar a trabajar en una actividad. Básicamente, antes de entrar en más detalles sobre esta aplicación, lo que hace es listar las actividades que tiene asignadas un cierto desarrollador, para que, una vez éste haya seleccionado una, configure el entorno alrededor de ésta.

A continuación vamos a definir el entorno de esta aplicación.

Mediante un "Windows form", se mostrarán en dos listados diferentes las actividades a desarrollar, (ClearQuest), y los defectos a resolver (SaberDM), además ésta aplicación, cuenta con un textbox, donde se mostrarán las descripciones de las correspondientes tareas seleccionadas.

Por otro lado, y gracias a que ambas bases de datos tienen un entorno web detrás de ellas, haciendo doble click sobre cualquier actividad, se mostrará esta tarea en dicho entorno.

Cuando un desarrollador seleccione empezar a trabajar en una actividad, su estado en la base de datos correspondiente cambiará de estado, (de asignada a abierta), y automáticamente, su rama de desarrollo adoptará el estado de la rama de integración en su última versión estable (latest promoted baseline).

De este modo, el desarrollador tiene su entorno totalmente configurado para empezar a desarrollar la actividad haciendo tan sólo un único click.

La siguiente figura muestra la interfaz de ésta herramienta.

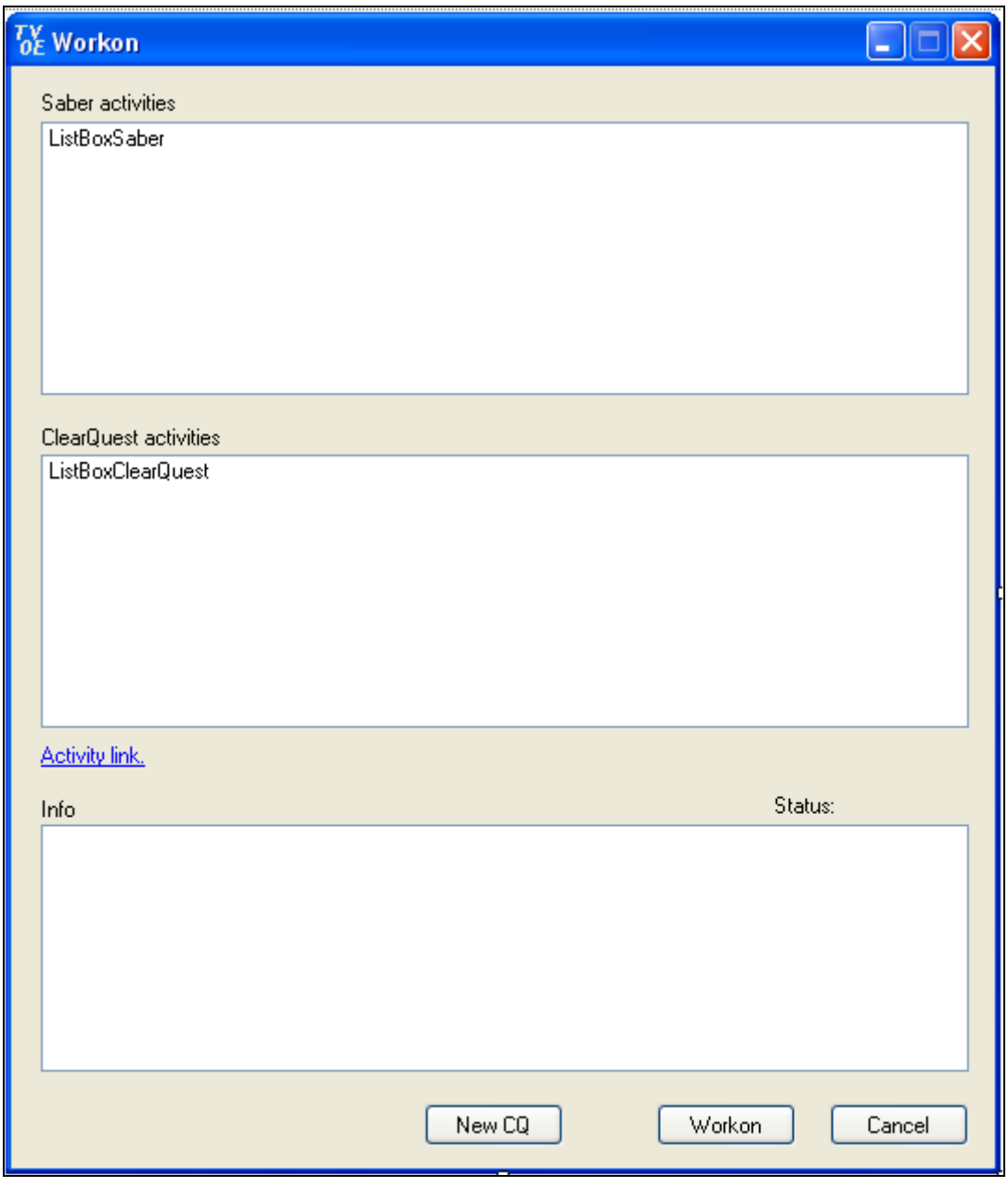

Figura 20
Interfaz del Workon

A continuación podemos ver el diagrama de estados de esta aplicación, sus conexiones a las diferentes bases de datos así como sus acciones frente al control de versiones.

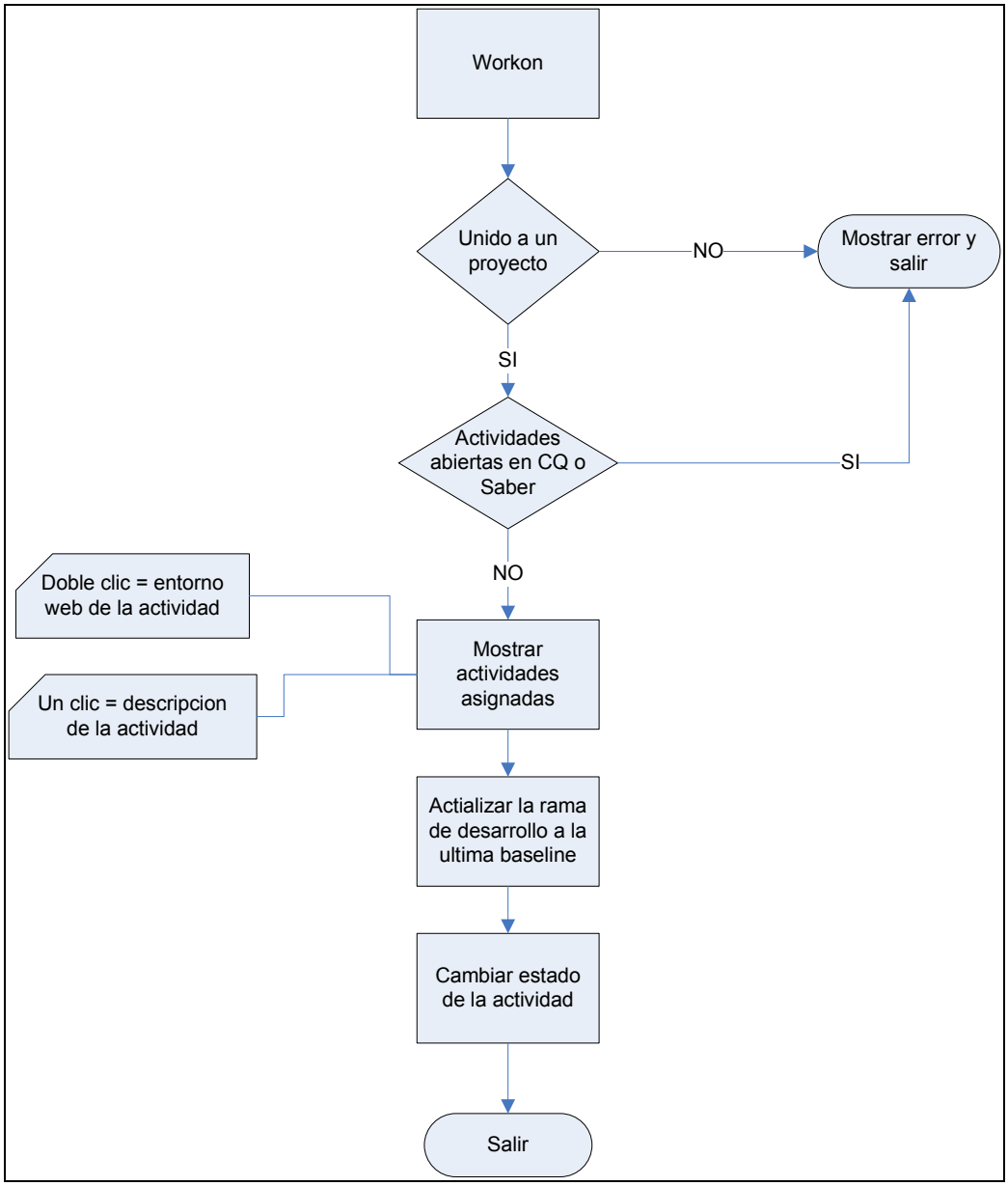

Figura 21

Diagrama de estados del Workon

3.3.3. Checkout

El marco de actuación para esta aplicación queda definido del mismo modo que la aplicación anterior, actuando sobre las bases de datos de actividades y defectos, y contra el control de versiones para realizar los cambios oportunos.

Una vez el desarrollador ya tiene su entorno completamente configurado, es momento de empezar a hacer modificaciones en la rama de desarrollo.

Para poder editar un fichero, primero hay que realizar un checkout, con el fin de no tener que abrir la aplicación de control de versiones para realizar ésta acción, se he diseñado una nueva herramienta para realizar esta acción.

Esta herramienta realizará un checkout del fichero que se le envíe por parámetro, comprobando antes que éste fichero se encuentra en la rama de desarrollo del usuario, así como comprueba que éste esté trabajando en una actividad en ese instante.

Esta herramienta como las demás será integrada en la IDE del desarrollador, pero a diferencia de las demás no tiene interfaz gráfica, simplemente preparará el fichero para ser editado, y mostrará un mensaje si todo es correcto, o bien un mensaje de error en caso contrario.

En el siguiente gráfico podemos ver reflejado explícitamente su funcionamiento.

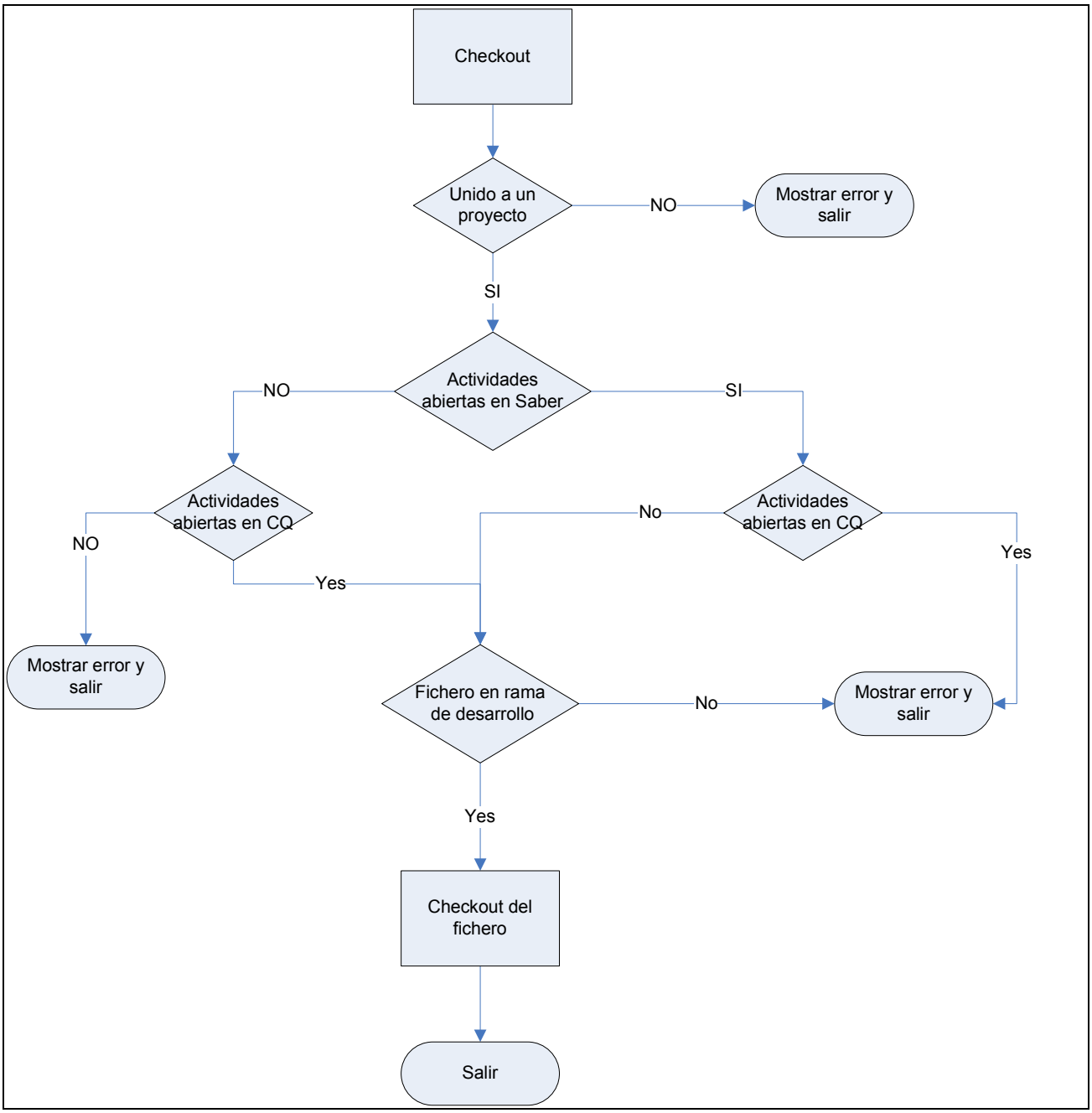

Figura 22

Diagrama de estados del Workon

3.3.4. Revert / Submit

Nuevamente, estas aplicaciones actuaran sobre la base de datos de actividades, así como contra el control de versiones para realizar las modificaciones oportunas.

Una vez el desarrollador ha efectuado cambios sobre un fichero tiene dos opciones, crear una versión de ellos (submit), o bien, deshacer estos cambios (Revert).

Hemos juntado ambas aplicaciones en un único bloque, pues si bien su funcionalidad es claramente diferente, el modo de actuación viene a ser el mismo.

Mediante un formulario se mostrarán todos los ficheros que el desarrollador tiene en el estado de edición (Checked out), para que posteriormente, el usuario seleccione aquellos fichero que quiera guardar, o bien deshacer.

El control de versiones nos obliga a insertar una descripción de los cambios al crear una nueva versión de un fichero, nuestras aplicaciones pondrán automáticamente estos comentarios, insertando en ellos el título de la actividad en la que el desarrollador esté trabajando.

El formulario anteriormente mostrado tiene el siguiente formato: mediante un listbox multiselección, semuestran todos los ficheros que el desarrollador tiene en checkout, seleccionará los que desee guardar o deshacer, y mediante un simple click, los ficheros volverán a quedar en un estado estable. Este formulario se muestra a continuación.

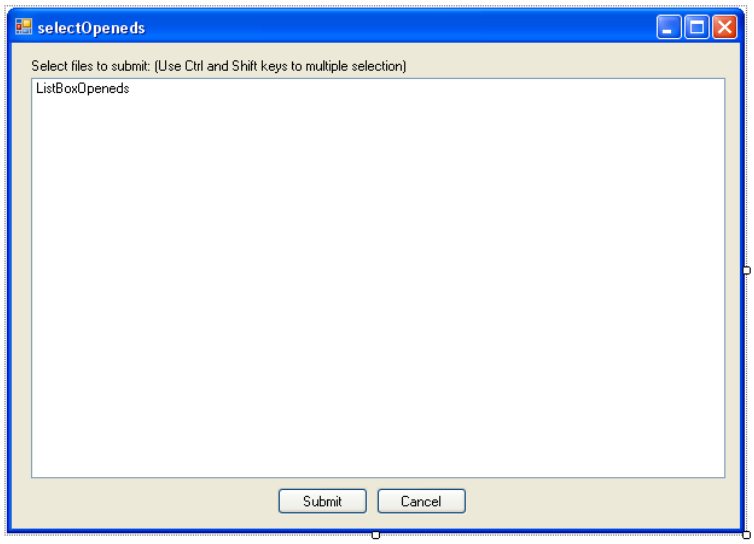

Figura 23

Interfaz del Revert / Submit

En la siguiente figura vemos reflejado su diagrama de estados, muy parecido al del Checkout, pero con ligeras modificaciones, pues este se conecta a las diferentes bases de datos de actividades y defectos para poder actualizar automáticamente el control de versiones.

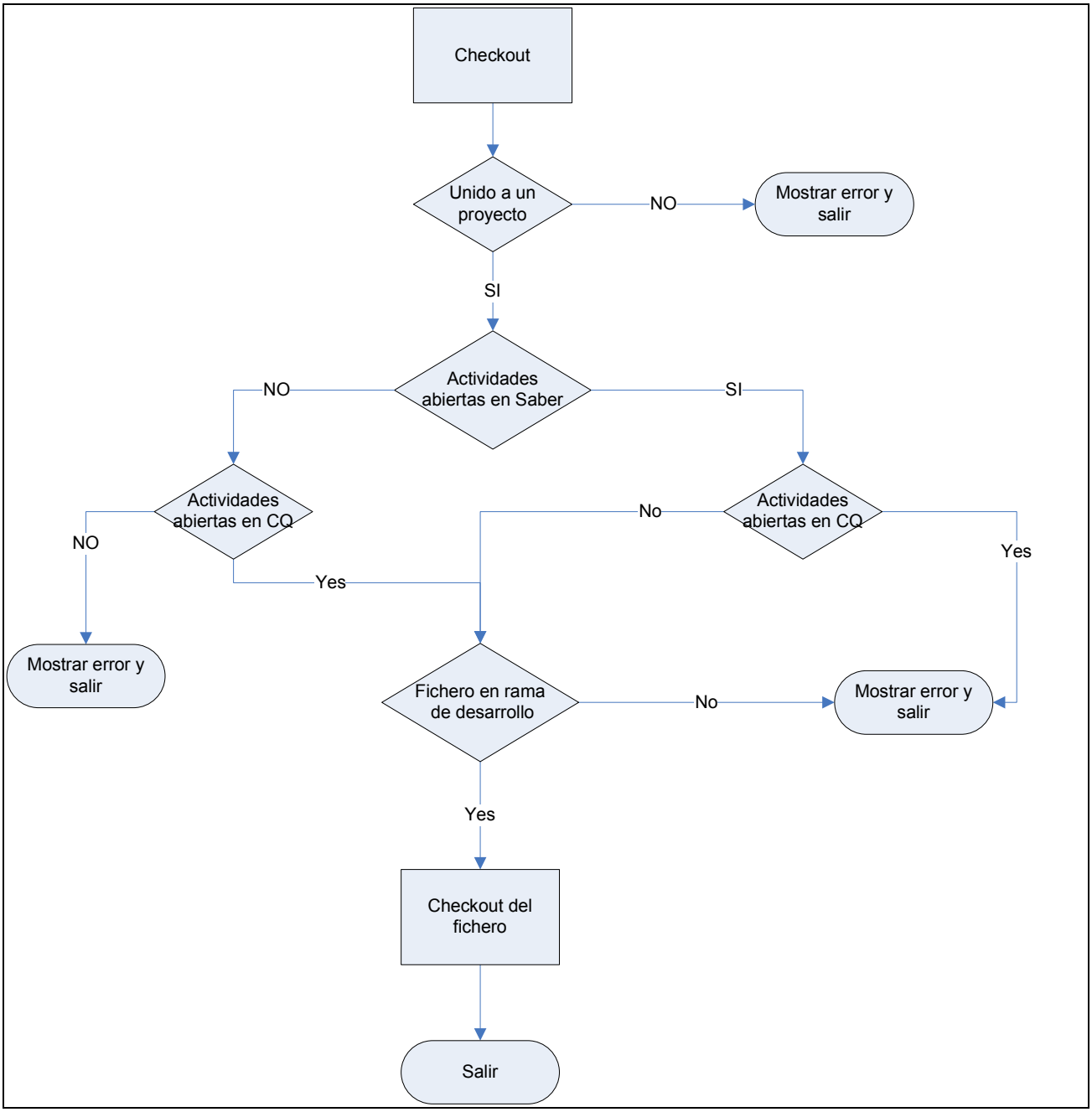

Figura 24

Diagrama de estados del Revert / Submit

3.3.5. Rebase

En este caso, el ámbito de actuación de esta aplicación será únicamente contra el control de versiones, tal y como muestra la siguiente figura.

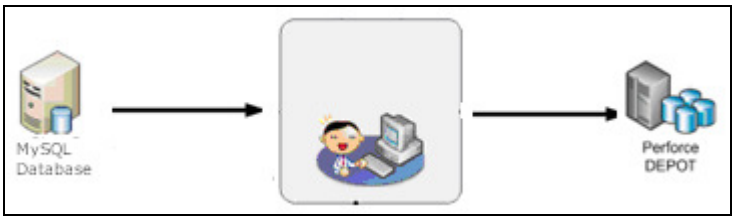

Figura 25

Entorno del Rebase

En cualquier momento, el desarrollador puede desear integrar cambios efectuados en la rama de integración, a su rama de desarrollo. Esta operación se hará mediante la herramienta de Rebase.

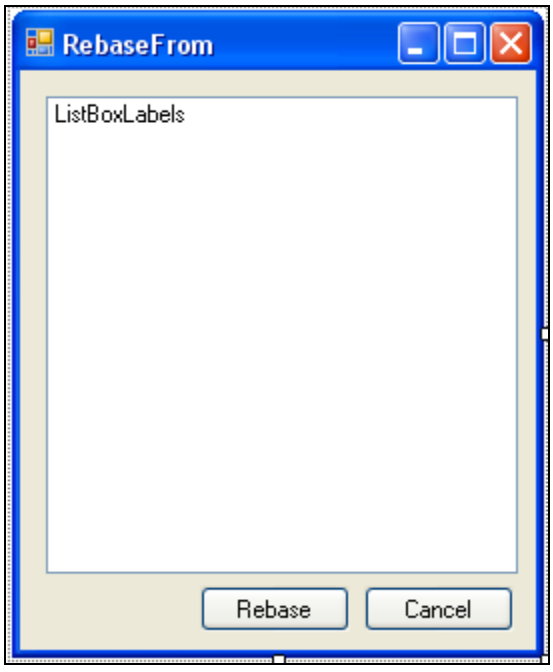

Figura 26

Interfaz del Rebase

El usuario, mediante un formulario de Windows, como el que se muestra en la figura anterior, seleccionará la baseline desde la cual quiere obtener los cambios. En este formulario, se mostrarán todas aquellas baselines desde la última obtención de código

de la rama de integración; cabe notar que la primera obtención de código se hace al hacer workon de una actividad.

Mostrando las baselines, se da la posibilidad al desarrollador a obtener los estados estables de código que ha habido en la rama de integración.

Además de dar la posibilidad de obtener el código desde una baseline estable, el desarrollador también tiene la posibilidad de integrar el último estado de la rama de integración a su rama, conociendo éste que los cambios efectuados en esta tal vez no sean correctos.

Su manejo se ve claramente reflejado en el gráfico de la página siguiente.

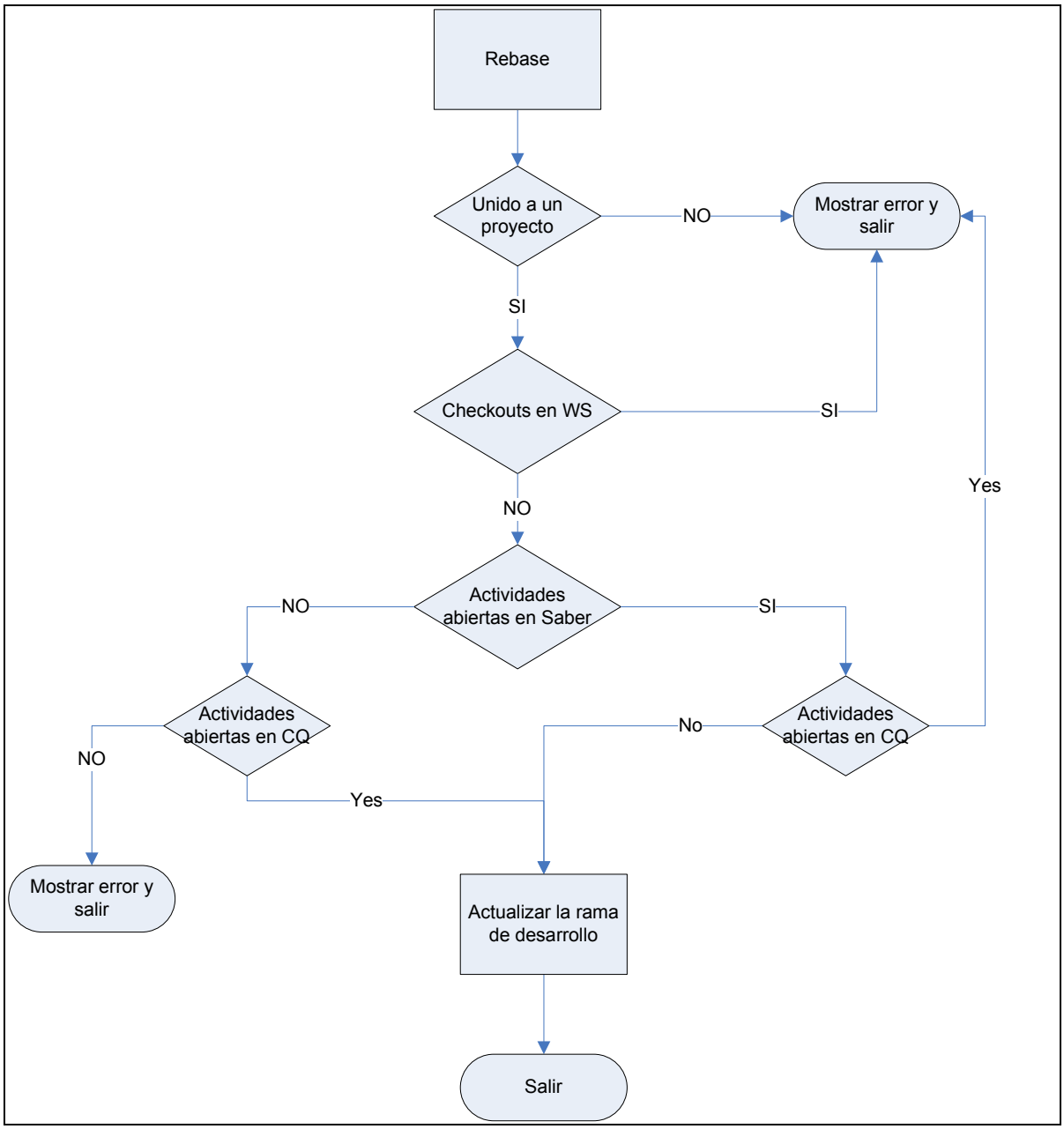

Figura 27

Diagrama de estados del Rebase

3.3.6. Deliver

De nuevo esta herramienta requerirá de las bases de datos de actividades así como del control de versiones, tal y como muestra la siguiente figura.

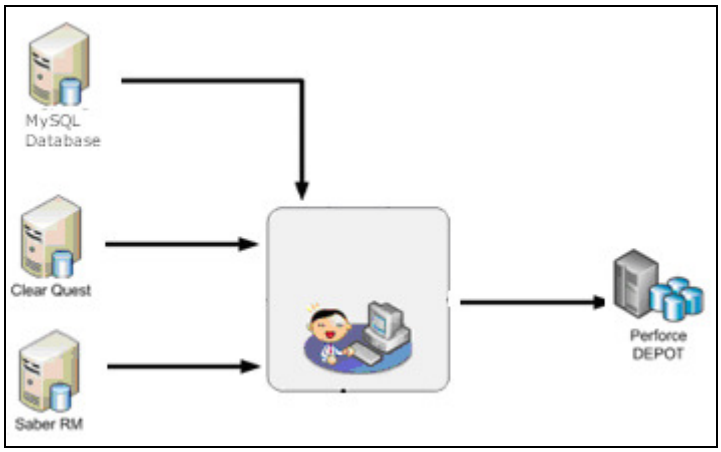

Figura 28

Entorno de Deliver

Una vez el desarrollador haya finalizado la tarea, esta deberá ser revisada por otro desarrollador, con el objetivo de verificar que el código ha sido desarrollado correctamente, que las descripciones introducidas son acordes con el código modificado, o bien que el código implementado se ciñe rigurosamente a las especificaciones requeridas.

Con este fin, le será enviado un mail la persona asignada como peer reviewer para esa actividad.

Por otro lado, se entiende que la actividad ha sido resuelta, esta, tomara el estado de Resolved en ClearQuest, mostrando un formulario como el siguiente para completar la información de la tarea en la base de datos de actividades.

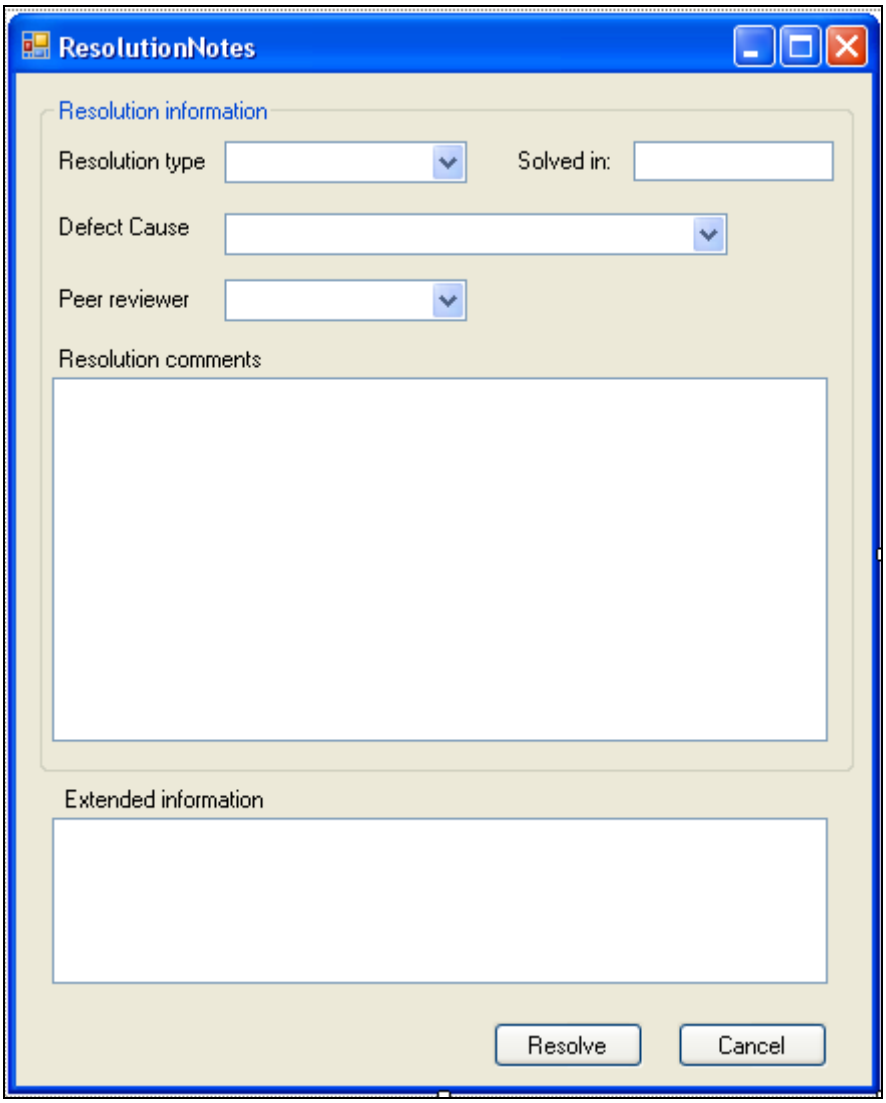

Figura 29

Interfaz del Deliver

En la siguiente figura podemos ver la definición interna de la herramienta.

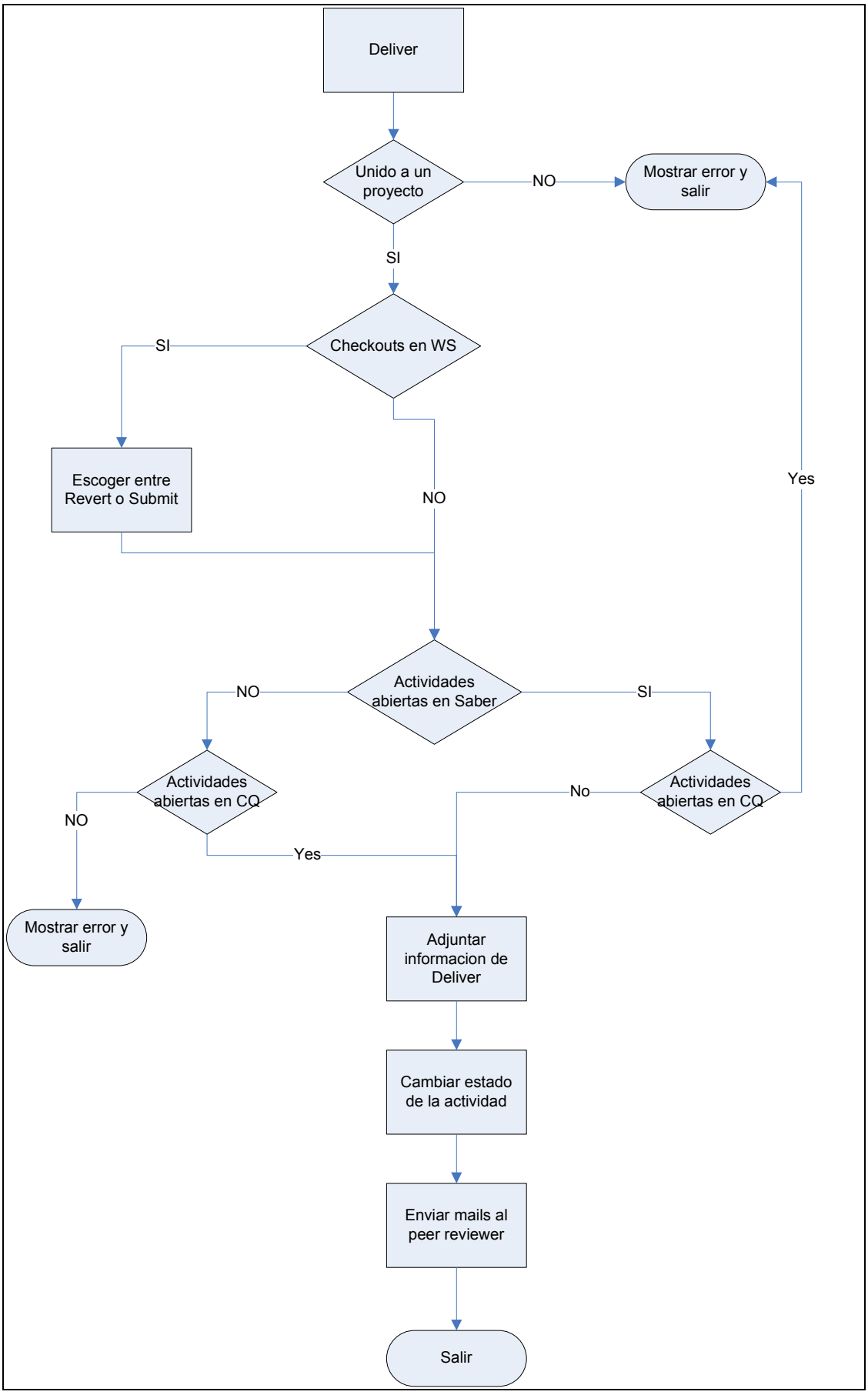

Figura 30

Diagrama de estados del Deliver

3.3.7. Integrate

Esta aplicación tendrá su abanico de actuación comprendido entre las bases de datos de actividades, el control de versiones y un usuario externo, que tomará el rol de revisor del código modificado, tal y como muestra la siguiente figura.

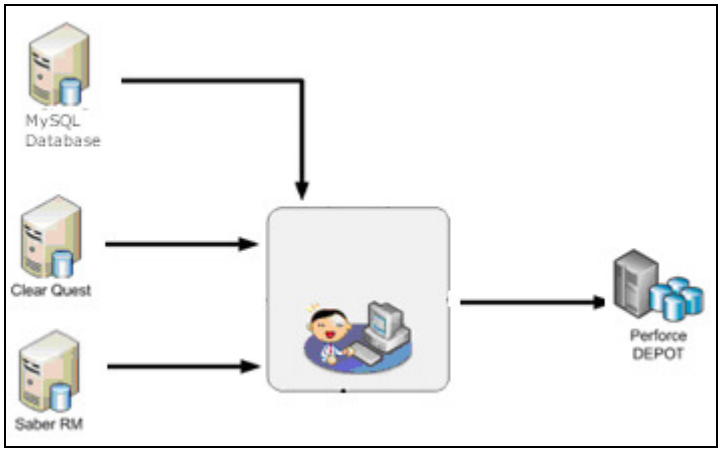

Figura 31

Entorno del Integrate

Llegados a este punto, es momento de que el código sea revisado. La persona asignada ejecutará esta aplicación, y mediante unas preguntas que irá respondiendo comprobando el desarrollo de la tarea, deberá decidir si la implementación es correcta (Accept), o por otro lado se deben hacer modificaciones, (Deny).

El formulario para las revisiones se muestra en la siguiente figura:

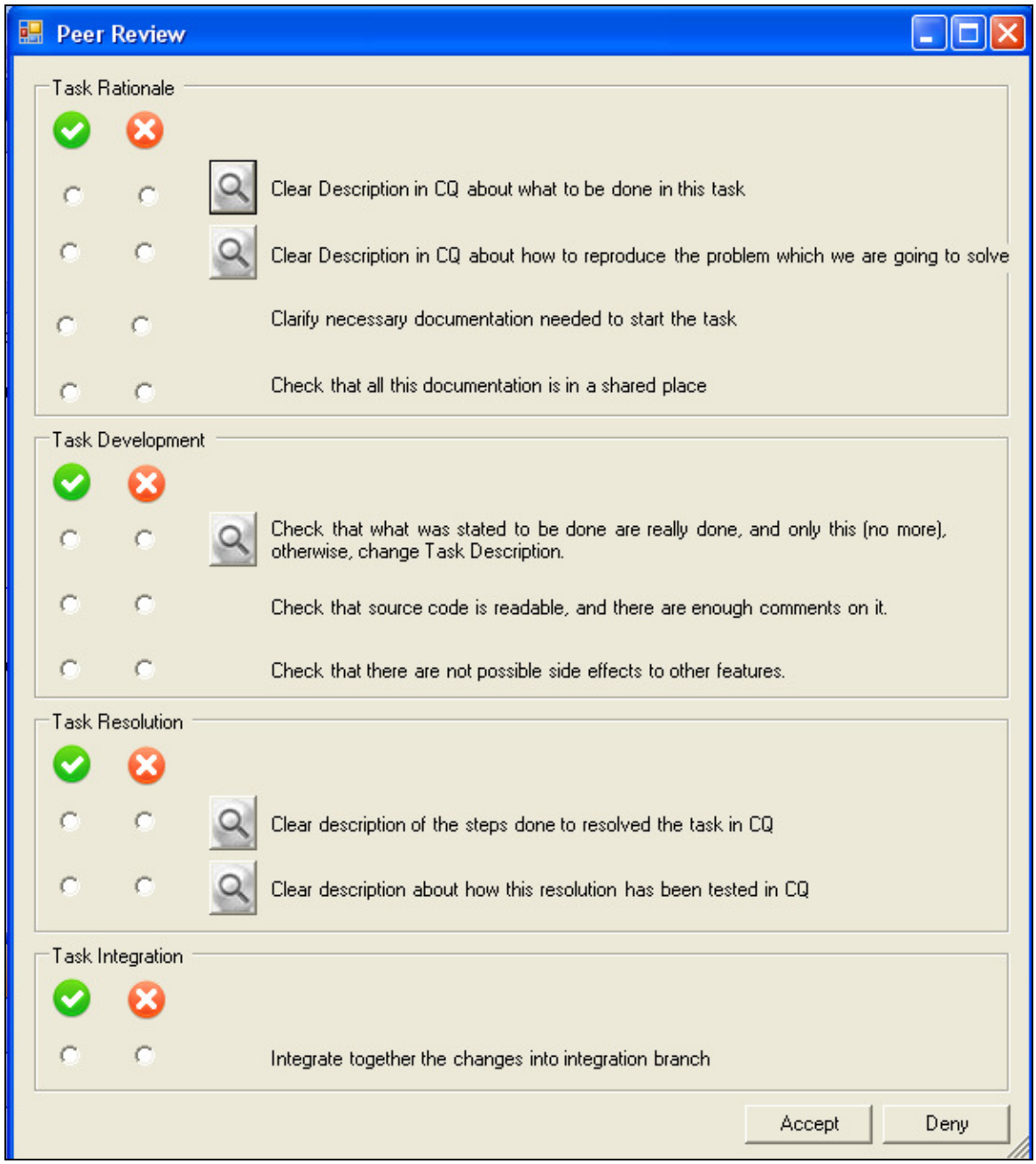

Figura 32

Interfaz del Integrate

Como se puede ver en la figura anterior, las preguntas son de simple respuesta, dando la posibilidad a comprobar según que campos directamente desde la aplicación.

Tanto en el caso de que la Peer review sea aceptada, como en el caso de que sea rechazada, se guardarán los resultados en ClearQuest. En el supuesto de que sea aprobada, los cambios realizados en el software se integrarán directamente en la rama de integración, y la tarea tomará el estado de Delivered.

Si por otro lado la tarea no es aprobada, no se integrará, y su estado no será cambiado en ClearQuest, pero sí insertados los resultados de la peer review. La tarea del desarrollador en este momento, será corregir el código, eliminando los errores detectados en la peer review.

Una vez estos errores se hayan corregido, no se efectuara una nueva peer review, y será el desarrollador el encargado de integrar sus cambios en la rama de integración.

La misma aplicación, si ya se ha efectuado una peer review para esa actividad, automáticamente integrará los cambios, siendo responsabilidad del desarrollador el haberlos corregido.

Una vez los cambios han sido integrados, la actividad será puesta en el estado de "Delivered", dándola por finalizada.

Ahora el desarrollador podría volver a empezar el ciclo con un nuevo Workon.

La implementación interna se ve reflejada en el siguiente diagrama de estados.

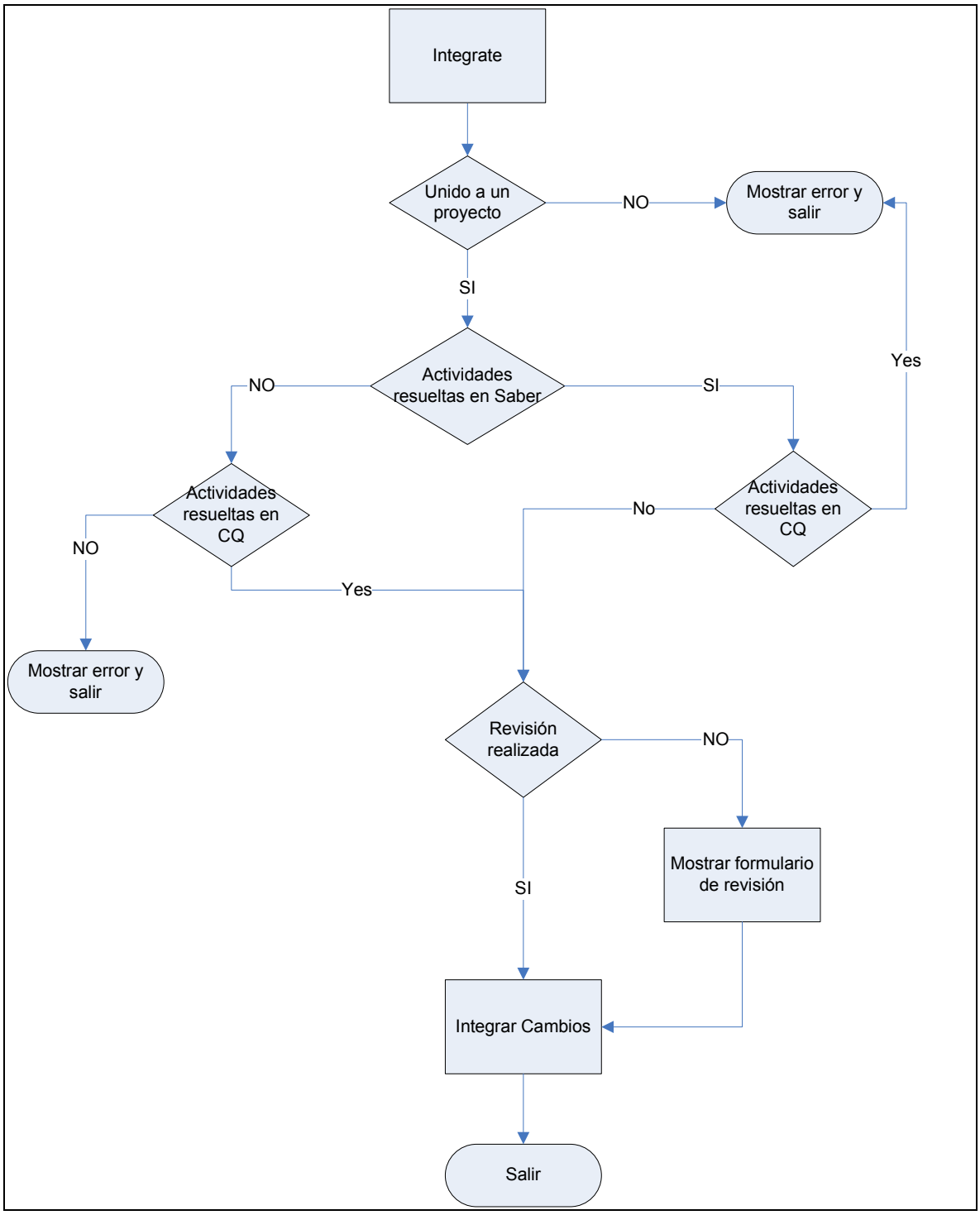

Figura 33

Diagrama de estados del Integrate

3.3.8. Postpone

Esta actividad únicamente tendrá efecto sobre la base de datos de las actividades, tal y como se ve reflejado en la siguiente figura.

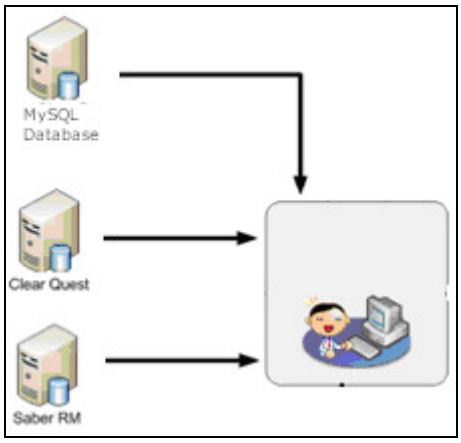

Figura 34

Entorno del postpone

En cualquier momento del proyecto, se puede dar el caso de que el desarrollador esté trabajando en una actividad, y le sea asignada una más prioritaria, en este caso, debido a que como hemos visto al definir nuestro sistema, solo permitimos al desarrollador trabajar en una actividad al mismo tiempo, debemos al menos darle la posibilidad de empezar una nueva tarea sin por ello perder los cambios efectuados en la que haya actualmente en curso.

Con este fin, se ha desarrollado el "Postpone", al ejecutar esta aplicación, todas las modificaciones serán automáticamente guardadas en la rama del desarrollador, almacenando la changelist (número de cambo efectuado en el control de versiones), en la tarea asociada en ClearQuest.

Una vez almacenado el estado en el control de versiones, y cambiada la actividad de "Opened" a "Postponed", el desarrollador puede hacer un nuevo Workon de otra actividad.

Al finalizar esta tarea, el desarrollador podrá reabrir las actividades que tenga pospuestas, o bien trabajar en otra nueva, ambos modos se realizarán interactuando mediante la aplicación de Workon.

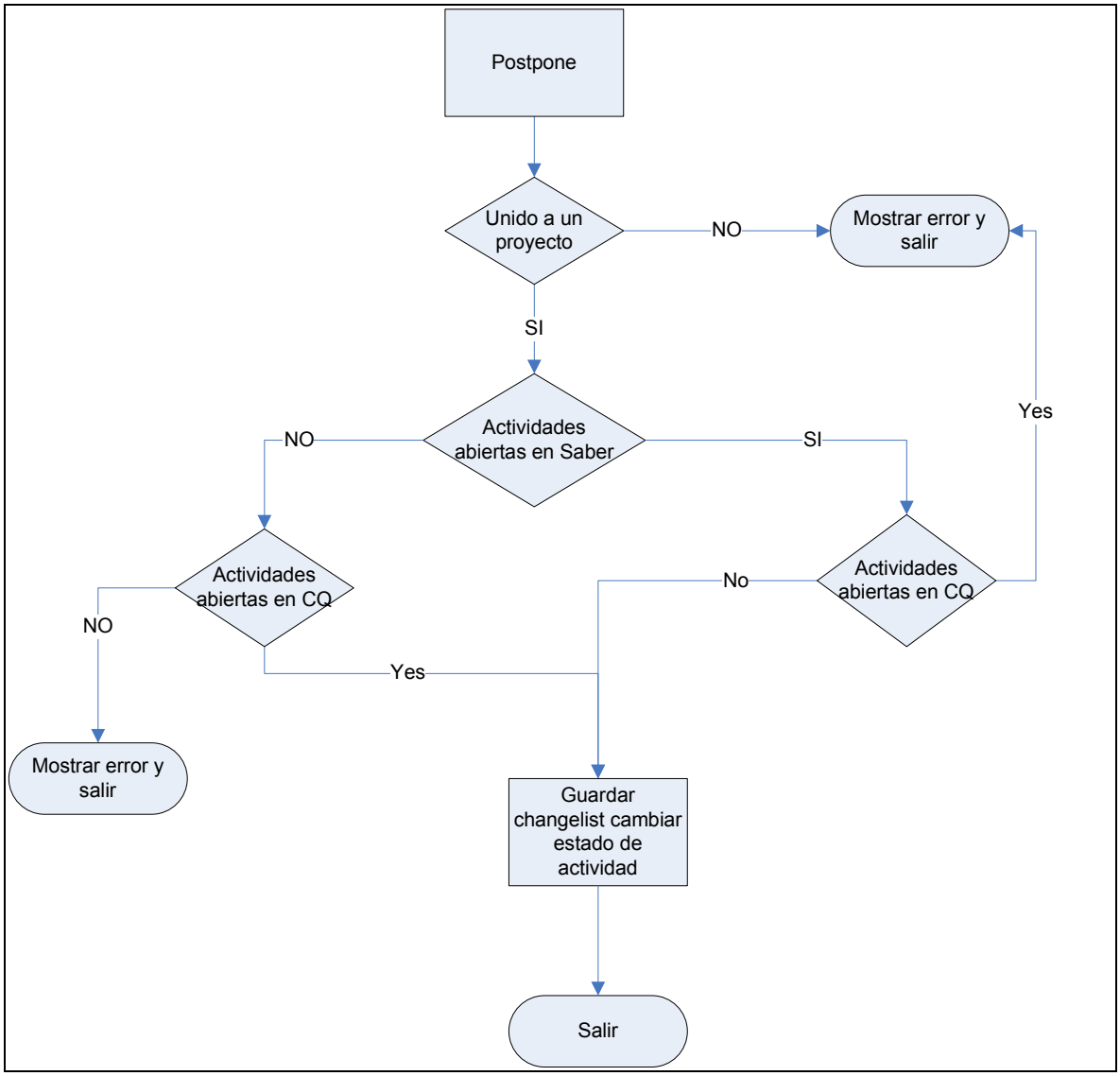

Figura 35

Diagrama de estados del Postpone

3.4. Reports Automáticos

Como hemos dicho anteriormente, periódicamente se hacen releases de la rama de integración, estas versiones estables de código, deben estar debidamente documentadas, por este motivo se ha implementado una herramienta para generar estos reports.

La interfaz de dicha herramienta se muestra en la siguiente figura.

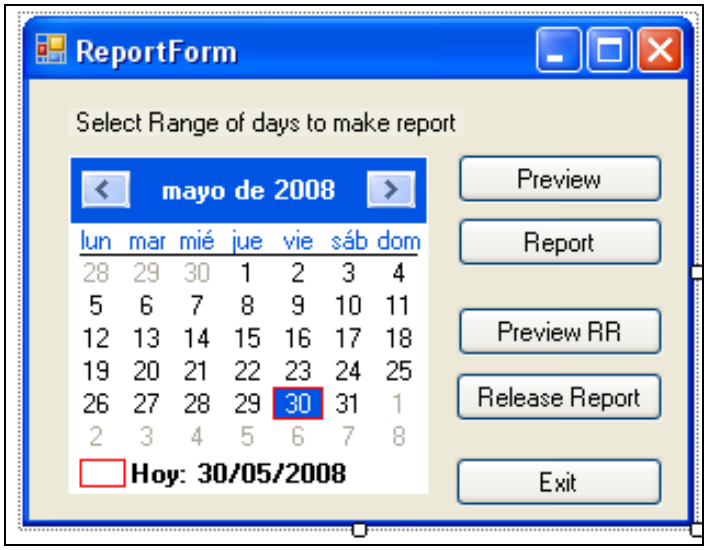

Figura 36

Interfaz de los Reports automáticos

Como podemos observar, dicha aplicación esta formada principalmente por un calendario, de esta forma, se puede seleccionar un rango de dias, y generar un report con las actividades finalizadas o entregadas esos mismos dias.

Por otro lado, al ejecutar la aplicación, se le pueden pasar por parámetro dos fechas, de este modo, se genera el report automáticamente, sin modo gráfico, permitiendo de esta forma, poderse lanzar como una operación más, integrada en la generación de la release, y generando automñaticamente el report.

En un report convencional, se mostrarán las actividades abiertas, resueltas y entregadas durante el rango de fechas determinado, mientras que para el report de la release se mostrarán únicamente las actividades entregadas y las confimadas.

En ambos casos, la información mostrada será, el identificador de la actividad, el desarrollador que la ha implementado, el estado actual de la misma y su descripción.

3.5. Schedule.

	Nombre de tarea	Duración	Comienzo	Fin	Predeces
1	**Herramientas de automatizacio**	**70 días**	**ma 01/01/08**	**mi 09/04/08**	
2	**Fase de diseño**	**9 días**	**ma 01/01/08**	**vi 11/01/08**	
3	Objetivos	3 días	ma 01/01/08	ju 03/01/08	
4	Especificaciones	5 días	vi 04/01/08	ju 10/01/08	3
5	Peer review doc.	1 día	vi 11/01/08	vi 11/01/08	3;4
6	**P4, CQ, Saber**	**58 días**	**vi 11/01/08**	**ju 03/04/08**	**4**
7	**Implementacion**	**35 días**	**vi 11/01/08**	**ju 28/02/08**	
8	Workon	7 días	vi 11/01/08	lu 21/01/08	
9	Checkout	3 días	ma 22/01/08	ju 24/01/08	8
10	Revert	3 días	vi 25/01/08	ma 29/01/08	9
11	Submit	4 días	mi 30/01/08	lu 04/02/08	10
12	Rebase	5 días	ma 05/02/08	lu 11/02/08	11
13	Deliver	3 días	ma 12/02/08	ju 14/02/08	12
14	Integrate	7 días	vi 15/02/08	lu 25/02/08	13
15	Postpone	3 días	ma 26/02/08	ju 28/02/08	14
16	**Testing**	**8 días**	**vi 29/02/08**	**ma 11/03/08**	
17	Peer review	1 día	vi 29/02/08	vi 29/02/08	15
18	Solve errors	2 días	lu 03/03/08	ma 04/03/08	17
19	Testing Application	5 días	mi 05/03/08	ma 11/03/08	18
20	**Deployment**	**15 días**	**mi 12/03/08**	**ju 03/04/08**	**16**
21	Deployment 2 dev	5 días	mi 12/03/08	ma 18/03/08	
22	Deployment 2 dev	5 días	mi 19/03/08	ju 27/03/08	21
23	Deployment all devel	5 días	vi 28/03/08	ju 03/04/08	22
24	**Documentacion**	**3 días**	**mi 12/03/08**	**vi 14/03/08**	**16**
25	Documentos de las	3 días	mi 12/03/08	vi 14/03/08	
26	**Reports**	**4 días**	**lu 17/03/08**	**ju 20/03/08**	**5**
27	Implementation	3 días	lu 17/03/08	mi 19/03/08	
28	Testing	1 día	ju 20/03/08	ju 20/03/08	27
29	**Releases**	**8 días**	**ma 25/03/08**	**ju 03/04/08**	
30	Implementación	6 días	ma 25/03/08	ma 01/04/08	5
31	Testeo	2 días	mi 02/04/08	ju 03/04/08	30
32	**Documentacion**	**4 días**	**vi 04/04/08**	**mi 09/04/08**	**29;26;6**

Figura 37

Schedule de la aplicación

4. Conclusiones y vías de continuidad.

El desarrollo del software, es una actividad que sufre continuas modificaciones, desde cambios en las especificaciones para cada proyecto, hasta el uso de diferentes herramientas, teniendo en cuenta que las tecnologías bajo las que se desarrolla un determinado proyecto, no son las mismas en todos los casos.

Una vez nos hemos introducido en este mundo, vemos que realmente los procesos a automatizar son inimaginables, pero en todos los casos está estrechamente ligado al tipo de proyecto que estemos desarrollando.

Las herramientas de automatización desarrolladas se han implementado utilizando la tecnología .NET de Microsoft, así como objetos COM para la comunicación con el control de versiones, siguiendo un proceso estandarizado de desarrollo que facilita el mantenimiento del proyecto como tal.

Dichas aplicaciones utilizan varios protocolos de comunicación, desde SOAP para interconectarse con las bases de datos de Defectos, como las API's específicas de aplicaciones como ClearQuest o Perforce, así como múltiples lenguajes de bases de datos, como son SQL y SQLAnywhere.

Para todo ello ha sido necesaria una exhaustiva investigación, fundamentalmente en Internet, lo que prueba que el uso de estas aplicaciones no esta muy extendida.

Con el uso de estas herramientas se consigue dar solución al problema de la inestabilidad en las integraciones de Software, se eliminan en gran parte los errores introducidos por el desarrollador en la creación de código, y se proporciona una visión mucho más amplia de los diferentes estados del proyecto.

Además, no solo se han conseguido los objetivos tanto a nivel de implementación como de schedule, sino que se ha implementado una herramienta realmente dinámica, y fácilmente adaptable a muy diversos proyectos encaminándonos de este modo hacia el "Agile development", y aleándonos de una tendencia tan intuitiva como peligrosa, el "extreme programing".

Por otro lado, podemos contemplar el partido que le hemos conseguido sacar a una aplicación, como en nuestro caso sería el control de versiones, trabajando de una forma determinada que el desarrollador manualmente sería incapaz de cumplir rigurosamente, y que en nuestro caso, explota al máximo la herramienta, incluso en ocasiones, obligándola a trabajar con un proceso para la cual no ha sido diseñada.
Viendo el funcionamiento de dichas herramientas en un entorno real, podemos afirmar que el uso de estas herramientas optimiza notablemente la trazabilidad del proyecto, así como su estabilidad y cohesión.

La elección de una u otra política bajo un control de versiones determinado, puede aportarnos unos excelentes resultados o por otro lado producirnos una trazabilidad excelente en el proceso, en nuestro caso, hubiéramos deseado trabajar creando una rama para cada actividad, esto nos hubiera reportado una trazabilidad en el proyecto excelente, de una forma relativamente fácil de administrar bajo un control de versiones como puede ser ClearCase, pero tras haber hecho los estudios coste, implantación del sistema y mantenimiento entre otros, y tras no poder incorporar este control de versiones, no nos ha sido posible implantar la estrategia deseada.

Al escoger Perforce como herramienta a utilizar por su relación calidad-precio, y tras ver que es viable su implantación, hemos redefinido las políticas para el versionado del proyecto, de tal forma que podemos conseguir abarcar los requerimientos del proyecto.

Con esto podemos deducir también que trabajar con un control de versiones o con otro, varían las estrategias a seguir para unos mismos requerimientos, pero en ambos casos, y gracias a las herramientas implementadas, podemos conseguir el mismo resultado.

Cabe tener en cuenta que el proceso de automatización no es un proceso cerrado, con esto, nos referimos a que, si bien este procedimiento se puede seguir para múltiples proyectos, siempre tendremos que tener en cuenta los requerimientos de cada uno para adaptar las herramientas de automatización a las especificaciones que debamos adaptarnos.

Teniendo en cuenta lo dicho anteriormente, podemos observar que estas herramientas son fruto de un fuerte trabajo de investigación, que se debe realizar siempre para

escoger el correcto funcionamiento de estas herramientas dentro de un determinado proyecto.

Habiendo conseguido satisfactoriamente los objetivos fijados, es momento de ampliar la funcionalidad de dichas aplicaciones, o bien plantearnos su uso en diferentes entornos.

Si bien el uso de estas herramientas se encamina hacia una integración continuada, gracias a la implementación de un esquema "push model", cabe ahondar mucho más en ellas para conseguir un proceso cerrado para poder establecerlas como un único producto, pudiéndolas de este modo facilitárselas a un usuario, y que sin necesidad de un administrador, sea capaz de configurarlas por si mismo para hacer que estas trabajen de un determinado modo para un determinado proyecto.

Por otro lado, dichas herramientas, son absolutamente dependientes de un entorno Windows, en el que debe haber instalado .Net Framework y las librerías correspondientes de las bases de datos utilizadas. Puesto que actualmente el desarrollo se encamina hacia la creación de software libre, sería muy importante realizar un estudio para que dichas herramientas funcionen bajo un entorno Linux, actuando sobre un control de versiones de libre distribución, e integrados en una IDE también de libre adquisición.

Posteriormente, deberíamos diseñar un sistema que actualizara automáticamente las versiones instaladas en los desarrolladores de dichas herramientas, evitando así que se deban suministrar manualmente dichas versiones cada vez que se efectúen modificaciones en las herramientas.

5. Bibliografia y referéncias

[1]Wikipedia: http://www.wikipedia.org

[2]Perforce: http://www.perforce.com

Información sobre el control de versiones utilizado para el desarrollo del proyecto en concreto.

API para el desarrollo y las comunicaciones con el control de versiones mediante la librería p4com.dll

[3]IBM: http://www.ibm.com

Información sobre el control de versiones ClearCase.

Información y API para el desarrollo de una aplicación Visual Basic que interactúa con ClearQuest.

[4]Microsoft: http://www.msdn.microsoft.com

Soporte para la implementación de la comunicación sobre un protocolo SOAP

[5]Continous Integration: Improving Software Quality and Reducing Risk

Extracción y studio de las diferentes políticas implementables, riesgos y consideraciones.

[6]Agile Software Development

Extracción de los fundamentos a seguir y las políticas de SCM

6. Anexos

6.1. ClearQuest Datasheet

Automate and control software delivery processes for improved effectiveness

Rational. software

IBM Rational ClearQuest

Highlights

- ***Improve project visibility and control with real-time, consolidated reporting and process enforcement***
- ***Speed software delivery with integrated, automated end-to-end processes***
- ***Improve software quality through enterprise quality management***
- ***Enhance team communication and coordination with automated workflows***
- ***Simplify compliance with access control, electronic signatures, repeatable processes, audit trails and lifecycle traceability***
- ***Scale from small workgroups to geographically distributed enterprises***

Control the software lifecycle and deliver high-quality software, faster

The more versions, team members and sites involved in software delivery, the greater the risk of software quality issues and project delays. Lack of communication, coordination and prioritization across workgroups can affect overall effectiveness. Steps may be missed or completed incorrectly. Disconnected functional teams can generate redundant or inconsistent information. And compliance with internal and external mandates becomes even more difficult to manage.

IBM Rational® ClearQuest® change management software enables you to manage the software lifecycle more effectively. You have access to the information you need to make better decisions. You can more effectively manage tasks and schedules and respond rapidly to customer needs. Automated workflows control and enforce development processes—from requirements definition through production—and help improve team communication and coordination.

By making process enforcement transparent, Rational ClearQuest software enables project teams to stay focused on their ultimate goal—delivering high-quality software on time.

Improve visibility and control

When the different groups and functions involved in a project all use different tools and processes, it's difficult to clearly understand what's occurring at any point in time across the software delivery lifecycle. Disparate systems and processes can also create serious coordination problems, making it tough for teams to collaborate on what should be inherently connected processes. Rational ClearQuest software allows you to customize and enforce consistent development processes and achieve an integrated, consolidated view across the project.

Process automation

Rational ClearQuest software can help you create repeatable, enforceable, predictable processes. Workflows are provided out of the box to jump-start your implementation. Workflows can also be easily customized with the ClearQuest Designer tool to meet your unique, specialized needs. States and actions can be defined, fields can be added and lists can be modified.

Mandatory fields help to ensure the right data is collected. E-mail notifications help improve response time and keep team members informed of changes or updates. By providing control over your processes, Rational ClearQuest software helps make sure nothing falls through the cracks, reducing overall project risk.

Real-time reporting and metrics
Rational ClearQuest software gives you insight into your processes with comprehensive support for querying, charting and reporting. Distribution, trend and aging charts help you visualize complex data. Charts can be easily created and refined to allow you to drill into the area of data that you need.

Individual team members can gain quick access to their prioritized development activities by generating to-do lists. Queries and reports enable you to view the associations of requirements and the status of your test planning, test authoring and test execution activities. Reports can be run out of the box using Business Objects Crystal Reports runtime libraries (included).

Rational ClearQuest software integrates with IBM Rational Portfolio Manager software to align project information with information across the portfolio to help you make more informed business decisions. Integration with IBM Rational ProjectConsole™ software provides a real-time graphical dashboard to view and analyze overall project trends. Additionally, Rational ClearQuest software can integrate with the Microsoft® Project application to help you better manage schedules and resources.

Defect and change tracking
Rational ClearQuest defect and change tracking capabilities help you manage issues throughout the project lifecycle. By documenting and managing issues to resolution, Rational ClearQuest software provides closed-loop change management that delivers better project control and helps improve software quality.

Lifecycle traceability
Rational ClearQuest software integrates tightly with requirements, development, build, test and deployment tools. By linking requirements, code, build records, test cases, deployment records and other development assets, you're able to extend traceability across the entire software delivery lifecycle.

Compress the software delivery lifecycle
Integrating and automating the software lifecycle doesn't just improve project visibility and control. It also helps you accelerate delivery.

Requirements tracking
Requirements records can be replicated directly from IBM Rational RequisitePro® software into Rational ClearQuest software. They can then be associated with other development assets such as defect submissions, build records and test cases. Rational ClearQuest associations are replicated back into the Rational RequisitePro application, providing analysts with connections to Rational ClearQuest information such as activities, change requests and test data.

Activity-based change management
Rational ClearQuest and IBM Rational ClearCase® software work together to help you define and manage changes to software assets as activities. Through the Unified Change Management capability, file versions in the Rational ClearCase application are grouped into logical activities and associated with change requests in the Rational ClearQuest application. This activity-based approach enables developers to manage their work at the task level, instead of managing individual files. It helps build engineers ascertain that the right files are incorporated into the build. Testers can readily confirm that the right functionality and builds are tested. Quality assurance engineers can quickly see and validate what has changed between builds. And project managers are better able to track project status.

Build tracking and automation
Rational ClearQuest software allows you to track and automate the process for building applications. Build records track when a build starts and when it ends; they also track build logs and other build information. Rational ClearQuest software integrates with IBM Rational Build Forge™ software to automate build and release management. During an automated build cycle, Rational Build Forge software can directly create and update Rational ClearQuest build records. An automated build that is usable by the entire team reduces the time spent chasing down compilation and convergence issues. It also reduces errors that delay downstream testing and deployment activities.

Integrated test management
Rational ClearQuest software manages the full range of testing activities from test planning, to test execution, to the capture and analysis of test results. With Rational ClearQuest software, you can define test plans, create test cases and associate test cases with specific test plans. Integration with a variety of test execution tools enables you to map test execution scripts authored in these test tools with test cases, and to initiate test execution. Rational ClearQuest software automatically captures the test results for reporting and analysis.

Deployment tracking and automation
Rational ClearQuest software integrates and automates development and deployment processes. Deployment records track deployments through test environments and into production. For added control, you can establish approval gates before deploying any of these environments. Deployment records can be associated with one another and with other Rational ClearQuest records, such as build records, test cases and test results, to help you track which builds were tested and deployed.

For deployment automation, Rational ClearQuest software integrates with Rational ClearCase and IBM Tivoli® Provisioning Manager software. You're able to deploy approved build files directly from source code control. In addition, you can launch the Tivoli Provisioning Manager software directly from Rational ClearQuest software to facilitate deploying the approved build.

Improve software quality
By automating interactions among disparate project functions and groups, Rational ClearQuest software helps you eliminate software errors that occur because of manual processes and handoffs, miscommunication and inconsistent information. Built-in defect and change tracking capabilities, together with enterprise quality management functionality, unify development and testing activities and prevent skipped steps or incomplete activities. Streamlined processes can lead to faster and more frequent test cycles, allowing you to detect errors earlier in the software delivery cycle. You can avoid late patches and shoehorned fixes that increase costs and delay project completion.

Enhance team collaboration
Rational ClearQuest software helps you improve team communication and coordination by integrating such typically siloed processes as analysis, development, testing and deployment. Automated workflows and e-mail notification help ensure that appropriate team members are immediately alerted when action is required. They also receive complete information about any change or update that can impact their activities. With everyone on the team working from the same information, issues surface quickly and those affected by the issues are able to collaborate on corrective action in real time. Misunderstandings and missed steps can be prevented.

Simplify compliance
In addition to process automation and lifecycle traceability, security features such as user authentication, user authorization, electronic signatures and audit trails are critical to help ensure compliance with internal and external requirements.

User authentication can be performed through the Rational ClearQuest directory of users or through industry standard directory servers using Lightweight Directory Access Protocol (LDAP). Support for Secure Sockets Layer (SSL) encryption allows communication between the Rational ClearQuest software and LDAP directory to be secure.

User authorization is performed through the Rational ClearQuest user database. You can also extend access control for key security checkpoints through scripting. Together, these capabilities help ensure that changes are made only by authorized individuals.

Rational ClearQuest software logs who changed what, when and why. Changes to activities are tracked in the history of the activity. In addition, electronic signatures verify the identities of individuals performing specific actions. By documenting all transactions across the software lifecycle, you can trace the origin and detail of all activities, and verify authorizations and sign-offs.

Unparalleled scalability

Rational ClearQuest deployments can support thousands of users working across dozens of sites. A wide range of access capabilities helps ensure that all team members, local and remote, have access to the most up-to-date information virtually anytime, anywhere. Whether your team is a small workgroup at a single location or a highly distributed team spanning multiple locations, Rational ClearQuest software provides the flexibility and scalability to support your organizational needs.

For more information

To learn more about how IBM Rational ClearQuest software can help you manage and control your development processes for improved effectiveness and delivery speed, visit:

ibm.com/software/rational/offerings/scm.html

IBM Corporation
Software Group
Route 100
Somers, NY 10589

Produced in the United States of America
04-07

6.2. ClearCase Datasheet

Manage and control software assets for faster application delivery

Rational. software

IBM

IBM Rational ClearCase

Highlights

- *Increase productivity with parallel development support, automated workspace management, asset reuse and activity-based change management*
- *Deliver high-quality code with fewer bugs through secure version management and reliable builds*
- *Balance individual and team needs through development and integration stream models, private workspaces and public integration areas*
- *Maintain development flexibility with integrated IDE access, integrations with open source and third-party tools, cross-platform support, remote access and disconnected usage*
- *Scale to accommodate small workgroups as well as large, geographically distributed enterprises*

Balancing development flexibility with effective control of software assets

It's a question of balance. On the one hand, you, as a software developer, prefer to work in a development environment where you have unfettered access to the information and assets you need to create high-quality software. On the other hand, your organization is accountable for delivering business value, which isn't easy because development is as much an art as it is a science. Working with distributed teams, adopting service-oriented architectures (SOAs) and ensuring compliance with an increasing number of mandates all add complexity.

IBM Rational® ClearCase® software can help balance your need for flexibility with the organization's need for control. It provides controlled access to software assets, including requirements, design documents, models, test plans and test results. Parallel development support, automated workspace management and baseline management enable you to create applications rapidly. Secure version management and reliable build auditing help ensure high-quality code. Development and integration models, private workspaces and public integration areas allow you to work independently, yet collaborate effectively with the team. User authentication and audit trails help your organization meet compliance requirements with minimal administrative hassle for you. And with access virtually anytime, anywhere, Rational ClearCase software gives you the freedom to work efficiently where and when you need.

Increase your productivity

Rational ClearCase software can help you get more done in a shorter period of time. You don't waste time working on the wrong versions of code. And support for parallel development, activity-based change management and asset reuse streamlines the development process.

Solution	IBM Rational ClearCase LT	IBM Rational ClearCase	IBM Rational ClearCase MultiSite
Core capabilities			
Version control	■	■	
Automated workspace management	■	■	
Parallel development support	■	■	
Activity-based change management and change sets	■	■	
Support for midrange development	■	■	
Integration with leading IDEs	■	■	
Local, remote (WAN) and Web client access	■	■	
Support for disconnected usage	■	■	
Access control with user authentication and user authorization	■	■	
Audit trails	■	■	
Transparent, real-time access to files and directories		■	
Build auditing		■	
Support for mainframe development		■	
Replication and synchronization of Rational ClearCase repositories			■
Web-based administration console			■
Server architecture			
Single server	■	■	
Multiple, distributed servers		■	
Replicated servers			■

Match the solution to your environment with the IBM Rational ClearCase family of products.

Sophisticated version control

With Rational ClearCase software, you can always be confident that you're working on the right versions of the right artifacts. Rational ClearCase software manages and controls source code, libraries, documentation, binaries, Web artifacts and virtually any project artifact that can be represented as digital content. It also controls versioning for directories, subdirectories and all file system objects. Developers can see the version, branch and file they are working on simply by viewing the Rational ClearCase version tree. In addition, Rational ClearCase software offers access to advanced functions that allow you to delete previous versions, create and delete branches, list version histories, and compare and merge versions.

Parallel development

Rational ClearCase software provides extensive support for parallel development, enabling developers to work on the same code base or release, more easily resolve conflicts and reduce confusion. Automatic file-branching functionality isolates specific changes or versions, allowing multiple developers on the same and different teams to work independently from the same code base. Development and integration stream models define how and when developers deliver code changes. Proven merging and differencing capabilities accept uncontested changes and highlight conflicting changes for faster resolution.

Automated workspace management
With Rational ClearCase software, you have fine-grained control over your personal workspaces and seamless access to the exact file and directory versions you need for different kinds of development activity.

Two types of views, or virtual workspaces, are available. Dynamic views provide transparent access to versions of elements on the network. Snapshot views give you the flexibility to use local copies of file versions while disconnected from the network, and the ability to easily synchronize changes when reconnected to the network. Development teams can mix and match their views based on preferences and project needs.

Activity-based change management
Rational ClearCase and IBM Rational ClearQuest® change management products work together to allow you to define and manage changes to software assets as activities. Through the Unified Change Management capability, file versions in Rational ClearCase software are grouped into logical activities and associated with change requests in Rational ClearQuest software. This activity-based approach enables you to manage your work at the task level, instead of managing individual files. You have a complete view of how development events, including defects and proposed project changes, affect specific files, versions, baselines or releases.

Asset reuse
The less new code you have to create, the faster you can deliver software. The challenge lies in knowing what assets exist, where they reside and how to categorize and reference them in design and development activities. IBM supports the Reusable Asset Specification (RAS) process, which defines a standard way to package reusable software assets. The integration between the Rational ClearCase, IBM Rational Software Architect and IBM Rational Rose® XDE™ Developer solutions makes it easy to search for, locate, download and apply assets from reusable asset repositories located locally or on the Web.

Deliver higher-quality code
Many problems can occur during software development: bugs that have been corrected reappear, previous releases of software are impossible to find or cannot be rebuilt, files mysteriously change or disappear, builds that previously worked suddenly break. The later in your software development cycle that you find a bug, the higher the cost and the greater the risk of delays. Through secure version management and reliable build auditing, and by automating build and release activities, Rational ClearCase software helps to prevent mistakes, reduce bugs and identify errors earlier in the delivery cycle to resolve them more quickly.

Secure version management
Rational ClearCase software provides a robust centralized repository where all development assets are captured and versioned in a secure way. Access control helps ensure that only authorized individuals make changes. User authentication is performed through operating system authentication mechanisms or through industry standard Lightweight Directory Access Protocol (LDAP). Support for user- and group-based permissions limits access to files and directories. User-based locks are available on Rational ClearCase objects (branches, labels, elements and metadata). Programmatic authorization can occur based on the action being performed.

Reliable build auditing
Rational ClearCase software provides effective build auditing. It helps streamline the edit-build-debug cycle and accurately reproduces software versions. Rational ClearCase software also provides the ability to generate a detailed software bill of materials, which can be used to automatically determine when built objects can be reused or shared by developers using multiple views. By detecting dependencies, reusing derived objects wherever possible and producing detailed build audit trails, Rational ClearCase software helps ensure the reproducibility of software versions.

Automated build and release management
To automate the entire build and release management process, Rational ClearCase software integrates with IBM Rational Build Forge™ software. The Rational Build Forge application continuously monitors Rational ClearCase repositories and executes builds either when a change occurs or on a scheduled basis. This automated build capability significantly reduces the time you and your teammates have to spend chasing down compilation and convergence issues. It also reduces errors that can delay downstream testing and deployment activities.

Balance individual and team needs
The development and integration stream models in Rational ClearCase software define how and when developers deliver code changes. Through private developer workspaces and public integration areas, your need to work independently is balanced with the organization's need to effectively integrate your work with that of your teammates.

Maintain development flexibility
Rational ClearCase software gives you freedom of choice. You can work from where you want, using the integrated development environment (IDE) you want, on the platform you want.

Access from anywhere at any time
Rational ClearCase software allows for easy access from various environments and locations. Desktop clients provide complete Rational ClearCase functionality in a flexible Microsoft® Windows® interface. Remote and Web clients enable you to access versioned objects over a wide area network (WAN).

Rational ClearCase software provides integrations with leading IDEs, including the IBM Rational Application Developer for WebSphere® Software environment, the open source Eclipse framework and Microsoft Visual Studio 2005, so you can work within your chosen environment.

Heterogeneous cross-platform support
Rational ClearCase software supports heterogeneous environments and cross-platform development. You can develop in virtually any development environment, including Microsoft Windows, Linux® (distributed and mainframe), UNIX®, Apple Macintosh (via Web browser), and IBM z/OS® environments, as well as the IBM i5/OS, IBM AIX®, Windows and Linux environments on the IBM System i™ platform. On the server side, your organization can choose from a variety of supported platforms to store software assets while you and your teammates continue to work in your preferred IDEs.

Unparalleled scalability
Rational ClearCase deployments can support thousands of users, working at dozens of sites, managing terabytes of data. Whether your team is a small workgroup at a single location or a highly distributed team spanning multiple geographies, Rational ClearCase software provides the scalability you need for your evolving organizational needs.

For more information
To learn more about how IBM Rational ClearCase software can help you manage and control your software assets across the lifecycle, visit:

ibm.com/software/rational/offerings/scm.html

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
03-07

6.3. Perforce Datasheet

Perforce, The Fast Software Configuration Management System Data Sheet

v.2007.3

Interfaces
- Visual client for Windows, Linux, Mac OS X, Solaris, and FreeBSD
- Standard command line across all platforms
- Windows Explorer plug-in
- Web browser with thumbnail viewer
- Integrates with Microsoft Visual Studio, Eclipse, IBM's Rational Software Development Platform, Borland's JBuilder, Apple's Xcode, and more
- Client API for custom applications

Configuration Management
- Client/server repository system
- Atomic change transactions group files changed at the same time
- Flexible locking: supports both concurrent and exclusive file checkout
- Version control of file adds and deletes for total reproducibility
- Full support for version labeling
- Automated change notification
- Graphical view of complete change history over any time period

Digital Asset Management
- Thumbnail viewer handles all major file formats: JPEG, PNG, GIF, BMP, PBM, MNG, PGM, PPM, XBM, and XPM; plug-in API for custom and proprietary formats
- Integrates with popular graphics tools: Adobe Photoshop, Autodesk's 3ds Max and Maya, and Softimage XSI

International Features
- Support for Unicode
- Automatic character set translation across multiple platforms
- Japanese localized GUIs and error messages

Branching
- Unique Inter-File Branching™ model
- Unlimited branching for parallel development and releases
- Comprehensive integration history tracks codeline relationships
- Smart merging/conflict resolution
- Intelligent integrations simplify ladder merges
- Tree-style graph of branch history

Workspace Management
- Unlimited number of workspaces per user/per host
- Client view mechanism selects repository files for workspace
- Workspaces updated on demand
- Support for offline operation

Database and Administration
- Central database tracks user activity
- High-performance, integrated database
- View-based access control for partitioning the repository
- Large repositories (millions of files) can fit on a single server
- A graphical administration tool helps visually manage user and group permissions
- Near zero maintenance, offline checkpointing for 24x7 uptime
- Database journaling for up-to-the-minute recovery
- Plain ASCII checkpoints for easy maintenance
- Supports external user authentication: LDAP, Active Directory

Defect Tracking
- Basic built-in defect tracking
- Defect tracking integration kit
- Integrates with leading defect tracking systems: Atlassian's JIRA, Bugzilla, ExtraView, Fog Creek's FogBugz, HP Quality Center, Serena's TeamTrack, TechExcel's DevTrack, and VA Software's SourceForge

Networking
- True client/server architecture built upon TCP/IP
- No file system mapping or mounting required
- Remote site caching for offsite or offshore teams
- Usable over networks as slow as dialup speeds
- Usable over the Internet or corporate intranets
- Operates through proxies, forwarders, and firewalls

Platforms
- Apple Darwin Power PC
- Apple Darwin x86
- Apple Mac OS X x86
- Cygwin x86
- FreeBSD x86
- FreeBSD x86_64
- FreeBSD SPARC 64
- IBM AIX 53
- Linux x86
- Linux x86_64
- Microsoft Windows IA64
- Microsoft Windows x64
- Microsoft Windows x86
- Sun Solaris 10 x86
- Sun Solaris 10 x86_64
- Sun Solaris 8 SPARC

Supported Filetypes
- Plain text
- Binary
- Symbolic link
- Unicode
- Text with execute bit set
- Binary with execute bit set
- Text with keyword expansion
- Macintosh resource fork

Firmado: Robert Garcia Atserias

Bellaterra, Junio de 2008

Resumen

El Software es fácil de cambiar, demasiado fácil, las herramientas de Software Configuration Management, nos permiten conseguir que un determinado proyecto sea estable y trazable, siempre y cuando estas se usen debidamente. Tampoco el uso de estas herramientas es sencillo. El objetivo de nuestro proyecto es el de implementar una herramienta que haga de nexo entre el desarrollador y todas las herramientas de desarrollo para asegurar que los cambios son consistentes.

Resum

El software és fàcil de canviar, massa fàcil, les eines de Software Configuration Management, ens permetràn que un determinat projecte sigui estable i traçable, sepre que aquestes s'utilitzin degudament. Tampoc utilitzar aquestes eines es senzill. L'objectiu del nostre projecte és el d'implementar una eina que faci de nexe entre el desenvolupador i totes les eines de desenvolupament, per assegurar que els canvis son consistents.

Abstract

Software is easy to change, too easy, Software Configuration Management tools allows specific project to be stable and consistent if managements of these ones is correct. Use these tools is not an easy work. Therefore, the objective of this project is to implement a tool that work as an interface between developer and development tools, to ensure that changes are consistents.