



Máster Universitario en Software Libre

Proyecto Fin de Máster

Curso Académico 2012/2013

Proceso de Desarrollo Iterativo e Incremental a través de SidelabCode Stack

Autor: Ricardo García Fernández

Tutor:

©2013 Ricardo García Fernández - ricardogarfe [at] gmail [dot] com.

This work is licensed under a Creative Commons 3.0 Unported License. To view a copy of this
license visit:

<http://creativecommons.org/licenses/by/3.0/legalcode>.



Índice general

1. Introducción	11
1.1. Etimología	12
1.2. Trabajo en TSCompany	13
2. Motivación	15
3. Objetivos	17
3.1. Proyecto, usuarios, permisos y roles	17
3.2. Carpetas compartidas	17
3.3. ITS	18
3.4. Repositorio de código	18
3.5. Gestión de librerías	19
3.6. TDD Test Driven Development	19
3.7. CI - Integración Continua	19
3.8. Interoperabilidad	20
4. Procesos de desarrollo	21
4.1. Software de Calidad	21
4.2. Proceso iterativo	23
4.3. Proceso incremental	25
4.4. Iterativo e Incremental	27
4.4.1. Fases del proceso	29
4.4.2. Desarrollo Iteración	29
4.5. Gestión de tareas	32

4.6. Código versionado	35
4.6.1. Desarrollo por Ramas	36
4.7. TDD y CI	37
4.7.1. TDD	37
4.7.2. Integración Continua	40
5. ALM Tools - Forjas de desarrollo	43
5.1. Historia	43
5.2. Qué es una forja	45
5.3. Objetivos	46
5.4. Estado del arte de forjas	49
5.4.1. Casos de Uso	50
5.4.2. Tablas comparativas	53
5.4.3. Problemas en algunas forjas	55
5.5. Conclusiones del estudio de forjas	56
6. SCStack	57
6.1. Arquitectura	58
6.1.1. Consola de Administración	58
6.1.2. Servicio web REST	59
6.1.3. API	61
6.2. Aprovisionamiento (Puppet)	61
6.3. Componentes	63
6.3.1. Usuarios, roles y grupos	64
6.3.2. API OpenLDAP	64
6.3.3. Gestión de Requisitos ITS	65
6.3.4. Plugins	67
6.3.5. Repositorios de Código	68
6.3.6. Revisión de Código	70
6.3.7. Integración Continua	72
6.3.8. Gestión de distribuciones y dependencias	75
6.3.9. Desarrollador: Eclipse y Maven	77

<i>ÍNDICE GENERAL</i>	5
6.4. Pruebas y Validación	79
6.4.1. Pruebas instalación	81
7. Comunidades FLOSS	85
8. Desarrollo de un proyecto	87
8.1. Estructura de un proyecto	87
8.1.1. Código fuente	87
8.1.2. Tareas	87
8.1.3. Binarios	87
8.2. Crear un proyecto	88
8.2.1. Proyecto en redmine	88
8.2.2. Repositorio git	88
8.2.3. Jobs en Jenkins	88
8.2.4. Repositorios Archiva	88
8.3. Alta usuarios	88
8.4. Proceso de desarrollo basado en ramas	88
9. Epílogo	89
9.1. Conclusiones	89
9.2. Lecciones aprendidas	89
9.3. Trabajo Futuro	89
Apéndices	90
A. Puppet	93
B. Instalación SidelabCode Stack	95
B.1. Requisitos	95
B.1.1. Descarga SidelabCode Stack	95
B.1.2. Configuración de módulos puppet	96
B.1.3. Apt Cacher	96
B.2. Vagrant	97
B.2.1. Descripción del entorno de instalación	97

B.2.2. Prerequisitos	98
B.2.3. Preparación de la VM	98
B.3. Puppet	99
B.3.1. Pre requisitos	99
B.3.2. Configuración	99
B.4. Post instalación	100
B.4.1. Primeros pasos	100
B.5. FAQ	103
Bibliografía	105

Índice de figuras

4.1. Software de Calidad	22
4.2. Desarrollo Iterativo	24
4.3. Proceso Iterativo	25
4.4. Modelo Incremental	26
4.5. Proceso iterativo e Incremental por Larman	27
4.6. Valor de negocio Iterativo e Incremental	31
4.7. TODO List	33
4.8. Desarrollo por Ramas	36
4.9. Diagrama de los estados del proceso TDD	38
4.10. Ciclo de la Integración Continua	41
4.11. Proceso desarrollo con Integración Continua	42
6.1. Ejemplo de pila a partir de cajones	57
6.2. Diagrama de arquitectura SCStack	58
6.3. Diseño de un API RESTful	60
6.4. Puppet Labs Logo	61
6.5. Ciclo de vida de los módulos Puppet	63
6.6. OpenLDAP logo	64
6.7. Redmine logo	65
6.8. Redmine página de bienvenida	66
6.9. Redmine Backlogs plugin: tareas Redmine agrupadas por estados dentro de una historia de usuario.	68
6.10. Git SCM Logo	69

6.11.Comparación de incrementos entre SVN (completo) y Git (incrementos en archivos)	69
6.12.Gerrit Kunfu Review Cuckoo	70
6.13.Git: Flujo de trabajo centralizado	71
6.14.Jenkins CI Logo	72
6.15.Ciclo de vida Jenkins CI	74
6.16.OpenSSH logo	76
6.17.Apache Archiva logo	76
6.18.Spring Tool Suite logo	77
6.19.Maven logo	78
6.20.SidelabCode Stack Librería API	79
6.21.SidelabCode Stack API lib Análisis de cobertura con Sonar	80
6.22.SidelabCode Stack REST Service Análisis de cobertura con Sonar	80
6.23.Estructura de módulo Puppet de SCStack	81
6.24.Vagrant Logo	82

Índice de cuadros

5.1. Comparación de forjas de desarrollo.	54
---	----

Capítulo 1

Introducción

SidelabCode Stack es una forja de desarrollo de Software para su uso como herramienta **ALM**. Es una herramienta FLOSS (Free Libre Open Source Software) con Licencia **TBC**.



El desarrollo de este proyecto se basa en el diseño e implementación del proceso de desarrollo acorde con las metodologías ágiles a través de la forja *SidelabCode Stack*. Unificando herramientas a través de las distintas APIs basadas en la interoperabilidad, facilitando la instalación, la replicación y la recuperación de los datos mediante el uso de Software Libre para construir Software de calidad.

El uso de metodologías ágiles se encuentra intrínsecamente relacionado a lo largo del proceso de desarrollo e implementación de la forja. Por otra parte podemos afirmar que no

es una herramienta intrusiva para la aplicación de las distintas metodologías ligadas a un proceso de desarrollo.

Presentando las distintas metodologías de desarrollo de software definidas analizando sus características tanto las buenas como las malas y como éstas se aplican al diseño de la herramienta encaminado un proceso de desarrollo fluido y ágil para un desarrollador o un grupo de desarrolladores.

El proceso de desarrollo de software a implementar como base para la herramienta SidelabCode Stack, es el **iterativo e incremental**.

Proceso ágil que se expande sin desviar su orientación al objetivo fijado. Tiene la capacidad de variar el contenido en el camino, tanto a nuevas inclusiones en como a la restricción de otras, mientras que su objetivo final, se mantiene fijo. Es un proceso que se adapta a los cambios, asumiendo en cada iteración nuevas características para transformarlas en resultados.

Un punto a tener en cuenta es la ergonomía con la que cuenta la herramienta para el día a día y las soluciones que aporta con respecto a otras herramientas que cohabitan en el mismo campo.

Todo con un mismo fin; producir software de calidad evaluable, es decir, no es necesario crear un producto perfecto sino se sabe mejorarlo y adaptarlo a nuevas necesidades. Para eso utilizamos las herramientas ALM en los proyectos ya que nos ayudan a tener una visión diaria y a posteriori otorgando una perspectiva para poder medir y evaluar las mejoras entre dos puntos de tiempo. Por lo tanto *la evolución del proyecto y la adaptabilidad a los cambios* como mayor valor.

1.1. Etimología

Sidelab es, un laboratorio de software. Partiendo de esta base, encontramos una definición exacta para SidelabCode ¹:

¹<http://code.sidelab.es/projects/sidelab/wiki/Sidelab>

Sidelab es el "laboratorio de software y entornos de desarrollo integrados"(Software and Integrated Development Environments Laboratory). Es un grupo de entusiastas de la programación con interés en prácticamente todos los aspectos del desarrollo, desde los lenguajes de programación y los algoritmos avanzados, hasta la ingeniería del software y la seguridad informática. Nuestros principales intereses se centran en el desarrollo software y la mejora y personalización de los entornos de desarrollo integrados (IDEs) y herramientas relacionadas.

En el otro extremo del nombre se encuentra *Code* se refiere al código en sí hacia donde se orienta esta herramienta. Por ultimo pero no menos importante tenemos *Stack*, es una pila de servicios para la gestión y el desarrollo de proyectos software.

Como resultado tenemos **SidelabCode Stack** una Forja de desarrollo de aplicaciones orientada a un proceso de desarrollo.

1.2. Trabajo en TSCompany

El prácticum del Máster me dio la oportunidad de entrar a trabajar en la empresa TSCompany para cubrir la plaza de becario. Desde el principio he estado interesado en la producción de software de calidad, en las herramientas e control de versiones, el desarrollo orientado a tests, la documentación y de como unificar las herramientas para obtener un rendimiento óptimo a la hora de desarrollar un proyecto, es decir, un desarrollador que desarrollar al 100 %.

Esta colaboración se basó en la aportación al proyecto de *Software Libre* SidelabCode Stack.

Profesionalmente siempre he tenido ejemplos cercanos acerca del desarrollo de software de calidad y me ha han inculcado el desarrollo orientado a tests *TDD*. Siempre he perseguido la simplicidad y de esta forma la replicación de un entorno, pruebas, instalaciones y la estandarización de la programación para una comprensión sencilla.

El objetivo inicial era pulir el funcionamiento de la herramienta SidelabCode Stack unifi-

cando el proceso de desarrollo en la instalación ya que habían módulos que todavía no estaban conectados, es decir, centralizados, por lo que la funcionalidad quedaba dispersa para el usuario final. Por ejemplo, el usuario tenía que repetir la ejecución de una serie de pasos para poder incluir un proyecto en un repositorio distribuido Git sin que éste tuviera relación con el gestor de tareas, había herramientas que estaban desacopladas.

Había que dar forma al modelo de desarrollo y promocionarlo dentro de la estructura de la forja para que resultase fácil al usuario empezar con el desarrollo de un proyecto dentro del entorno SidelabCode.

Después de la formalización de las distintas versiones del proyecto, se invirtió un tiempo para la implantación de la forja en la empresa TSCompany para su uso diario. Partiendo de la migración de la información de todos los proyectos y usuarios de la empresa para así poder gestionarlos a través de un interfaz común, SidelabCode Stack. Debido al uso de herramientas de Software Libre la migración fue fluida y nada agresiva para los usuarios. El entorno de trabajo seguía siendo el mismo pero en este caso aportando un extra de calidad y métricas para la evaluación de los proyectos desarrollados.

Capítulo 2

Motivación

En el mismo punto, damos la bienvenida a *SidelabCode Stack*, la herramienta ALM para el proceso de desarrollo.

Empezaremos ubicando la forja de desarrollo SidelabCode Stack. ¿ Cual es la motivación que nos lleva a implementar un nuevo ALM ?

Debido a la necesidad de tener un esqueleto para el desarrollo de un proyecto a través del uso de herramientas de Software Libre actuales generando una nueva herramienta de Software Libre.

Partiendo del principio DRY (*Don't Repeat Yourself*) y de la mano de *No reinventar la rueda* el proyecto surgió con la necesidad de crear una forja de desarrollo Libre y usable en distintos entornos.

Después de analizar Las posibles soluciones existentes, se optó por la unificación de las herramientas evaluadas para la generación de una nueva forja.

Después de analizar estas soluciones ALM existentes se optó por crear una nueva Forja a partir de las herramientas necesarias para el proceso de desarrollo de Software de Calidad, siguiendo las metodologías ágiles existentes.

La necesidad de crear una herramienta que aglutinase la gestión de las tareas, el código fuente y la orientación del desarrollo a los Tests. Todo el proceso de desarrollo unificado para facilitar el trabajo.

La integración del proceso de desarrollo en la implementación de una Forja ALM es el punto clave de SidelabCode Stack. Para crear un entorno adecuado se tuvieron en cuenta distintas características en el proceso de creación:

- Ser efectiva: la modificación de una capacidad específica no requerirá el cambio de la plataforma completa, ni un cambio en el despliegue de la solución.
- Ser mantenible y que sus funcionalidades sean fácilmente extensibles y reutilizables.
- Permitir la fácil portabilidad de los datos de los proyectos entre distintas forjas para la migración.

Capítulo 3

Objetivos

Constancia, metodología, facilidad de uso basado en herramientas al alcance de cualquier desarrollador dentro de un proceso de desarrollo para crear software de calidad. Estos son los principios en los que se basa la implementación de SidelabCode Stack como forja de desarrollo teniendo como objetivos tangibles la unificación del uso de las herramientas necesarias para cumplir con éstos como objetivo.

Integrar el desarrollo completo de un proyecto a partir de estos objetivos convertidos en los siguientes requerimientos.

3.1. Proyecto, usuarios, permisos y roles

La gestión del proyecto a partir de la definición de los usuarios, permisos y roles definidos para el mismo. Centralizar esta gestión para así obtener un control fluido y centralizado de los "*poderes*" otorgados según el rango del usuario a través de una herramienta.

3.2. Carpetas compartidas

Generar un espacio de carpetas públicas/privadas para cada uno de los proyectos a partir de los permisos de cada usuario. Asegurando el acceso público/privado a los recursos que

se publiquen.

3.3. ITS

Empezando por el Análisis de Requerimientos en un proyecto es necesaria una herramienta que ayude en el proceso de desarrollo, en este caso un *ITS Issue Tracking System* para la gestión y el mantenimiento de las tareas de cada proyecto. La piedra angular de todo proyecto de software. El ITS otorga un control total del flujo de trabajo y responsabilidades a los usuarios del proyecto, una base de documentación y gestión del tiempo prioritaria en todo proyecto de software.

Utilizar el ITS para gestionar las tareas y los posibles errores dentro de cada proyecto para planificar la corrección o la implementación de éstas, dividiendo las entregas del proyecto en versiones durante el proceso continuo de desarrollo.

3.4. Repositorio de código

El Repositorio de código asociado a cada proyecto. Sin repositorio de código no hay proyecto. De esta forma la cooperación entre usuarios de un proyecto se agiliza bajo el manto de un sistema de control de versiones de código fuente. Esta herramienta produce la información necesaria para generar un histórico de cambios y poder evaluar la evolución del proyecto.

Además se ha de dotar de una herramienta para la Gestión del repositorio. Es decir, la gestión de parches, actualizaciones y desarrollo por ramas dentro de un entorno controlado.

3.5. Gestión de librerías

La gestión de librerías centralizada para los proyectos dentro de la forja, ofrece la posibilidad de interconectar distintas librerías dentro de todo el entorno. Publicando las librerías clasificadas a través de las distintas versiones en un repositorio de librerías común. La reutilización del código y la posibilidad de mejora entre distintos proyectos que compartan librerías mejora la calidad del mismo.

3.6. TDD Test Driven Development

El desarrollo dirigido por Tests *TDD* ayuda a la generación de código de calidad, legible y reutilizable. Siguiendo este proceso, el tiempo dedicado a la replicación de errores disminuye por lo que se puede dedicar más tiempo a la corrección del error en sí.

Otorga datos válidos para el análisis del código y controlar la evolución de la cobertura de Tests para poder evaluar la deuda técnica. Genera los datos suficientes para facilitar la corrección del a misma.

3.7. CI - Integración Continua

El TDD, el desarrollo por ramas y los despliegues en diferentes entornos completan un proceso de desarrollo que culmina en la integración continua del proyecto. Este es el control al más alto nivel dentro de la parte técnica en donde el control de la calidad del código se toma como estandarte. Prevé los errores previos entre distintas versiones del código al unificar las ramas, fallos en los tests y validación de la calidad del código.

Crear despliegues en distintos entornos o replicar entornos similares al entorno final desde esta herramienta para minimizar los errores o poder replicar fácilmente los mismos cuando surjan.

3.8. Interoperabilidad

La interoperabilidad entre todas las herramientas para gestionar la configuración a través de una herramienta centralizada. Este apartado es donde se muestra el potencial de la forja de desarrollo ya que la configuración se expande o replica a las demás herramientas dando vida a la comunicación entre ellas para crear la forja.

Se transforma la configuración inicial y se establecen los protocolos de comunicación entre cada una de las herramientas dentro del proceso de desarrollo habiendo definido los pasos a seguir a partir de las *APIs* existentes.

Capítulo 4

Procesos de desarrollo

Un proceso de desarrollo en el mundo del software se define como:

El proceso de transformación de unos requerimientos en una solución. Un conjunto de pautas a seguir para completar el ciclo de vida de la solución de una manera organizada, sistemática y que ayude a las personas a completar los objetivos fijados

En SidelabCode Stack se optó por implementar el proceso de desarrollo de software *iterativo e incremental* para crear el diseño de la forja SidelabCode Stack a través de metodologías ágiles.

El proceso de desarrollo influye directamente en la calidad del software que se construye. Es la parte más importante en el software ya que un proceso de desarrollo óptimo para una solución otorga las herramientas necesarias para una mejor evolución del mismo. Cada proceso de desarrollo ha de aplicarse a la solución según los requisitos de la misma, no todos los procesos de desarrollo son válidos para todos los proyectos.

4.1. Software de Calidad

Desarrollar Software de Calidad, para ellos nos encontramos con la palabra *Calidad* tan subjetiva en muchos ámbitos, pero que en el desarrollo puede ser bastante objetiva ya

que se trata de Software, una ciencia evaluable. La Calidad del Software es el conjunto de cualidades que lo caracterizan y determinan su viabilidad y utilidad; Mantenibilidad, Fiable, Eficiencia y Seguridad.



Figura 4.1: Software de Calidad

Un software hecho para ejecutarse una sola vez no requiere el mismo nivel de calidad mientras que un software para ser explotado durante un largo necesita ser fiable, seguro, mantenible y flexible para disminuir los costes.

- **Mantenibilidad:** El software debe ser diseñado de tal manera, que permita ajustarlo a los cambios en los requerimientos. Esta característica es crucial, debido al inevitable cambio del contexto en el que se desempeña un software.

- *Fiabilidad*: Incluye varias características además de la fiabilidad, como la aplicación de estándares, complejidad, tratamiento de errores.
- *Eficiencia*: Tiene que ver con el uso eficiente de los recursos que necesita un sistema para su funcionamiento.
- *Seguridad*: La evaluación de la seguridad requiere un control sobre la arquitectura, el diseño y las buenas prácticas.

4.2. Proceso iterativo

¿ Que es el proceso iterativo ?

La primera versión debe contener todos los requerimientos del usuario y lo que se va a hacer en las siguientes versiones es ir mejorando aspectos como la funcionalidad o el tiempo de respuesta.¹

Se centra más en la inmediatez de la primera versión y en las mejoras posteriores que se van creando enfocadas a la solución final. En el proceso también juega una parte fundamental la comunicación con el cliente a través de la visualización de los resultados por iteraciones. De esta forma se consigue una buena coordinación entre el cliente y el equipo de desarrollo para la consecución de los objetivos.

¹Procesos Iterativos e Incrementales - <http://esalas334.blogspot.es/1193761920/>



Figura 4.2: Desarrollo Iterativo

Se han de tener en cuenta posibles cambios entre iteraciones pero nunca del resultado completo, para que de esta forma se pueda controlar a tiempo la *desviación* que pueda existir en el proceso de la creación de producto.

Como la idea que representa la palabra iterativo, un proceso de desarrollo de software iterativo es aquel al que se lo piensa, como una serie de tareas agrupadas en pequeñas etapas repetitivas. Estas "pequeñas etapas repetitivas" son las iteraciones.²

La base el proceso de desarrollo Iterativo provee un conjunto de pasos para el desarrollo de la solución que se repiten iteración tras iteración para la creación de mejoras tangibles y/o evaluables.

²Proceso de Desarrollo Iterativo - Fernando Soriano - <http://fernandosoriano.com.ar/?p=13>

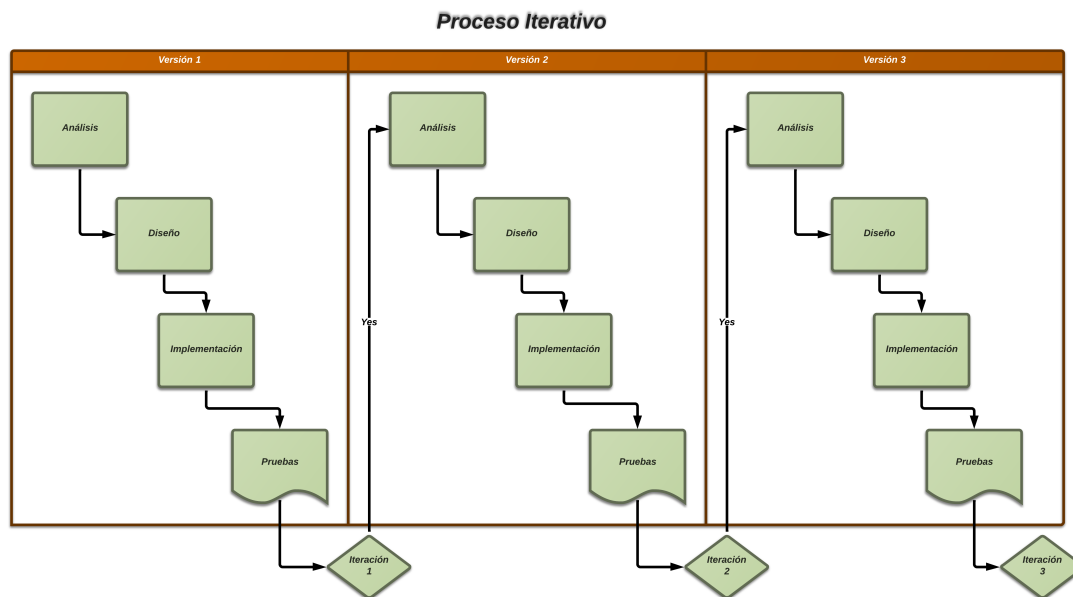


Figura 4.3: Proceso Iterativo

En cada iteración se construye una pieza funcional del producto final, completa, testeada, documentada e integrada en la solución final. La visión completa de este proceso muestra una línea de iteraciones separadas funcionalmente unas de otras que en conjunto, forman la solución final. Iteraciones independientes unas de otras a través de un desarrollo lineal agrupando pequeños ciclos de desarrollo.

- *Duración fija*, quiere decir que una vez establecidos los tiempos o planificación de la iteración, la iteración termina en la fecha exacta establecida. Si el equipo no pudo cumplir lo planificado, el desarrollo pendiente pasa a otra iteración.
- Estimación de tiempos cortos, las *"buenas prácticas"* hablan de que una iteración debiera durar entre 2 y 6 semanas.
- Es como un ciclo de desarrollo completo, ya que en una iteración se realizan actividades de análisis, diseño, implementación, pruebas, etc.

4.3. Proceso incremental

El Proceso Incremental fue propuesto por *Harlan D. Mills* en 1980.

Sugirió el enfoque incremental de desarrollo como una forma de reducir la repetición del trabajo en el proceso de desarrollo y dar oportunidad de retrasar la toma de decisiones en los requisitos hasta adquirir experiencia con el sistema.

El modelo incremental combina elementos del modelo lineal secuencial (aplicados repetidamente) con la filosofía interactiva de construcción de prototipos. El modelo incremental aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario.

Cada secuencia lineal produce un *incremento* en el desarrollo de la solución. Por ejemplo, en relación a la forja SidelabCode Stack; en la primera versión estaba accesible el módulo de Jenkins, en el siguiente incremento la configuración de Jenkins se ligaba automáticamente a la configuración de los usuarios por proyecto, el siguiente incremento se publicaban las instrucciones para gestionar Jenkins a partir de una cuenta y facilitar la configuración para los distintos entornos.

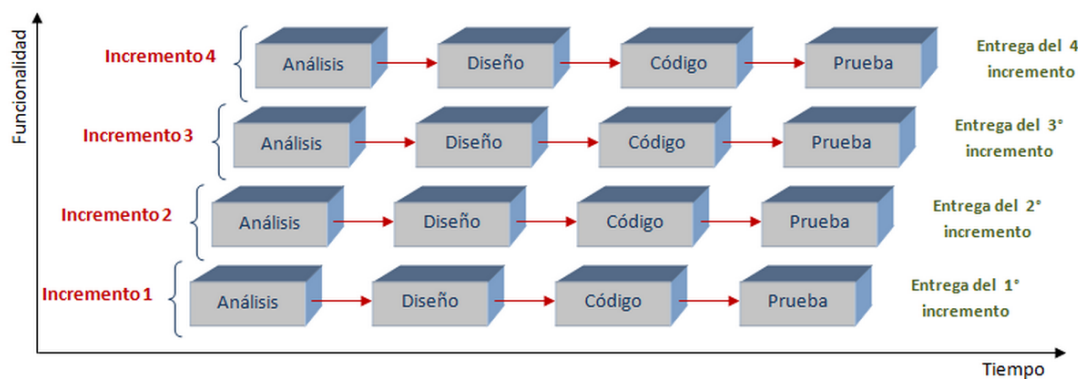


Figura 4.4: Modelo Incremental

Al iniciar el desarrollo, los clientes o los usuarios, identifican a grandes rasgos las funcionalidades que proporcionará el sistema. Se define un bosquejo de requisitos funcionales y será el cliente quien se encarga de priorizar que funcionalidades son más importantes. Con las prioridades definidas, se puede confeccionar el plan de incrementos, en donde cada incremento se compone de un subconjunto de funcionalidades a desarrollar.

4.4. Iterativo e Incremental

Desarrollo iterativo e incremental. La conjunción de estos dos tipos de desarrollo aúnan las mejores cualidades de ambos para gestión de un equipo de trabajo en la construcción de una solución.

El proceso iterativo e incremental se basa en incrementos por cada una de las iteraciones en el proceso de desarrollo. La idea básica de este proceso es desarrollar una solución a través de las iteraciones de ciclos a partir de los incrementos en la funcionalidad para que los desarrolladores mejoren su productividad en torno al proyecto a partir de pequeños hitos que completan versiones usables de la solución desde la primera implementación.

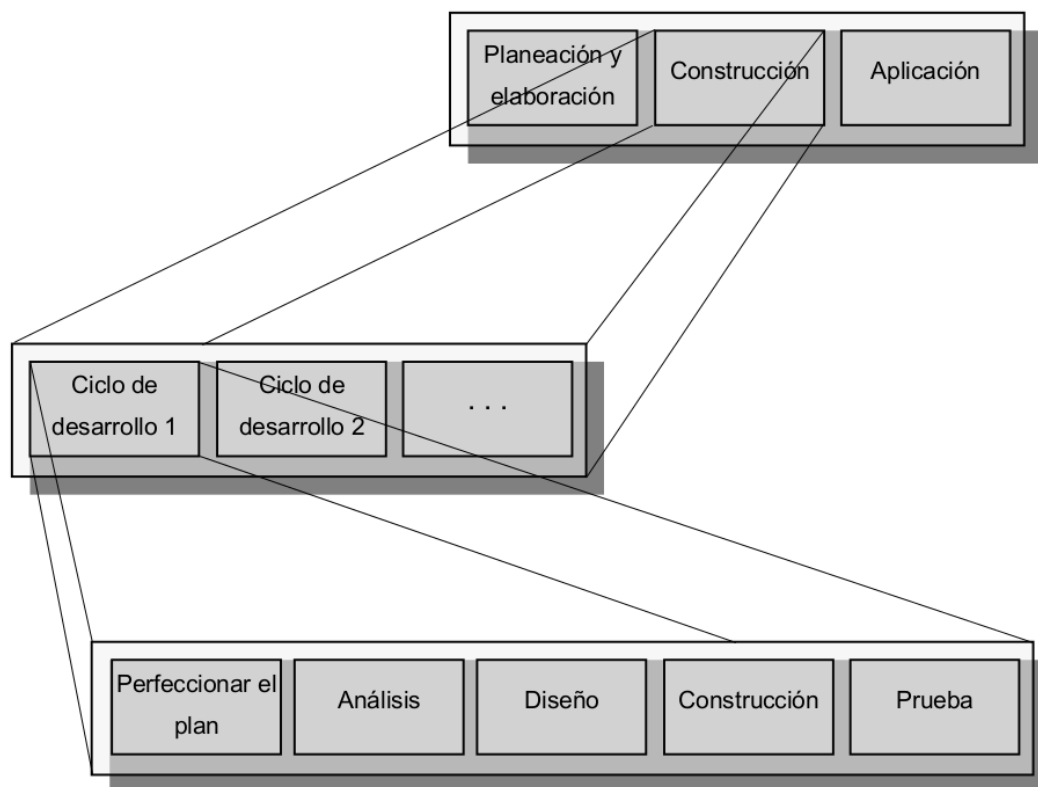


Figura 4.5: Proceso iterativo e Incremental por Larman

La evolución de la solución se basa en las iteraciones pasadas añadiendo los nuevos requisitos/objetivos (pueden ser mejoras o nuevas funcionalidades). Se basa en incrementar

el valor del trabajo hecho para tener un control del proceso más exhaustivo priorizando los objetivos. Por cada iteración existen modificaciones en el diseño y en las funcionalidades.

La comunicación y la implicación en el desarrollo del proyecto con el usuario/cliente desde el inicio del proyecto es crucial ya que el proceso parte desde una solución inicial para incluir versiones usables con el usuario/cliente. De esta forma todas las partes aportan sus distintos puntos de vista de una manera continua implicándose en el proceso y midiendo el crecimiento de la solución paso a paso, incremento a incremento.

Este proceso de desarrollo cercano a las *Metodologías Ágiles*, como ellas tiene el mismo fin, la implicación, desarrollo, fiabilidad, confianza, aprendizaje, versatilidad, responsabilidad, comunicación con la solución desarrollada y el usuario/cliente durante el proceso.

Una de las claves en este proceso es la retroalimentación y el aprendizaje del grupo de trabajo a partir de las iteraciones. De esta forma el trabajo hecho repercute en las iteraciones futuras aportando nuevos conocimientos sobre el proceso y el desarrollo. Creando mejores iteraciones y evoluciones de la solución, *profesionalizando el trabajo*.

Las Iteraciones han de ser de una duración corta de *2 a 6 semanas* para que la comunicación a todos los niveles del proyecto siga siendo fluida y para que si en algún caso se haya de desechar una Iteración no se pierda mucho trabajo desarrollado en ella. Esto no ocurre muy a menudo pero se contempla por diferentes causas:

- Abandono del proyecto.
- Cambio de Cliente/Usuario.
- Falta de recursos.

Las Iteraciones *cortas* otorgan al modelo un alto nivel de versatilidad a la hora de evolucionar y evaluar el recorrido del trabajo hecho en el proyecto, para así poder predecir la organización de las futuras iteraciones.

4.4.1. Fases del proceso

El proceso de desarrollo Iterativo e Incremental está basado en tres fases:

- Iniciación.
- Iteración.
- Lista del Control del Proyecto.

El objetivo de la *Iniciación* es la implementación inicial para crear un producto con el cual el usuario/cliente pueda interactuar y tener las primeras impresiones. El equipo de desarrollo y el usuario/cliente toman como punto base esta fase de *Iniciación*.

Esta primera implementación ha de servir de guía para la evolución del desarrollo en cada una de las iteraciones, la base.

La *Lista de Control del Proyecto* es el lugar donde se definen las tareas que han de cumplirse durante el proceso de desarrollo. Todos los aspectos relacionados con la implementación de la solución se encuentran definidos en la lista, funcionalidades, diseño, errores, mejoras. La Lista de Control está en constante evolución, no es un muro estático, ya que se van adjuntando las funcionalidades y/o mejoras y posibles nuevas funcionalidades. De esta forma se evalúan las prioridades en la fase de análisis por cada iteración y se decide que tareas han de implementarse y cuales son desechadas para la siguiente iteración.

La *Iteración* es un conjunto modular de acciones a llevar a cabo para cumplir con las tareas que se definen para evolucionar la solución por incrementos. Debe estar sujeta a cambios en el diseño, nuevas tareas añadidas a la lista de control y sobretodo, ser simple.

4.4.2. Desarrollo Iteración

El desarrollo del proyecto viene medido por las Iteraciones que se conectan una a otra secuencialmente.

La iteración comienza a partir del análisis basado en la retroalimentación de los usuarios

y los servicios de análisis disponibles. Los elementos a tener en cuenta en el análisis son:

- Estructura.
- Modularidad.
- Ergonomía.
- Eficiencia.
- Objetivos logrados.

Se han tener en cuenta los posibles riesgos que puedan surgir en la Iteración para evaluarlos y eliminarlos en el momento de definir la línea base de la arquitectura. Además el equipo de desarrollo ha de dominar y estar al tanto del lenguaje empleado en los requisitos, el problema que se va a abordar y de esta manera ser capaces de asumir los posibles riesgos o imprevistos que puedan surgir.

Los resultados del análisis se reflejan en la Lista de Control del proyecto para añadir, modificar y ordenar por prioridades para la siguiente Iteración.

En las Iteraciones posteriores ha de aumentar la capacidad de reducir los riesgos, desarrollar los componentes e ir evolucionando incremento a incremento hacia la versión final para el usuario/cliente.

Cada Iteración se reduce a un *miniproyecto* (Se les llama miniproyectos porque no es algo que el usuario haya pedido) que consta del proceso de requisitos, análisis, diseño, implementación y prueba.



Figura 4.6: Valor de negocio Iterativo e Incremental

El proceso de desarrollo en una Iteración se reduce a una serie de guías o pasos a seguir en torno a las modificaciones que surgen a la medida que se avanza en el desarrollo. El proceso Iterativo e Incremental se basa en la flexibilidad por lo tanto las modificaciones de la Iteración han de ser otra herramienta más para lograr los objetivos y como tales:

- Cualquier dificultad encontrada en el diseño, desarrollo y prueba de una modificación puede alertar de la necesidad de cambiar el diseño o la implementación. Se han de desarrollar las modificaciones con una estructura modular y aislada (en su medida) para poder trabajar en un problema o solución concreta y que no afecte al resto de la implementación.
- Las modificaciones han de ser sencillas de implementar, sino se ha de rediseñar el la solución.
- Las modificaciones han de ser más sencillas conforme se van completando iteraciones. Si esto no ocurre existe un problema de diseño y puede incurrir en el exceso

de soluciones *ad-hoc*, parches.

- Los parches se contemplan como soluciones temporales en distintas iteraciones para que no sea necesario un cambio en el diseño, pero sólo para casos excepcionales. Si los parches proliferan se ha de replantear el diseño.
- La implementación existente ha de ser analizada constantemente para certificar que sigue el camino marcado por los objetivos a corto y largo plazo del proyecto. De esta forma se controlan las posibles desviaciones de tiempo y el trabajo hecho por el grupo de trabajo.
- Los herramientas de análisis se han de utilizar para validar los análisis y/o funcionalidades de las implementaciones parciales de la solución.
- La participación del usuario/cliente en el proceso ha de ser solicitada y analizada para contemplar posibles deficiencias o errores en la implementación actual. De esta forma es como se ha de crear un canal de comunicación para interactuar con el grupo de trabajo.

4.5. Gestión de tareas

¿ Qué hay que hacer ? ¿ Quién tiene que hacerlo ? ¿ Cuando hay que hacerlo ?

David Allen propulsor de la metodología *GTD* (Getting Things Done):

El método GTD se basa en la idea de trasladar las tareas y los proyectos previstos de la mente mediante el registro de forma externa y luego dividiéndolos en los elementos de trabajo viables. Esto permite centrar la atención en la adopción de medidas en las tareas, en lugar de en recordarlos

La organización centra la parte más importante en el desarrollo Iterativo e Incremental. Es el apartado donde se ha de dedicar más esfuerzo pero en donde se reciben mejores recompensas al trabajo bien hecho. Cuanto mejor se organiza la información con respecto al proceso, más y mejor crece la capacidad de afrontar nuevas iteraciones en el grupo de trabajo partiendo de la información generada a través del mismo.



Figura 4.7: TODO List

En este apartado se introduce el concepto de *Gestión de Tareas* para una óptima organización. Un gestor de tareas no es otra cosa que una agenda. Limitándonos a la definición de la *RAE* (Real Academia de la lengua Española):

"Libro o cuaderno en que se apunta, para no olvidarlo, aquello que se ha de hacer."

Una agenda es la herramienta que permite apuntar los trabajos, citas, recordatorios, listas de la compra para organizarlos según un orden de prioridad en base a las necesidades y el tiempo.

El Gestor de Tareas nos permite **organizar** las tareas que se han de hacer con respecto al proyecto. Es la herramienta que contiene la *Lista de Control del Proyecto* y la gestiona con respecto a cada iteración para adjuntar las tareas y repartirlas entre los miembros del grupo de trabajo.

El gestor de tareas va un nivel más allá y dota de vida a las tareas creando un seguimien-

to:

- Añadiendo información. En que se basa cada tarea.
- Cambios de encargados de las tareas. Pueden pasar de un usuario a otro.
- Ciclo de vida de las tareas. Desde que se crea hasta que se finaliza (diferentes motivos).
- Evaluar la prioridad. Dando un peso específico a cada una de las tareas dentro del proyecto.
- Visibilidad a todo el grupo de trabajo. Todo el mundo puede acceder a la información dependiente de cada tareas.
- Permite la planificación de la iteración. Desde el nivel más bajo al más alto se puede hacer un seguimiento minucioso del estado del proyecto.

La gestión de la tareas depende de todo el grupo de trabajo, desde el cliente (Lista de Control), el encargado de definir las iteraciones (Obteniendo las tareas de la Lista de Control) y el grupo de desarrolladores (Gestionando las tareas conforme avanza su trabajo). En cada uno de los estados se ha de hacer una planificación de cada una de las tareas y a cada iteración de la solución la planificación será mejor con respecto a la anterior, ya que el histórico ayuda a aprender del trabajo hecho hasta la fecha.

Como características principales de un Gestor de Tareas podemos definir:

- Gestión de recursos.
- Gestión de tiempo.
- Gestión de iteraciones.
- Histórico de información.

4.6. Código versionado

El código fuente de la solución es la clave de la misma por lo tanto ha de existir un mecanismo que permita el control y salvaguarde la información a través de los cambios durante el tiempo. Estamos hablando de un **VCS** (En Inglés *Version Control System*) - *Sistema de Control de Versiones*.

VCS: Version Control System. Un sistema de control de versiones es una herramienta para la gestión de archivos y su ciclo de vida dentro de un proyecto. Gestiona todas las acciones realizadas en los archivos, crear, guardar, copiar, eliminar, mover. La información se refleja en una base de datos y proporciona un histórico dotando una gestión dinámica de los recursos a los usuarios.

Mediante el uso de una herramienta para versionar el código se hace un seguimiento de la evolución del desarrollo de la solución. En este caso pasaremos a utilizar las siglas **SCM** (Source Code Management) - *Gestión del Código Fuente*.

Las herramientas de versionado de código nos permiten asociar las tareas específicas para cada iteración a los cambios efectuados en el código. Agrupar los cambios por usuario, proyecto, fecha, iteración o tarea asociada. Estas dos herramientas multiplican la productividad del grupo de trabajo si se hace un buen uso de ellas, aplicando como es el caso, el uso de la metodología Iterativa e Incremental.

Existen dos tipos de repositorios:

- *Centralizados:* Los usuarios dependen de una estructura centralizada que salvaguarda toda la información. No puede trabajar sin conexión al repositorio.
- *Distribuidos:* Cada usuario gestiona una copia completa del repositorio y unifica los cambios con otros desarrolladores estableciendo un repositorio centralizado. Puede trabajar ya que tiene una copia del repositorio en su poder.

En este caso, se han integrado ambos tipos de repositorio en la forja de desarrollo, como hemos dicho antes, no es excluyente a otros procesos de desarrollo. En el caso que nos ocupa, se utilizará el repositorio de tipo *Distribuido* ya que otorga mayor libertad al grupo de trabajo.

El desarrollo de la solución se adapta al uso del Repositorio distribuido, ya que éste aporta una flexibilidad para la gestión de bifurcaciones del código que en un repositorio centralizado no tenemos fácilmente³.

4.6.1. Desarrollo por Ramas

Los repositorios de código son una herramienta más en el proceso de desarrollo de software por lo tanto hemos de utilizar aquel que se adapte a las necesidades del proceso. En este caso un repositorio distribuido y a su vez, aplicar una metodología de desarrollo a la herramienta.

El *Desarrollo por Ramas* (Feature branch⁴) o *Desarrollo por Canales*.

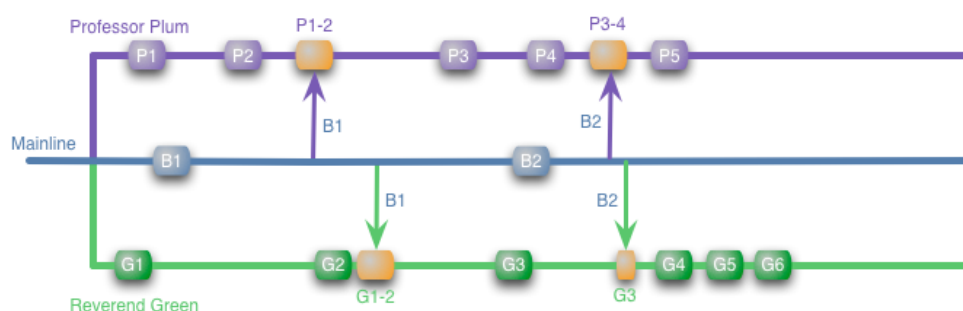


Figura 4.8: Desarrollo por Ramas

El Desarrollo por ramas ayuda a la creación de código de una forma ordenada y coordinada dentro de un grupo de trabajo. Se basa en la creación y la integración continua de nuevas funcionalidades o solventar problemas derivados de otras implementaciones de una manera eficiente, ágil y distribuida por canales.

El desarrollo basado en canales o ramas se ha definido dentro del proceso de desarrollo Iterativo e Incremental en donde ayuda a la integración del código en la rama principal de desarrollo. Como se ha definido en el punto referente al proceso de desarrollo, cada iteración consta de un incremento en el que se han agrupado las tareas (de la Lista de Control) a implementar. Por lo tanto cada tarea se corresponde al desarrollo de una

³Git Svn Comparison - <https://git.wiki.kernel.org/index.php/GitSvnComparison>

⁴Martin Fowler - Feature branch - <http://martinfowler.com/bliki/FeatureBranch.html>

funcionalidad o cobertura de un error. Estamos hablando de desarrollos pequeños y que fácilmente se integran en la rama principal del proyecto.

Divide y Vencerás: resolver un problema difícil, dividiéndolo en partes más simples tantas veces como sea necesario, hasta que la resolución de las partes se torna obvia.

Fragmentar la solución en pequeños módulos organizados facilitando el desarrollo y aislando el ruido y además poder controlar las posibles *excepciones* o *errores* que se produzcan minimizando el daño y el alcance. Pero para ser eficientes ha de existir una herramienta que ayude a coordinar los cambios y el esfuerzo invertido para no desvirtuar el proceso de desarrollo.

4.7. TDD y CI

En este punto se introduce un nuevo apéndice del proceso junto a la herramienta que ayuda a que el desarrollo del código no se desvirtúe mediante el desarrollo por ramas. La *Integración Continua* (CI) y el *Desarrollo Dirigido por Tests* (TDD).

4.7.1. TDD

El TDD se basa en la repetición de un ciclo de desarrollo corto en el que el desarrollador escribe un caso de prueba (test). Este ha de definir el comportamiento que se busca en la nueva funcionalidad y ha de fallar. A partir de este instante ya están definidos los requerimientos de la funcionalidad en caso de prueba y se ha de escribir el código necesario para que a través del test se obtenga el resultado esperado. Acto seguido se ha de refactorizar el código y se da por terminada la tarea.

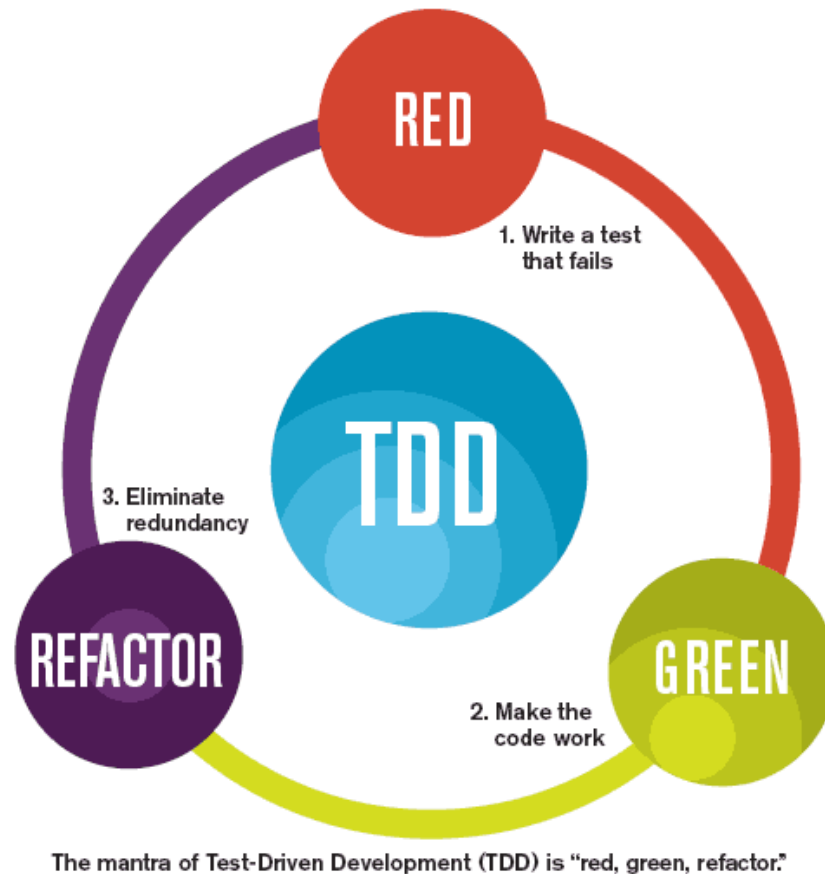


Figura 4.9: Diagrama de los estados del proceso TDD

Este pequeño ciclo de vida se repite constantemente para orientar el desarrollo basado en las necesidades y de esta forma controlar el resultado de los requerimientos y generar una trazabilidad alrededor del código. La solución desarrollada queda integrada en la solución global en donde se incluye, por lo tanto se aumenta la seguridad a través del aislamiento de cada una de las funcionalidades o pasos para que no interfieran entre ellos.

Este proceso de desarrollo proporciona una estructura adecuada para la evolución del código desarrollado ya que facilita el proceso de 'refactoring', control de la deuda técnica en un proyecto y la cobertura de tests.

TDD significa un desarrollo responsable de la solución fraccionando cada implementación dentro de las iteraciones correspondientes. El TDD es *el día a día* de un desarrollador.

Es necesario saber cuando se han de implementar los test ya que hay niveles en los que no existe funcionalidad ni complejidad alguna y *no merece la pena* dedicar esfuerzo a ello.

Como puede ser el caso de los test en métodos *getter* y *setter* en un objeto Java.

Las buenas prácticas para el TDD:

- Tener el código separado de los tests, en carpetas diferentes.
- Los test forman parte de cada iteración.
- Los desarrolladores son responsables de testar el código que escriben.
- Las herramientas y los procesos están altamente automatizados.
- El equipo de QA se encarga de mejorar las herramientas.
- Los tests deben fallar la primera vez que son escritos.
- Los nombre de los tests deben ir acorde con la intención, deben ser nombres expresivos.
- Refactorizar para eliminar código duplicado después de pasar los tests.
- Repetir las pruebas después de cada refactorización.
- Solo se debe escribir nuevo código, cuando algún test falla. Cada test debe comprobar un nuevo comportamiento o diferente.
- Escribe primero el *assert*.
- Minimiza los *asserts* en cada test.
- Todos los tests deben ser pasados antes de escribir otro test.
- Solo se refactoriza cuando todos los tests pasan.
- Escribe el mínimo y simple código para pasar las pruebas.
- No usar las dependencias entre tests. Los tests deben pasar en cualquier orden.
- Los tests deben ser rápidos.
- Usa Mock objects para testear código cuando haya alguna limitación, y así ejecutar los tests más rápido.

4.7.2. Integración Continua

Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible⁵

La *Integración Continua* (CI - Continuous Integration) es un *modelo informático* propuesto inicialmente por *Martin Fowler* que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes.

La CI está basada en dos principios básicos dentro del proceso de desarrollo de software, la *integración de la compilación* y la *ejecución de tests de todo un proyecto*. De esta manera la fiabilidad de la solución desarrollada aumenta y se obtiene una visión a grandes rasgos del código fuente. Permite la localización temprana de bugs (errores), gestionando de una forma eficiente la evolución de la solución desarrollada.

El resultado del proceso se proporciona a los desarrolladores para continuar con el trabajo en la tarea correspondiente:

- Si el resultado es **positivo**: el siguiente paso es cerrar la tarea y empezar con la siguiente.
- Por otra parte, un resultado **negativo**: la herramienta de CI alerta al desarrollador para que evalúe de nuevo la solución desarrollada. A partir de los informes generados el desarrollador tiene más información para subsanar el error. Al volver a completar el ciclo se envía de nuevo el código a la herramienta CI para comprobar el correcto funcionamiento.

Este proceso se automatiza para evitar la repetición de tareas repetitivas (DRY) y agilizar el desarrollo de la solución. Se ha de invertir tiempo para la gestión de esta herramienta ya que si la herramienta no se utiliza correctamente va a traer más dolores de cabeza que

⁵Martin Fowler Continuous Integration - <http://martinfowler.com/articles/continuousIntegration.html>

alegrías.

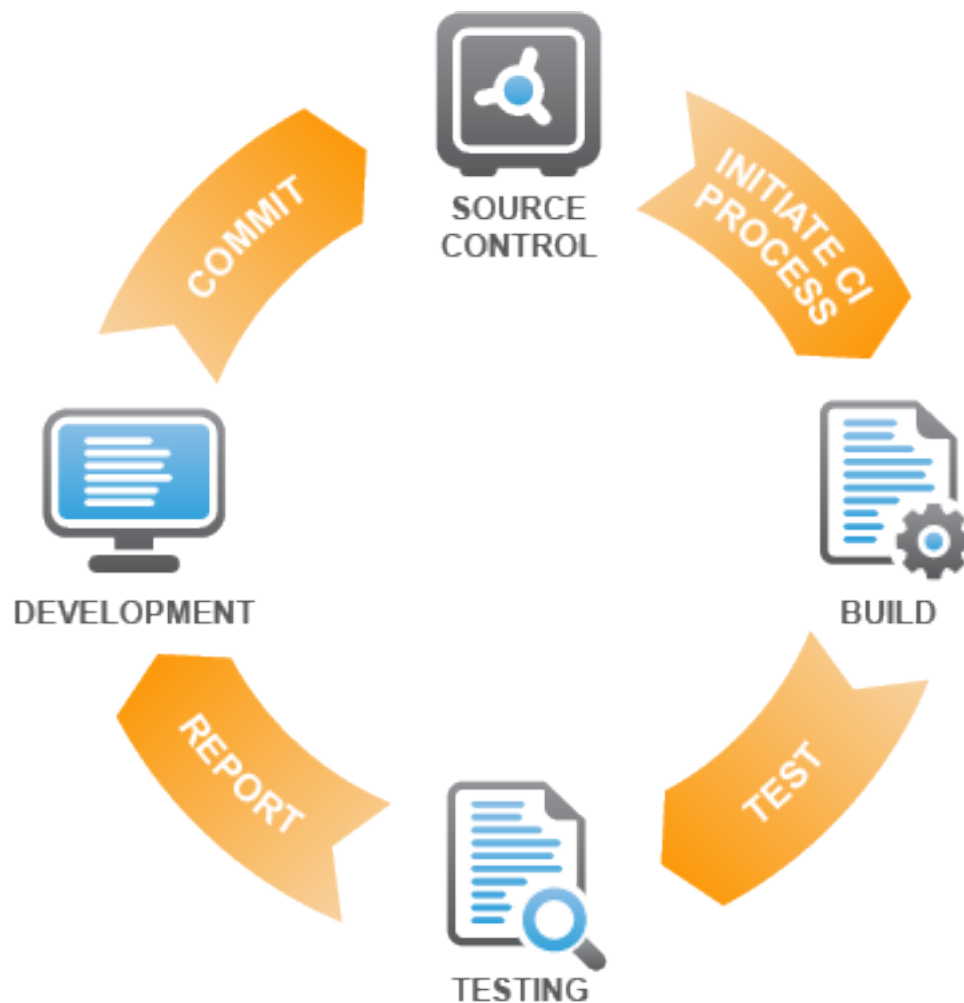


Figura 4.10: Ciclo de la Integración Continua

Este apartado culmina el proceso de desarrollo Iterativo e Incremental ya que en el confluyen todas las herramientas utilizadas unificando los resultados, haciendo un seguimiento de la evolución y generando la solución final por cada iteración.

En el proceso de desarrollo por ramas, cada desarrollador se encarga de su desarrollo asociado a una tarea dentro de la iteración, generación de test e integración en el sistema de CI. A partir de este instante el sistema de CI es el encargado de evaluar el código proporcionado por el desarrollador, evaluando de forma automática cada uno de los pasos descritos:

- Compilación.

- Tests.
- Integración del código fuente en la rama de desarrollo de la iteración.
- Despliegue de la solución.
- Tests de integración.
- Generación de una solución funcional (Continuous Delivery).

Esta nueva iteración dentro del proceso de desarrollo mantiene la viabilidad del proyecto activa a través de una rama de desarrollo preparada para cada iteración. En esta rama se unifican los desarrollos que han pasado las pruebas anteriores desde la compilación a la generación de una solución funcional. *El resultado está listo para entregar al cliente.*

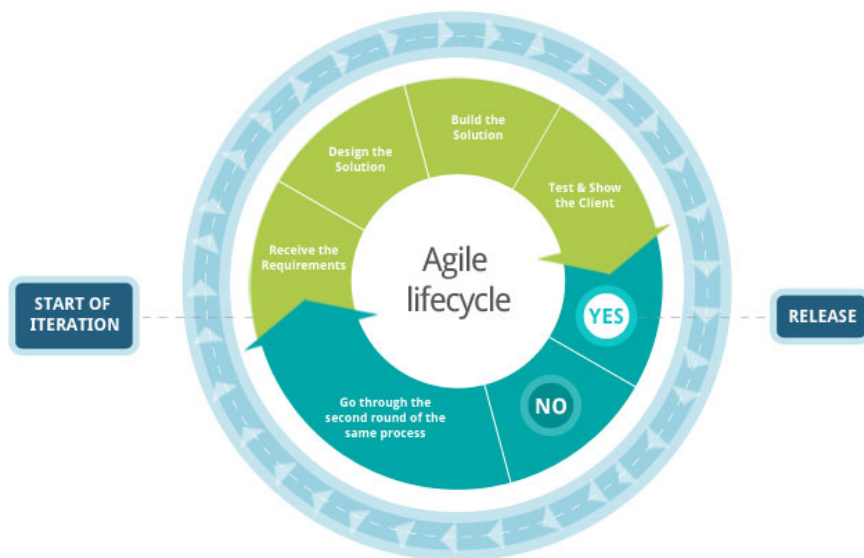


Figura 4.11: Proceso desarrollo con Integración Continua

Se cierra el círculo de una iteración a través de la herramienta de CI automatizando los pasos repetitivos del ciclo particular hasta completar una solución que aúne los requerimientos establecidos en la iteración.

Capítulo 5

ALM Tools - Forjas de desarrollo

ALM Tools: Application Lifecycle Management Tools. La gestión del ciclo de vida de una aplicación, es decir el conjunto de herramientas encargadas de guiar al desarrollador a través de un camino basado en metodologías, que establecen el ciclo de vida, para la desarrollo de un Software hacia su estado del arte.

En el desarrollo de SidelabCode Stack se ha buscado la integración del proceso de desarrollo Iterativo e Incremental a través de las ALM Tools como vehículo. No existe en si mismo una herramienta ALM, sino que la herramienta ALM gobierna a las herramientas que participan en el proceso de desarrollo desarrollando una guía de buenas prácticas.

5.1. Historia

En todas las empresas o comunidades que desarrollan Software siempre se aplica un proceso de desarrollo a la creación del producto. Cada una utiliza distintas herramientas para la gestión de los proyectos de Software, gestor de correo, gestor de incidencias, repositorio de código, integración continua. Pero en la mayoría de los casos de una forma dispar y sin seguir ninguna convención.

A veces el no conocimiento de otras herramientas o la no inclusión de nuevas puede hacer que el desarrollo del proyecto no mejore, partiendo de la base de que el desarrollo puede

ser óptimo para las herramientas utilizadas, carece de perspectivas de mejora a corto plazo.

Si se opta por la integración de una nueva herramienta en el proceso de desarrollo el coste de integración se habría de evaluar ya que se debería dedicar un esfuerzo a la integración ad-hoc de la nueva herramienta para el uso en este mismo entorno con el coste que conlleva, evaluación, test, integración, interoperabilidad, es decir un nuevo proyecto dentro del mismo proyecto.

Después de esta integración en el proceso de desarrollo en la empresa habría que hacer un esfuerzo para salvaguardar la información a través de las distintas herramientas por separado.

El proceso de unificación y reutilización de herramientas a los desarrolladores nos puede parecer familiar si lo comparamos con el uso de los Frameworks a principios de los años 2000. Muchas empresas o comunidades empezaron a adecuar e implementar sus desarrollos en base a un Framework creado por ellos mismos. Estos Frameworks se adecuan a sus requerimientos pero su uso era interno en la empresa y por lo tanto lejano a los estándares. Uno de los más famosos es sin duda el caso de Spring, proyecto de más de 10 años de edad que goza de buena salud y aceptación, incluso equivoca a algunos entre Java y Spring. En este pequeño paralelismo podemos encontrar el estado de las forjas de desarrollo ALM Tools, cuando el proyecto requiere de herramientas para facilitar su ciclo de vida y se van ensamblando una tras otra, que perfectamente las podemos llamar librerías, en una integración **ad-hoc** y siguiendo unos pasos repetitivos en cada nuevo proyecto, en los que humanamente todos nos podemos equivocar debido a que depende de cada uno seguir cada uno de los pasos. Estas herramientas tienden a convertirse en pequeños estándares dentro de cada grupo de desarrolladores y a repetirse en futuros proyectos, pero debido a los desarrollos **ad-hoc** carecen de escalabilidad e integración con nuevas soluciones de una forma ágil, es un escollo actualizar y por otro lado replicar un estándar para la implantación de la forja, no se tiende a dejar puertas abiertas para que más adelante el herramienta mejore. Se podría definir como uno de los casos de inanición en el desarrollo de Software o muerte por éxito.

En este punto es donde entra la famosa interoperabilidad entre las herramientas que define

una comunicación estándar. El punto clave de las ALM Tools, la **integración e interacción de las herramientas** dentro de un proceso de desarrollo.

En post de evitar la falta de replicación, además de la importancia de la interoperabilidad, se ha de tener en cuenta la replicación del contenido o la gestión de la instalación de un ALM. Siempre se ha de pensar mirando un paso por delante. No es necesario implementar las mejoras pero sí, dejar un espacio o conector para que casen bien. Un ejemplo que puede ilustrar esta frase es la programación basada en Interfaces en Java, ya que las Interfaces ofrecen soluciones para implementar a medida a partir de un *esqueleto*, si se actualiza la Interfaz (en este caso es el esqueleto de la clase) para añadir una nueva funcionalidad con un método, los métodos anteriores mantienen su comportamiento dentro de cada clase que la implementa y adquieren la posibilidad de aumentar la funcionalidad implementando la nueva solución, adecuada a su entorno, de esta forma la interoperabilidad entre las clases que utilicen esta Interfaz también se mantiene ya que todas implementan las funcionalidades de la Interfaz como clase en la que pivotan.

5.2. Qué es una forja

Partimos de la definición que ofrece *Cenatic* en el título de su estudio sobre forjas:

Entorno de desarrollo colaborativo de Software

Una forja de desarrollo es una herramienta que actúa como elemento catalizador de este proceso abierto de desarrollo. Las forjas juegan un papel clave para aprovechar las ventajas de las metodologías, en este caso a través del proceso de desarrollo Iterativo e Incremental, aportando múltiples ventajas:

- Mayor eficiencia.
- Mejor calidad en el producto final.
- Reutilización de esfuerzos.
- Modularidad.
- Adaptación a estándares.

- Mayor agilidad en el proceso de desarrollo.

Esta herramienta permite centrar toda la atención y potencial del desarrollador en el mismo desarrollo. Facilita la evolución del código desarrollado y el seguimiento del proyecto a todos los niveles.

Además las forjas de desarrollo aportan la sencillez para instaurar este entorno de desarrollo en el grupo de trabajo. Facilita la instalación, la replicación de contenido, la comunicación y la publicación de resultados del desarrollo de cada proyecto al alcance de los usuarios de la forja.

Una forja es una comunidad en donde convergen proyectos de software (en el caso que nos atañe) a través de las interacciones de los usuarios con el código. Toda la información gira en torno al repositorio de código. El valor aportado por una forja en pos del repositorio por si sólo es la interoperabilidad con otras herramientas para la mejora de la gestión del desarrollo del proyecto; manuales, tickets, diagramas, planificaciones, wiki, revisiones, roles.

Tener las herramientas necesarias no es suficiente para obtener un desarrollo fluido, fiable y deseable. Se han de conocer las herramientas que componen la forja, no las herramientas en si, sino su funcionalidad, para que de esta forma se pueda definir un proceso de desarrollo basado en las herramientas existentes o añadiendo nuevas herramientas para que se adapten al proceso de desarrollo elegido.

La forja ha de trabajar para los usuarios, pero los usuarios han de saber como hacer que la forja trabaje para ellos.

5.3. Objetivos

El objetivo principal de una forja de desarrollo es poder sacar el máximo partido al desarrollo del proyecto ayudando a establecer el proceso de desarrollo elegido para cada proyecto dentro de un entorno integrado de trabajo.

Disponer de un entorno integrado de trabajo en el que concentrar los esfuerzos de usuarios

permite también obtener una serie de ventajas desde el punto de vista de la potenciación del propio proceso de desarrollo, desde varios puntos de vista:

- *Mejor control de esfuerzos:* Ya que todos los usuarios están en contacto a través del entorno de la forja, los responsables de asignación de tareas encuentran más facilidades para identificar los miembros más adecuados para resolver determinadas necesidades, así como detectar de forma temprana cualquier problema importante que pueda comenzar a surgir dentro del proyecto, desde el punto de vista de recursos necesarios para sus mantenibilidad.
- *Aspectos legales y de licencias:* Dentro de la forja, hay cabida para poder indicar de forma clara y explícita toda la información necesaria sobre licencias bajo las que se distribuye los productos del proyecto. Este es un aspecto muy importante para poder garantizar la compatibilidad de los productos del proyecto con otras soluciones desarrolladas en otras iniciativas, evitando la posterior aparición de problemas legales o incompatibilidades en fases de integración más avanzadas.
- *Homogeneizar las prácticas y estilo desarrollo:* En proyectos de desarrollo la documentación de la forja y el establecimiento por parte de los usuarios de una serie de directrices que marquen el proceso de desarrollo, las herramientas que se debe utilizar, así como la política que debe seguirse en diferentes estratos de los flujos de trabajo es crucial para mantener un entorno de trabajo efectivo y homogéneo que asegure la cohesión de los diferentes elementos generados para dar como resultado productos mucho más estables, integrados y de mayor calidad.
- *Establecimiento de guías de evolución:* La forja es el entorno ideal para poder anunciar y consensuar entre todos los usuarios una guía de evolución clara del proyecto, no solo desde el punto de vista del desarrollo formal de código, sino también de los objetivos generales y de utilidad que se persiguen dentro de la iniciativa para así poder mejorar el mismo proceso de desarrollo.

El proceso Iterativo e Incremental requiere un seguimiento constante de cada una de las iteraciones a partir de los objetivos establecidos en la Lista de Control.

- En cada iteración se ha de desarrollar la solución asociada a cada requerimiento

basando el desarrollo en la orientación a test (TDD) partiendo de una nueva rama de desarrollo.

- Acto seguido el resultado se ha de implantar en la rama de desarrollo de la iteración, por lo que se maneja a través del servidor de integración continua para poder adjuntar el desarrollo a la iteración dependiendo si el resultado ha sido positivo o no.
- De esta manera se completan los pequeños ciclos de vida que ha de manejar la forja.

La gestión del ciclo de vida del proceso de desarrollo se divide en el uso de las siguientes herramientas que componen la forja ALM *SidelabCode Stack*:

- *Usuarios y Roles* - Gestión de los usuarios, permisos y roles para cada proyecto a través de un directorio de identificación.
- *Gestión de Requisitos* - A través de un *Issue tracking system*. Herramienta para gestionar los requisitos y estado actual de cada uno de ellos accesible a los desarrolladores del proyecto.
- *Gestión de Repositorios* - Herramienta para la gestión de repositorios; centralizados o distribuidos.
- *Ciclo de vida* - Desarrollo y seguimiento de las soluciones mediante el ciclo de vida establecido; requisitos, test, desarrollo, integración continua.
- *Gestión de Despliegues* - después de cada iteración finalizada y publicación del resultado.

La inclusión de estas herramientas para facilitar el proceso de desarrollo iterativo e incremental a través del uso de la forja SidelabCode Stack es lo que permite definir cada comportamiento dentro del proceso mediante la comunicación entre cada una de ellas.

Esta definición del proceso de desarrollo se plasma en la guía que proporciona SidelabCode Stack para el desarrollo iterativo e incremental.

5.4. Estado del arte de forjas

Hoy en día las forjas ALM abundan, además de gozar de una gran popularidad entre los proyectos de Software Libre, como podemos ver en los casos de SourceForge, Googlecode, Bitbucket y Github (más adelante discutiremos cada proyecto). En este caso como *SaaS* (Software as a Service) debido al servicio que ofrecen. El punto diferenciador se encuentra en las forjas ALM las que son *FLOSS*, ya que permiten replicar ese mismo entorno en tu propia máquina o poder trasladar el proyecto de una herramienta a otra. Un dato muy importante a tener en cuenta, porque siempre se ha de mirar hacia adelante. Sobretudo destacan porque si se crea una necesidad con respecto a la gestión, al ser *FLOSS* siempre se puede implementar con el propio conocimiento del desarrollador a diferencia de las que no son *FLOSS*. En este caso el usuario *depende de la propia compañía*, un tercero que es el único que puede implementar la solución convirtiendo el servicio en *Vendor-Locking*.

En esta lista se muestran las forjas de desarrollo ALM más populares:

- SourceForge con Allura - <http://sourceforge.net/projects/allura/> - **FLOSS**.
- Cloudbees DEV@Cloud <http://www.cloudbees.com/dev.cb> - **SaaS privado**.
- CollabNet con CloudForge <http://www.cloudforge.com/> - **SaaS privado**.
- Plan.io - <http://plan.io/en/> - **SaaS privado**.
- ClinkerHQ <http://clinkerhq.com/> - **SaaS privado**.
- Github - <http://github.com/> - **SaaS privado**.
- GitlabHQ - <http://gitlab.org/> - **FLOSS**.
- GForge - <http://gforge.org/gf/> - **FLOSS**.
- Collab.net - [urlhttp://www.collab.net/](http://www.collab.net/) - **SaaS privado**.
- Google Code - <http://code.google.com/intl/en/>.
- Bitbucket - <https://bitbucket.org/> - **SaaS privado**.

5.4.1. Casos de Uso

Nos vamos a servir de la investigación hecha por *Cenatic* [3] sobre el uso de las forjas y el estudio sobre el incremento de la productividad por *Dirk Riehle* [4] en la implantación de una forja de desarrollo.

A partir de estos estudios el caso se muestran los casos de *SAP Forge* basado en *GForge* y *GoogleCode* con su propia forja. Dos casos de éxito de implantación de una forja desde dos planteamientos diferentes en el uso: internamente y como servicio para proyectos FLOSS (además de internamente).

SAP

En el estudio sobre el aumento de la productividad [4] nos presenta un detalladamente el caso de utilización de una forja para el lanzamiento y desarrollo de proyectos software dentro de la intranet empresarial. En este caso, se eligió *GForge* como paquete para proporcionar las funcionalidades básicas de una forja, sobre el que se elaboró un entorno ligeramente más personalizado. De este modo, se presenta la forja de *SAP* como un entorno de desarrollo colaborativo, centralizado, y fácilmente accesible por cualquier miembro de la empresa que desee participar, gracias a su acceso a través de una URL interna y de fácil memorización. Es importante remarcar que cualquier trabajador dentro de los límites de la red interna puede acceder a la forja.

Tras su primer año de andadura, según [4] *SAP Forge* había atraído más de 100 proyectos y unos 500 usuarios registrados, lo que aproximadamente representa el 5 % de la población total de desarrolladores *SAP*. La inclusión de un proyecto dentro de la forja no era obligatorio, sino potestad del líder del proyecto.

Como datos a destacar en este análisis hay tres características que han de ser mencionadas después de la puesta en marcha del proyecto *SAP Forge*:

- **Búsqueda de proyectos:** Un recurso para los usuarios en el que están indexados todos los proyectos a partir de sus metadatos: nombre, descripción, etc. Todos los proyectos están accesibles para los usuarios (miembros de la empresa) para la con-

sulta. Aplicando el modelo de colaboración abierta del Software Libre, cuantos más ojos interesados, mejores resultados.

- Información de los desarrolladores: Se genera una base de datos de conocimientos asociados a cada uno de los desarrolladores que son usuarios de la forja. Disponen de una lista de aptitudes obtenida a partir de la información que genera la forja de desarrollo. Esta información facilita la elección de, según los requisitos, poder encontrar a la persona idónea para implementar la solución, cercar las búsquedas y obtener mejores referencias de los trabajadores.
- Publicidad del proyecto: Se puede hacer un seguimiento de la vida del proyecto a partir de los usuarios, listas de correo, tráfico generado. De esta forma se pueden encontrar los proyectos más usados, útiles y activos, definiendo intereses y reconociendo patrones de casos de éxito aplicables a otros proyectos.

El texto [4] nos expone el caso de un proyecto (*Mobile Retail Demo*) que se introdujo en la *SAP Forge*. A partir de este movimiento el interés y la colaboración en el proyecto se incrementaron por parte de los desarrolladores. El uso, éxito y la calidad crecieron exponencialmente ligados a la colaboración que había facilitado la forja a los usuarios.

Con unas mediciones más exactas basadas en una encuesta interna a los desarrolladores participantes en los proyectos de la forja, se puede ver reflejado el grado de satisfacción por los resultados obtenidos en cuanto al desarrollo de los proyectos y la colaboración entre ellos:

De un total de 83 participantes en la encuesta, un 66 % indicó que habían utilizado las herramientas de búsqueda de proyectos de la SAP Forge, para localizar otros proyectos que fuesen de su interés. Un 24 % indicaron que su proyecto había recibido ayuda del exterior (dentro del ámbito interno de SAP), principalmente en forma de reportes de error y sugerencias de mejora. Finalmente, un 12 % de los encuestados comentaron que finalmente colaboraron con otros proyectos diferentes basándose en sus preferencias personales para realizar la selección.

GoogleCode

Este es un ejemplo del uso de una forja como servicio externo a la compañía. Google proporciona su propia forja de desarrollo para que los usuarios la utilicen bajo la condición de que los proyectos sean FLOSS asociando un tipo *específico* de Licencia: Apache License 2.0, Artistic License/GPLv2, GNU General Public License 2.0, GNU Lesser Public License, MIT License, Mozilla Public License 1.1, New BSD License.

A diferencia de una forja de uso interno, la colaboración entre proyectos se multiplica debido a que el número de usuarios "se dispara" para dar paso a una gran cantidad de distintas situaciones e intercambio de información mediante la colaboración.

GoogleCode es el proyecto insignia de Google en el ámbito de las forjas de desarrollo de código. Se trata de un proyecto para proporcionar espacio web y herramientas de soporte al desarrollo colaborativo de software, abierto a cualquier grupo o desarrollador individual interesado en publicitar su proyecto.

El proyecto se publicó en *Julio de 2006* como plataforma y ha recibido un asombroso número de peticiones de alojamiento. Según los datos publicados por Google en 2009 [6] se pueden encontrar más de 250.000 proyectos alojados en la forja de *GoogleCode*.

GoogleCode ofrece distintas herramientas para el desarrollo del ciclo del proyecto:

- Gestión de tareas mediante un ITS (Issue Tracking System).
- Documentación a través de una Wiki.
- Repositorio: Centralizados y Distribuidos - Subversion, Mercurial o Git.
- Descarga de binarios.

GoogleCode es en la actualidad uno de los repositorios más consultados a la hora de localizar librerías y aplicaciones ya existentes.

5.4.2. Tablas comparativas

A partir de los datos obtenidos de la tabla comparativa sobre forjas de desarrollo en wikipedia [7] se obtiene una visión más amplia de los componentes comunes en las forjas de desarrollo más conocidas:

Name	Code review	Bug tracking	Web hosting	Wiki	Translation system	Shell server	Mailing List	Forum	Personal branch	Private branch	Announce	Build system	Team
Alioth (Debian)	no	yes	yes	no	no	yes	yes	yes	yes	yes	yes	no	no
Assembla	yes	yes	yes	yes	yes	no	no	no	yes	yes	yes	yes	yes
BerriOS	?	yes	yes	yes	?	yes	yes	yes	?	?	yes	?	?
Bitbucket	yes	yes	no	yes	no	no	no	no	yes	partial Yes	no	no	yes
CloudForge	?	yes	no	yes	no	no	no	no	?	?	?	?	?
CodeHaus	no	yes	no	yes	no	no	yes	no	no	no	no	yes	?
CodePlex	no	yes	no	yes	no	no	yes	yes	no	no	no	no	no
Fedora Hosted	yes	yes	no	yes	no	no	no	no	no	no	no	no	no
GitHub	yes	yes	yes	yes	no	no	no	no	yes	partial Yes	no	no	yes
Gitorious	yes	no	no	yes	no	no	no	no	yes	no	no	no	yes
Gnal	?	yes	yes	no	yes	?	yes	no	?	no	?	No	?
GNU Savannah	yes	yes	yes	no	no	yes	yes	no	no	no	yes	no	yes
Google Code	yes	yes	partial Yes	yes	no	no	partial Yes	no	partial Yes	no	no	no	no

Cuadro 5.1: Comparación de forjas de desarrollo.

Una lista extra sobre forjas actuales (en las que no se basó el análisis):

- Cloudbees DEV@Cloud <http://www.cloudbees.com/dev.cb> - **SaaS privado**.
- CollabNet con CloudForge <http://www.cloudforge.com/> - **SaaS privado**.
- Plan.io - <http://plan.io/en/> - **SaaS privado**.
- ClinkerHQ <http://clinkerhq.com/> - **SaaS privado**.
- GitlabHQ - <http://gitlab.org/> - **FLOSS**.
- GForge - <http://gforge.org/gf/> - **FLOSS**.
- Collab.net - <http://www.collab.net/> - **SaaS privado**.

En la tabla se aprecia que las herramientas que más se utilizan en las forjas de desarrollo son el ITS y la wiki. Mientras que los sistemas de traducción y de construcción pasan al último escalafón. La construcción de la solución está presente en todas las fases del proceso Iterativo e Incremental ya que el desarrollo se basa en las construcciones previas, por lo tanto es más que necesaria una herramienta de construcción integrada en la forja.

5.4.3. Problemas en algunas forjas

Los requisitos más importante a la hora de desarrollar la Forja SidelabCode Stack para adaptar el proceso de desarrollo Iterativo e Incremental fueron:

- La gestión de la información a través del ITS.
- Gratuidad en la gestión de los repositorios privados y públicos.
- La capacidad de tener un entorno instalado en un servidor propio.
- Desarrollar nuevas funcionalidades.
- Modularizar componentes.
- Interconectar nuevos módulos.

Partiendo de estos primeros requisitos las Forjas existentes Github, Bitbucket, Collab.net, Cloudbees, Plan.io, ClinkerHQ, Google Code, Fedora Hosted, BerliOS, Gitorius, GNU Sa-

vannah, Gna!, CodePlex, CodeHaus, CloudForge, caen eliminadas para una posible solución.

En la siguiente iteración de los requisitos se busca la gestión de la documentación, wiki, carpetas públicas y revisión del código, por lo que: Alioth y GitlabHQ se descartan.

En el último escalafón tenemos a *GForge*. GForge es un modelo orientado a la gestión del proyecto basando en la forja Savannah. Se ha convertido en una compañía dedicada a la gestión de forjas de desarrollo a través en un modelo SaaS. El proyecto se discontinuó creando un nuevo fork llamado FusionForge en 2009 [8] . Por lo que en ese año no parecía una apuesta segura para crear una forja de desarrollo. En cuanto a cuestiones técnicas, FusionForge no soportaba repositorios distribuidos como es el caso de Git en sus primeras versiones.

Como hemos analizado en la comparativa de la sección Tablas comparativas, el punto correspondiente al sistema de construcción integrado, no se haya muy extendido en las forjas analizadas. Por lo que es necesario que nos permita adjuntar esta herramienta a la forja de desarrollo para completar el proceso.

5.5. Conclusiones del estudio de forjas

Después de haber analizado por encima las soluciones existentes, habiendo encontrado soluciones privadas o con pocos recursos, se reafirma la opción construir una forja de desarrollo que integre el proceso Iterativo e Incremental, desde la recogida de los requerimientos (ITS) hasta la construcción de la solución (sistema de construcción integrado). Crear un sistema modular que permita integrar estas herramientas del ciclo de vida de un proyecto como vehículo en la forja de desarrollo.

Capítulo 6

SCStack

SCStack es el conjunto de herramientas que componen la Forja de desarrollo. La palabra *stack* en Inglés significa *pila*, con respecto a la forja se trata del conjunto de herramientas apiladas y conectadas que dan forma a la Forja.



Figura 6.1: Ejemplo de pila a partir de cajones

6.1. Arquitectura

La arquitectura del Sistema de Gestión de la Forja se ve reflejada en el siguiente esquema:

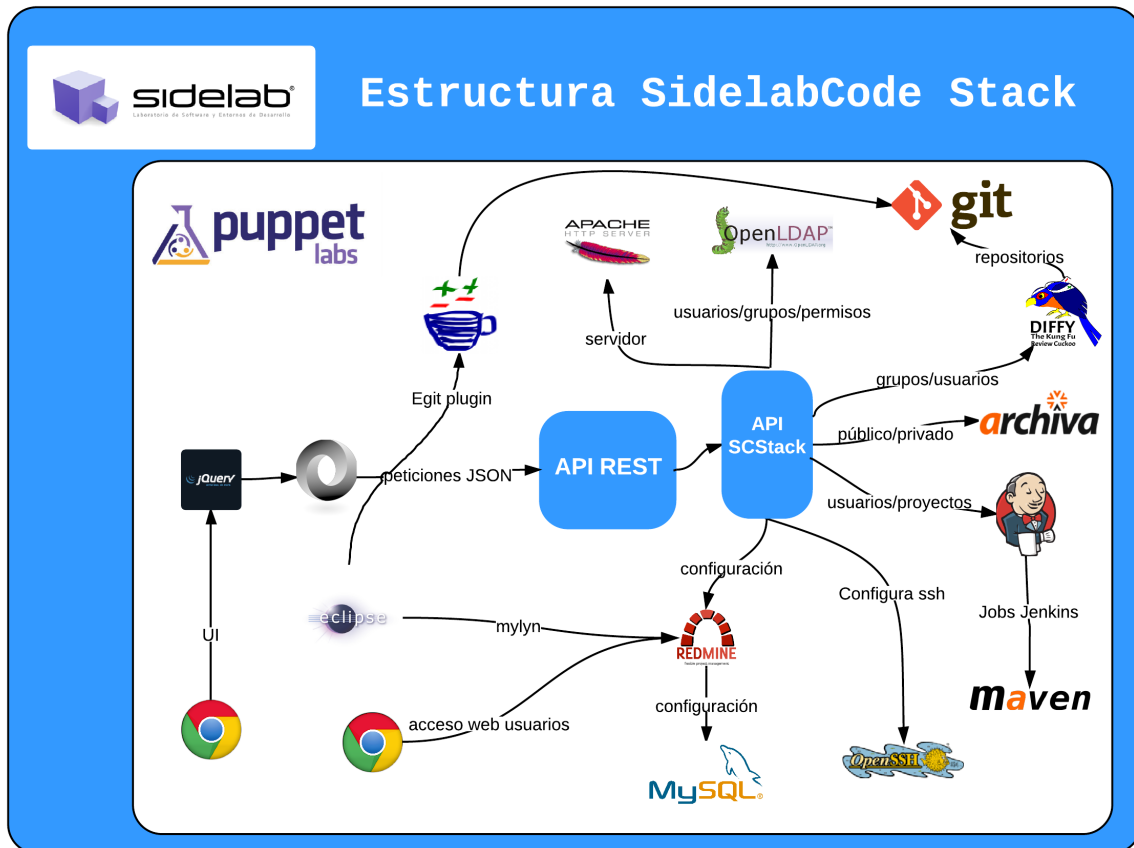


Figura 6.2: Diagrama de arquitectura SCStack

El proyecto consta de tres componentes conectados entre sí para la comunicación del usuario con las herramientas para la gestión de los proyectos: La Consola de Administración, el servidor de servicios web REST y el API.

6.1.1. Consola de Administración

Consola de Administración: Se trata de la capa de la vista encargada de interactuar con el usuario. A través de la interfaz que se ejecuta sobre un navegador web, el usuario admin-

istrador gestiona los usuarios y los proyectos de la Forja: crear, editar, modificar, eliminar y buscar), el típico sistema CRUD-F¹ (*CRUD: Create, Read, Update and Delete also Find*). Está diseñada con el framework Javascript *jQuery* por lo que no requiere ninguna librería externa para su uso, únicamente el navegador del cliente. Las operaciones que se realizan en la capa UI se trasladan al servidor REST a través de peticiones *http* asíncronas mediante las interfaces REST definidas.

6.1.2. Servicio web REST

REST: Representational State Transfer se trata de una arquitectura acuñada por *Roy Fielding*², uno de los autores de la especificación del protocolo *HTTP*. Esta arquitectura de comunicación se basa en cuatro principios:

- Un protocolo cliente/servidor *sin estado*: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión.
- Un *conjunto de operaciones* bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son **POST**, **GET**, **PUT** y **DELETE**.
- Una sintaxis *universal* para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.
- El uso de *hipermedios*: tanto para la información de la aplicación como para las transiciones de estado de la aplicación. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

¹http://en.wikipedia.org/wiki/Create,_read,_update_and_delete

²<http://roy.gbiv.com/>

El uso de los servicios REST para la comunicación mediante el protocolo http con la Consola de Administración agiliza la adaptación de la funcionalidades al cliente, el tráfico y las posibles adaptaciones de nuevos clientes para el acceso a través de cualquier plataforma.

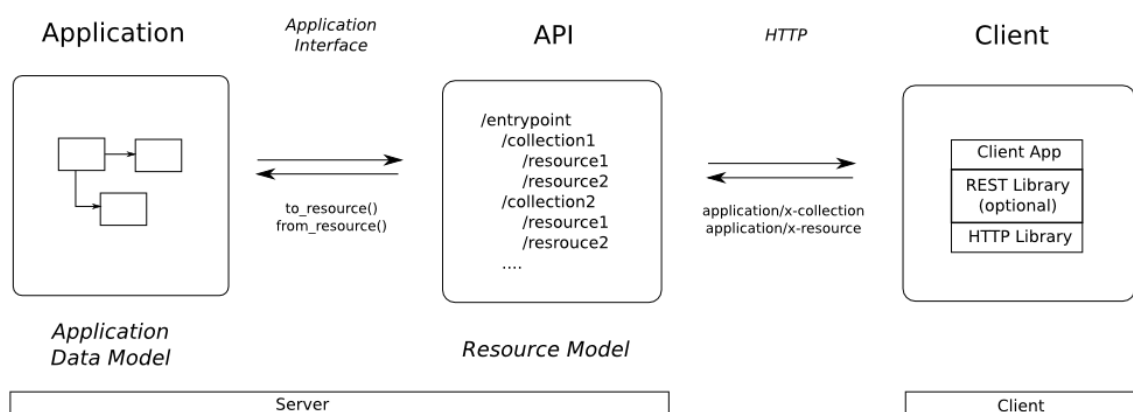


Figura 6.3: Diseño de un API RESTful

Servicio web REST: Es el componente encargado de gestionar las peticiones http a los servicios REST de la Consola de Administración y el API de SidelabCode. Define e implementa la lógica a seguir para la construcción de un proyecto a través de la UI mediante las llamadas ordenadas al API de cada una de las herramientas involucradas en el servicio como componentes de la forja.

Los servicios REST ofrecen tres interfaces distintas accesibles a través de http: XML, JSON y HTML para la comunicación la API. Además, hay que destacar que el servicio web es servido a través del framework *Restlet* a través de Internet de forma continua y remota a cualquier usuario o proceso cliente.

Se encarga de la seguridad y validación para las operaciones permitidas o denegadas del usuario de la UI, proporcionando, según su rol, determinados servicios a través de la Consola de Administración, buscar proyectos, crear usuarios, editar proyectos, crear proyectos, etc.

6.1.3. API

API: La API es el núcleo funcional de SCStack, coordina y realiza las tareas para cada componente de la forja con respecto a los usuarios y proyectos involucrados. Traslada las órdenes ejecutadas en la capa UI a las distintas herramientas para configurar su funcionamiento. Está conectada a todos los servicios que ofrece pivotando a través del directorio LDAP: Control de Versiones, Directorios, Integración Continua, ITS, Revisiones, Gestión de Dependencias, Seguridad y Autenticación que analizaremos extensamente más adelante componente a componente.

6.2. Aprovisionamiento (Puppet)

Aprovisionamiento: 'Accción o efecto de aprovisionar', *aprovisionar:* abastecer³. Aplicado a la Ingeniería del Software el Aprovisionamiento nos provee de los componentes necesarios para construir una solución.

El aprovisionamiento trata de la automatización de tareas para construir el entorno deseado, en este caso la instalación de SidelabCode Stack. La evolución en el modelo de instalación para facilitar la ejecución de módulos, configuración y comunicación entre los distintos componentes.

La herramienta empleada para el aprovisionamiento es **Puppet** de la compañía *Puppet-Labs*⁴.



Figura 6.4: Puppet Labs Logo

³Definición del diccionario de la RAE - http://buscon.rae.es/drae/?type=3&val=aprovisionar&val_aux=&origen=REDRAE

⁴<https://puppetlabs.com/>

Puppet: es una herramienta *Software Libre*⁵ de aprovisionamiento desarrollada en *Ruby*⁶. Gestiona la infraestructura a través de su ciclo de vida, desde el aprovisionamiento y la configuración de parches automatizando la ejecución de órdenes para la instalación y configuración del entorno.

- Automatizar tareas repetitivas.
- Desplegar rápidamente aplicaciones críticas.
- Gestionar proactivamente el cambio.
- Escalar de 10 servidores para 1000.
- Instalaciones locales o en la nube.

Puppet utiliza un enfoque declarativo, basado en el modelo de automatización:

- *Definir* el estado deseado de la configuración de la infraestructura mediante lenguaje de configuración declarativa de Puppet.
- *Simular* los cambios de configuración antes de la ejecución.
- *Corroborar* el estado final mediante despliegues automáticos comprobando las posibles desviaciones en la configuración.
- *Informe* sobre las diferencias entre los estados reales y deseados y cualquier cambio que haya hecho cumplir el estado deseado.

⁵<https://puppetlabs.com/puppet/puppet-open-source/>

⁶<https://github.com/puppetlabs/puppet>

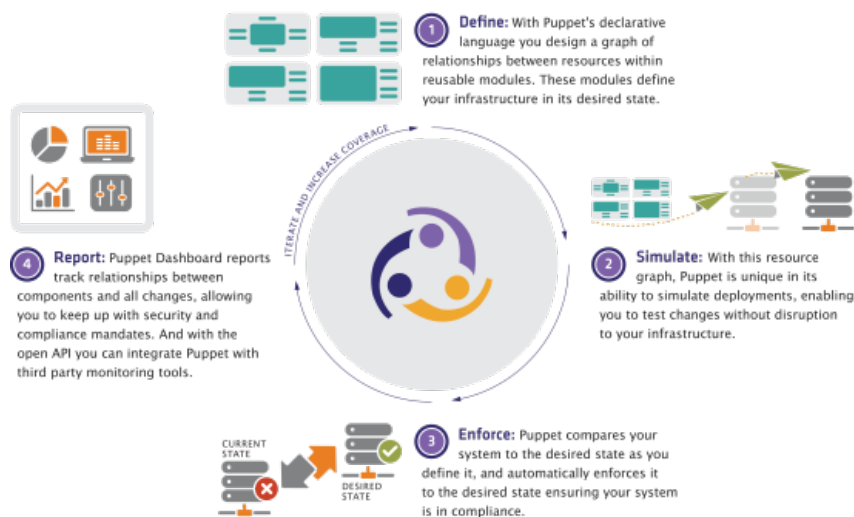


Figura 6.5: Ciclo de vida de los módulos Puppet

El diseño del aprovisionamiento a través de Puppet se basa en módulos. Cada uno de estos módulos se encarga de gestionar la instalación y configuración del componente definido.

Que aporta Puppet ? Proporciona un control sobre la instalación de cada herramienta y la configuración asociada por defecto que se define. De esta forma el proceso de instalación se automatiza permitiendo la replicación del mismo, completo o por módulos en diferentes entornos, locales o virtuales. La instalación a partir de módulos ha de definir una cadena de dependencias entre cada uno de ellos de manera que se vayan habilitando funcionalidades e interacciones (Apéndice A).

En SidelabCode Stack Puppet es el encargado de gestionar la *instalación y configuración* de cada uno de los componentes mediante módulos Puppet independientes para completar el proceso de instalación (Apéndice B) entre 5 y 10 minutos.

6.3. Componentes

En este apartado se van a analizar los distintos componentes que integran la Forja SidelabCode Stack y el papel que desempeñan en el funcionamiento del sistema partiendo del esquema en donde se refleja la arquitectura de la Forja SidelabCode 6.2 y la relación que

mantienen con las necesidades descritas en el capítulo Procesos de desarrollo a la hora de gestionar el proceso *Iterativo e Incremental*.

6.3.1. Usuarios, roles y grupos

Gestión de usuarios, roles, grupos y proyectos a través de *OpenLDAP*⁷.



Figura 6.6: OpenLDAP logo

SCStack utiliza la tecnología de directorios *LDAP* como sistema de autenticación y de información centralizado de la Forja Software. En dicho servidor de directorios se almacena información relativa a todos los usuarios, proyectos software y repositorios de la Forja. Esta tecnología, además, cuenta con la ventaja de que la mayoría de aplicaciones web con sistemas de autenticación ofrecen interfaces que garantizan una completa integración con directorios LDAP, este es el caso de Redmine, Drupal, Wordpress y el propio servidor web Apache. La implementación de este protocolo en la Forja Sidelab se lleva a cabo mediante un servidor de directorios muy estable y de libre distribución que es OpenLDAP.

6.3.2. API OpenLDAP

Debido a que el directorio LDAP es la estructura de información centralizada de la Forja, cualquier tipo de acción que se quiera realizar sobre el sistema requerirá el acceso por parte de la API a este servicio de directorios, bien para la recuperación de datos en las consultas o para la manipulación de registros a la hora de crear, editar o borrar usuarios o proyectos.

Todas las acciones de la Consola de Administración pasan a través de la API que comunica con OpenLDAP dando acceso y generando las distintas autenticaciones en cada una de las

⁷OpenLDAP - <http://www.openldap.org/>

herramientas interconectadas basándose en los registros de OpenLDAP como generador y autenticador de credenciales.

6.3.3. Gestión de Requisitos ITS

La gestión de requisitos en SidelabCode Stack se lleva a cabo a través del Issue Tracking System *Redmine*⁸.



Figura 6.7: Redmine logo

Este apartado es el más cuidado e importante en el conjunto de SCStack ya que se trata de la herramienta encargada de centralizar la gestión de *Lista de Control del Proyecto* por cada Iteración. El ITS Redmine es una aplicación web de gestión de proyectos Software multiplataforma desarrollada en *Ruby* a través de *Ruby on Rails*. Por supuesto se trata de una herramienta FLOSS.

Redmine proporciona la gestión de tareas (de cualquier tipo; *features, bugs, parches...*) para cada uno de los proyectos creados. Dentro de SCStack se encuentra enlazado a la configuración de usuarios a través de OpenLDAP para la validación y cuenta con una base de datos propia *MySQL*. Un apartado importante ya que esto permite gestionar las migraciones de Redmine de manera rápida, eficiente y fluida de una versión a otra o incluir la información de otro Redmine⁹ en una nueva instalación de la Forja, únicamente haciendo un backup de la base de datos y algunos directorios clave. Incluso la migración a Redmine

⁸Redmine - <http://www.redmine.org/>

⁹Upgrading Redmine - <http://www.redmine.org/projects/redmine/wiki/RedmineUpgrade>

desde otros gestores de tareas¹⁰.

Proporciona una interfaz adaptada para cada proyecto de la forja asociado a un repositorio de código fuente con un visor integrado. Se visualizan los cambios entre distintas versiones del código y comparaciones entre distintas ramas de desarrollo para seguir la evolución del proyecto.

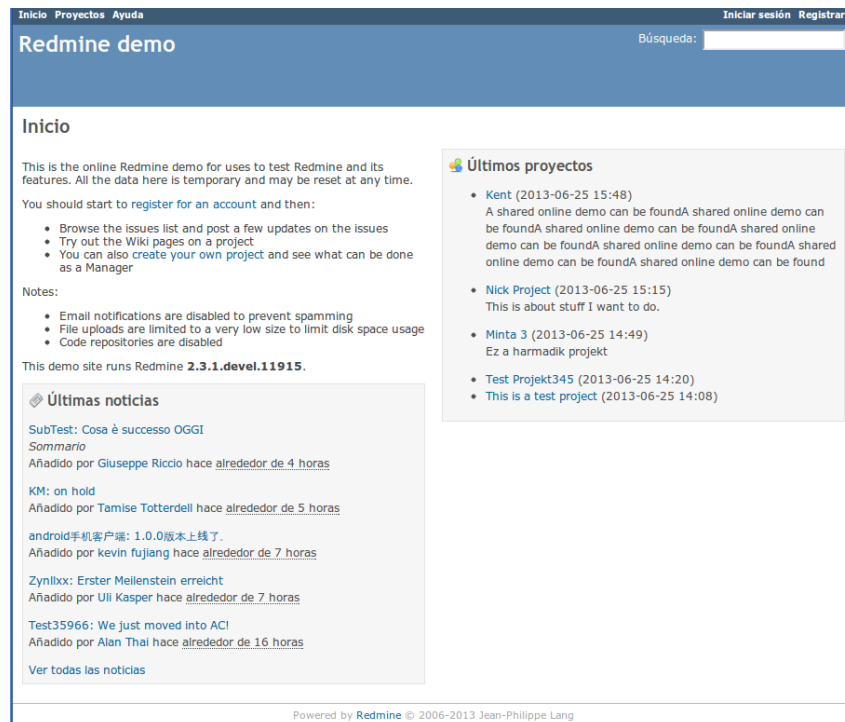


Figura 6.8: Redmine página de bienvenida

Dentro del proceso Iterativo e Incremental como herramientas nos proporciona informes de seguimiento de actividad dinámicos (*día, persona, proyecto...*), un *Wiki* para la documentación y una página de noticias.

Es la interfaz web de entrada al proyecto para los desarrolladores, por ello es la parte más importante y en donde se centran gran parte de los esfuerzos para la fluidez y la importancia que tiene la misma.

Por otra parte Redmine proporciona una API Rest para facilitar la interoperabilidad entre los distintos componentes. Esta API se maneja a través de la capa de negocio de la Consola de

¹⁰Migrate to Redmine - <http://www.redmine.org/projects/redmine/wiki/RedmineMigrate>

Administración de SCStack para así a través del API se registren en Redmine los usuarios, proyectos y grupos con sus respectivos permisos partiendo de los valores introducidos a través de la Consola de Administración. De esta forma, Redmine pasa a gestionar a través de su BBDD MySQL los usuarios y proyectos después que se hayan autenticado a través de OpenLDAP para así cargar la configuración obtenida de su BBDD.

Las operaciones de gestión administrativa en Redmine permanecen desactivadas, de modo que ningún administrador de proyectos puede añadir ni borrar miembros de sus proyectos, tampoco pueden crearse ni borrarse usuarios o proyectos, etc. *Obligando* así que todas las operaciones de gestión de la Forja se lleven a cabo desde el Software diseñado específicamente para ello, la Consola de Administración, garantizando así la consistencia.

6.3.4. Plugins

A la hora de aplicar el desarrollo Iterativo e Incremental se intenta incrementar la eficacia, la visibilidad, la rapidez y la comprensión del proceso de desarrollo. Por eso se han utilizado distintos plugins para Redmine que ayudan a la comprensión y agilizan el proceso Iterativo e Incremental para los usuarios (desarrolladores, gestores, etc. . .) a través del plugin FLOSS *Backlogs*¹¹.

Backlogs aporta una gestión visual en modo tablón de las Iteraciones activas en el proceso. Gestiona 'manualmente' a través de *Drag and Drop* la organización de las tareas del proyecto y agiliza la comprensión del estado de la iteración, proyecto, desarrollo en sí, en un sólo vistazo 6.9.

¹¹Redmine Backlogs - <http://www.redminebacklogs.net/>

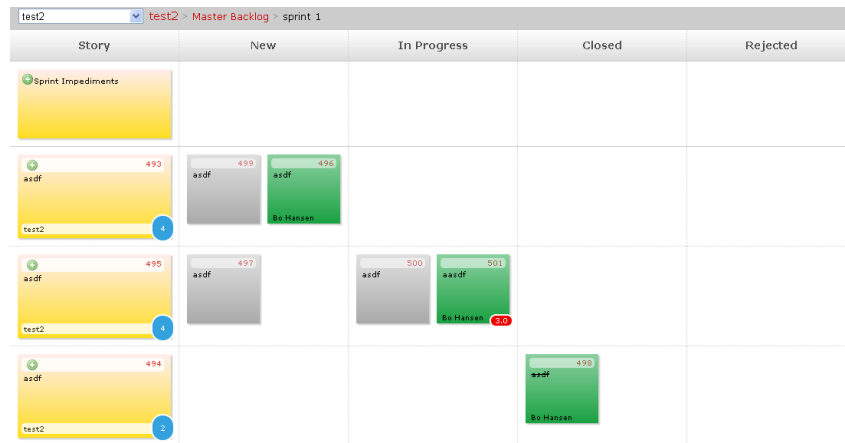


Figura 6.9: Redmine Backlogs plugin: tareas Redmine agrupadas por estados dentro de una historia de usuario.

Otro apartado importante en el desarrollo Iterativo e Incremental es la gestión de la documentación, en este caso Redmine nos proporciona una Wiki por cada proyecto. Una Wiki es una herramienta para la documentación colaborativa que mantiene su histórico de cambios (asociados a usuario y tiempo) y que mediante el lenguaje de marcado wiki desde el cual se exportan a distintos formatos como: *html*, *pdf*, *etc*, a un coste de recursos bajo ya que se trata de texto plano. El caso más famoso del uso de Wikis como documentación colaborativa es la *Wikipedia*¹².

6.3.5. Repositorios de Código

Los repositorios de código (VCS) son los encargados de gestionar el ciclo de vida del código fuente como hemos visto en el apartado de Código versionado 4.6 y existen dos tipos: centralizados y distribuidos. Para el proceso de desarrollo Iterativo e Incremental hemos seleccionado el repositorio distribuido *Git*¹³.

¹²Wikipedia - <http://www.wikipedia.org>

¹³Git - <http://git-scm.com/>



Figura 6.10: Git SCM Logo

Atendiendo a los requerimientos del proceso:

El desarrollo de la solución se adapta al uso del Repositorio distribuido, ya que éste aporta una flexibilidad para la gestión de bifurcaciones del código que en un repositorio centralizado no tenemos fácilmente

En el proceso Iterativo e Incremental abundan las ramificaciones del código en el repositorio debido a ello se opta por la elección del repositorio distribuido Git en pro del repositorio centralizado SVN, que también se incluye en la Forja SCStack. Las ramificaciones, la cantidad de fusiones entre ramas [2], entornos de desarrollo sumando la facilidad, agilidad y el bajo coste en Git nos proporcionan la herramienta adecuada para la gestión del código fuente en SCStack.

En el repositorio Git no se guardan diferencias se guardan snapshots en comparación con Subversion.

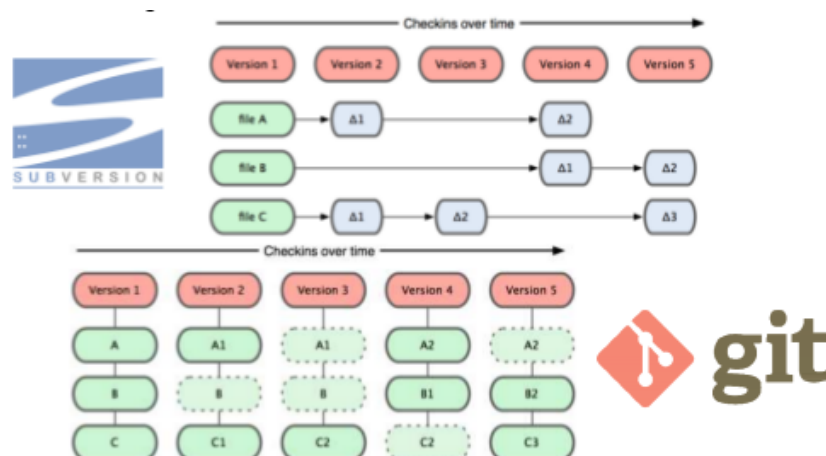


Figura 6.11: Comparación de incrementos entre SVN (completo) y Git (incrementos en archivos)

For example the Mozilla repository is reported to be almost 12 Gb when

stored in SVN using the fsfs backend. Previously, the fsfs backend also required over 240,000 files in one directory to record all 240,000 commits made over the 10 year project history. This was fixed in SVN 1.5, where every 1000 revisions are placed in a separate directory. The exact same history is stored in Git by only two files totaling just over 420 Mb. This means that SVN requires 30x the disk space to store the same history¹⁴

Asociada a cada iteración se ha de gestionar la viabilidad de una rama del repositorio. En esta rama se trabaja con respecto a las tareas a desarrollar para la iteración. Se hacen las pruebas necesarias para cada desarrollo para después integrar la solución en la rama principal y así crear una nueva versión del proyecto.

La ramificación del desarrollo permite gestionar por iteraciones incrementales aisladas en una nueva rama, de esta forma, el contenido de la rama principal es fiable, ya que para añadir contenido a la rama ha tenido que pasar una iteración nueva y el proceso de desarrollo que conlleva; planificación, test, implementación e integración. El resultado está altamente controlado y distribuido en base a módulos para poder acotar posibles errores posteriores o revertir cambios a través de un proceso minuciosamente controlado.

Integridad

Los commits se identifican por un hash sha1

- Svn: rev 33
- Git: d025a7b3217f05110ebbf48065b8d02a0ad22ae3

O más amigablemente: d025a7b

Los ficheros también se identifican por su sha1 de esta forma si un fichero se corrompe durante la transmisión por la red se detecta inmediatamente

¹⁴Git Svn Comparison Smaller Space Requirements - https://git.wiki.kernel.org/index.php/GitSvnComparison#Smaller_Space_Requirements

Los 3 estados

Los ficheros en Git pueden estar en tres estados:

- **Modificado:** el fichero ha cambiado desde el último checkout
- **Staged:** un fichero modificado ha sido marcado para ser añadido en el próximo commit
- **Committed:** el fichero se encuentra en la base de datos de git

Hay un 4º estado: **untracked**.

Las 3 áreas de un proyecto git

1. El directorio git (git directory):

- Contiene los metadatos y la base de datos de git
- Es lo que se copia cuando se clona un repositorio
- Normalmente es una carpeta .git en algún directorio

2. La carpeta de trabajo (working directory):

- Es un checkout de una versión específica del proyecto
- Se extrae del directorio git
- Es el espacio donde modificamos los ficheros

3. Staging area:

- Fichero en el directorio git que indica qué cambios van en el próximo commit

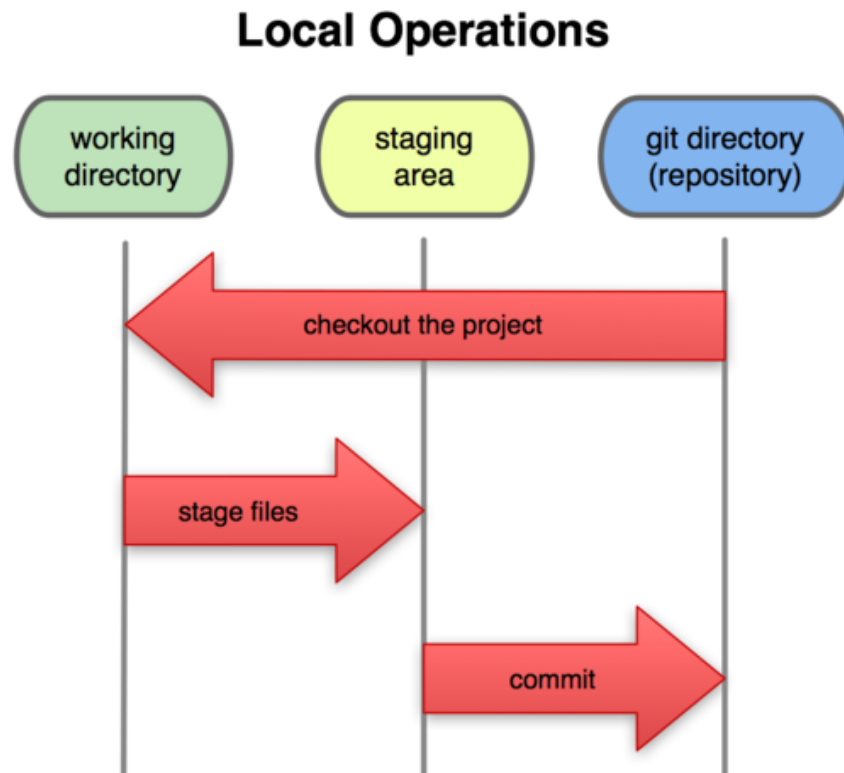


Figura 6.12: Tres áreas en Git

La identidad

Git necesita conocer algunos datos del desarrollador (aparecen en los commits para identificar al autor)

- Nombre
- Email

Si no están correctamente configurados pueden aparecer varios problemas:

- Los commits **fallan** porque el usuario no está autorizado
- Commits del mismo usuario '*físico*' **no son considerados como del mismo usuario** porque el nombre '*lógico*' cambia.

Por lo que se ha de tener muy en cuenta este apartado y configurar correctamente en el archivo.

6.3.6. Revisión de Código

La gestión del repositorio Git está orientada a través de la herramienta *Gerrit*¹⁵.



Figura 6.13: Gerrit Kungfu Review Cuckoo

Gerrit es una herramienta de revisión de código basada en web. Facilita el control online de las revisiones de código para los proyectos a través de la herramienta Git.

Gestiona la autenticación y la gestión de permisos, roles y grupos para cada uno de los repositorios existentes a través de una interfaz ligera. Un punto a destacar es la orientación hacia un repositorio centralizado de Git, es decir, el control de código se centraliza en un repositorio para la integración dependiente de los repositorios de los desarrolladores¹⁶.

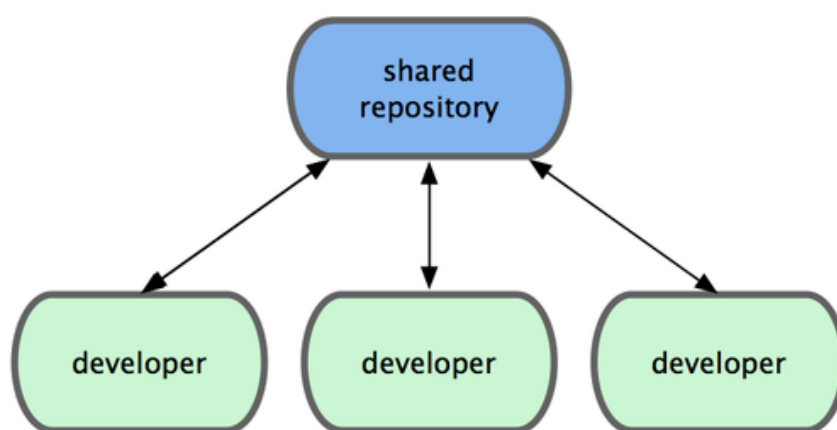


Figura 6.14: Git: Flujo de trabajo centralizado

¹⁵<http://code.google.com/p/gerrit/>

¹⁶Git Flujo de trabajo centralizado - <http://git-scm.com/book/es/Git-en-entornos-distribuidos-Flujos-de-trabajo-distribuidos#Flujo-de-trabajo-centralizado>

Centralizar la rama principal del proyecto en el repositorio descentralizado casa perfectamente con el proceso Iterativo e Incremental ya que a cada iteración cerrada añadimos los cambios a la rama estable del proyecto desarrollado centralizando el código a empaquetar para convertir en la solución final, incremento a incremento.

La configuración viene a través del API de SCStack y una consola interactiva SSH¹⁷ que proporciona Gerrit. En SCStack esta función recae sobre los valores obtenidos del servidor *OpenLDAP* a partir de los datos introducidos en la Consola de Administración a la hora de gestionar la creación de un proyecto y el grupo de usuarios asociados al proyecto siguiendo el flujo de datos centralizado a través de la API Rest central. La creación del proyecto, la gestión de usuarios y permisos sobre el repositorio Git creado.

Este procedimiento se encapsula a través del envío de órdenes a la consola a través del API habiendo configurado la autenticación a través de un conjunto de clave SSH en la instalación de SCStack.

6.3.7. Integración Continua

La integración continua dentro del proceso de desarrollo Iterativo e Incremental es la encargada de agilizar la comunicación entre los distintos actores involucrados. Evalúa la evolución de la solución y provee una respuesta firme y profesional para afianzar los resultados de cada incremento.

¹⁷Gerrit Command Line Tools - <http://gerrit.googlecode.com/svn/documentation/2.0.34/cmd-index.html>



Figura 6.15: Jenkins CI Logo

En este aspecto, la herramienta que SCStack necesita es un gestor de integración continua como *Jenkins-CI*¹⁸:

In a nutshell Jenkins CI is the leading open-source continuous integration server. Built with Java, it provides over 400 plugins to support building and testing virtually any project.

Jenkins-CI¹⁹ proporciona la integración de la integración continua (valga la redundancia) en el proceso de desarrollo de software de una forma modular e interoperable. No se trata de un servidor intrusivo ni para el proceso de desarrollo ni para el desarrollador en si.

En este aspecto SCStack incluye el módulo de Jenkins-CI en la instalación para facilitar el trabajo de las pruebas, integración y generación de versiones mediante la virtualización de escenarios más cercanos al sistema de producción. Des esta forma se perfila mejor el rendimiento, la replicación de errores y el control de la evolución del código fuente. Dentro del desarrollo, el código fuente es el centro de la calidad y debido a esta afirmación, se necesita un servidor de CI adaptado al proceso de desarrollo.

- Tags.
- Construir las versiones vivas.

¹⁸Jenkins-CI - <http://jenkins-ci.org>

¹⁹Meet Jenkins-CI - <https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins>

- Branches de releases.
- Desplegar una versión específica con un clic.

Jenkins-CI se instala integrado en el marco de trabajo SCStack interoperando entre el repositorio de código fuente Git, la herramienta de gestión Gerrit y el despliegue de los tests en servidores como Apache o Tomcat dependiendo del producto final (esto no tiene nada que ver con el desarrollo, sería un requisito del proyecto). Jenkins-CI mantiene diferentes versiones vivas a la vez en el repositorio del proyecto para cumplir con los objetivos:

- Asegurar la calidad.
- Hacer el despliegue ágil.
- Minimizar el riesgo.

Jenkins-CI trabaja en base a *Jobs*. Un Job en Jenkins-CI describe el proceso de integración, pruebas o despliegue que va a ejecutarse cuando se cumpla un requisito definido o manualmente. La integración de las distintas ramas de desarrollo en cada iteración en el ciclo de vida 6.15 las gestiona Jenkins-CI a través de los Jobs definidos:

- Jobs de *integración*.
 - Descargan el código del repositorio Git.
 - Construyen.
 - Pasan tests.
 - Despliegan la versión construida en "local".
- Jobs de *release*.
 - Realizan los pasos anteriores y además.
 - Tag si los tests pasaron.
 - Push del tag al repositorio remoto.
- Jobs de *despliegue*.
 - Descargan el binario del repositorio de binarios

- Desplegar en un servidor de aplicaciones.

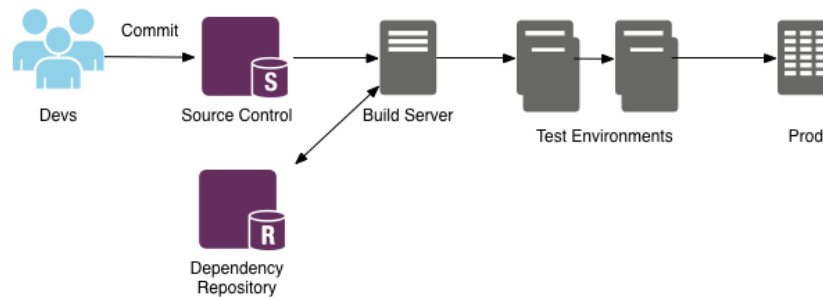


Figura 6.16: Ciclo de vida Jenkins CI

Dentro de la forja SCStack se configura el uso de Jenkins-CI a partir de usuarios dedicados a la gestión de jobs. Se plantean tres opciones:

- **Opción I:** Un usuario *jenkinsci* con acceso de *lectura/escritura* a *todos* los repositorios.
 - *Pros*; Simplicidad: sólo hay que gestionar un usuario. Una *única* clave ssh */home-/tomcat/.ssh/id_rsa.pub*.
 - *Contras*; Un usuario para dominarlos a todos. Cualquier error en un job para el proyecto *X* puede afectar a los repositorios del proyecto *Y*.
- **Opción II:** Un usuario *@jenkinsci@* con acceso de *lectura/escritura* a todos los repositorios y otro *jenkinsci_read* con acceso sólo de lectura
 - *Pros*; Sólo hay que gestionar *dos usuarios*: basta con asociarlos a *todos* los proyectos.
 - *Contras*; Seguimos teniendo un usuario para dominarlos a todo: *jenkinsci*. Cualquier error en un job para el proyecto *X* puede afectar a los repositorios del proyecto *Y*. Hay que gestionar dos claves ssh. No es trivial.
- **Opción III:** Un *usuario por proyecto* para integración continua.
 - *Pros*; El usuario de ci de un proyecto no tiene acceso a los repositorios de otro proyecto.

- *Contras*; Hay que gestionar múltiples *claves ssh*.

Se decide utilizar un usuario por proyecto definida en la **tercera opción**, después de haber evaluado las distintas opciones.

La gestión de los usuarios se gestiona a través de la consola de administración de SC-Stack. De esta manera la herramienta nos aporta una vía única de entrada dotando de la funcionalidad específica para distintos estados del proceso de desarrollo nada intrusivos en el desarrollo del proyecto. Separando a través del API de SCStack la gestión y partiendo de los roles/usuarios/grupos definidos como base en OpenLDAP.

El repositorio remoto debería contener versiones más o menos estables y distinguidas para el desarrollo por ramas establecido el proceso:

- **Develop**: Pueden fallar algunos tests, no es problema.
- **Release-X**: Los tests deberían pasar, eventualmente se podrían desactivar.
- **Nightly builds**: Construcciones que comprueban la '*salud*' del proyecto. Se hacen sobre los branches *development* y *release-X* (siendo X la versión más reciente).
 - Se ejecutan los tests.
 - Se despliegan dos versiones por cada rama: Limpia: una bbdd nueva y Migración: con actualización de bbdd ya existente sobre la bbdd que ya hubiera para ese despliegue.
- **Preproducción**: Las versiones que van a preproducción son aquellas de las que se ha hecho tag:
 - Pasan los tests.
 - El entorno de pre puede estar en *local* o en *Remoto*.
 - Se despliegan dos versiones: Limpia y Migración.
- **Producción**: Las versiones que van a producción son aquellas de las que se ha hecho tag.
 - Pasan los tests.

- El despliegue se realiza en *Remoto*.
- Se despliegan dos versiones: Limpia y Migración.

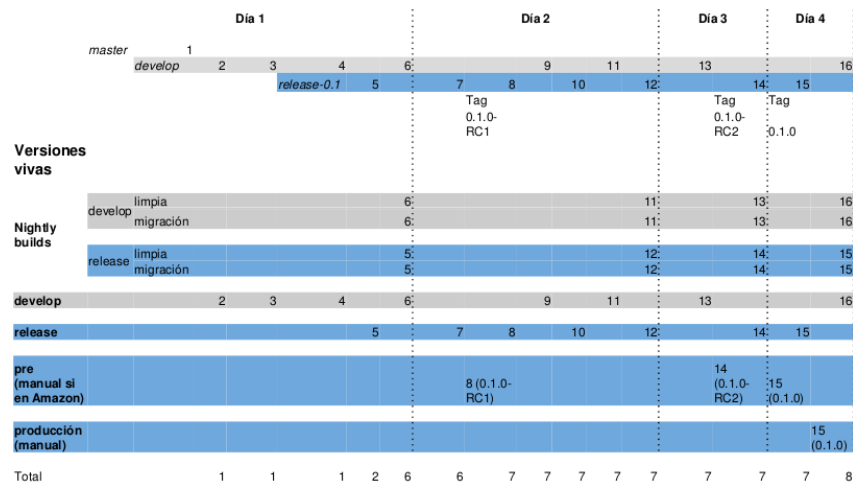


Figura 6.17: Integración Continua de versiones a partir de distintas ramas

6.3.8. Gestión de distribuciones y dependencias

Gestión de recursos compartidos, comunicación segura a través de *OpenSSH* y gestión de dependencias y distribuciones a través de *Archiva* como repositorio *Maven*.

OpenSSH

*OpenSSH*²⁰ es un conjunto de aplicaciones de libre distribución que permiten realizar comunicaciones cifradas a través de la red usando el protocolo SSH.



Figura 6.18: OpenSSH logo

La función fundamental que brinda este protocolo a la Forja Software consiste en garantizar

²⁰<http://www.openssh.org/>

que los usuarios de los distintos proyectos puedan acceder, por ejemplo a través de SFTP²¹ o Secure Shell SSH²², a las carpetas privadas y públicas de los proyectos en los cuales participan, a través de una conexión cifrada, y gestionar sus ficheros de una forma sencilla, ágil y segura.

Archiva

*Archiva*²³ es una aplicación FLOSS para la gestión de repositorios de artefactos de construcción, desarrollado por *Apache Software Foundation*. Este el compañero perfecto para herramientas de construcción como Maven.



Figura 6.19: Apache Archiva logo

Entre las funcionalidades que podemos destacar de esta herramienta están, el proxy para repositorio remoto (reduce el tráfico de red en equipos de trabajo), la gestión del control de acceso a los repositorios definidos, la construcción de artefactos de almacenamiento y, la gestión y mantenimiento de repositorios Maven facilitando la indexación, búsquedas, informes, estadísticas.

Posee una interfaz web sencilla de configurar en la que publica el contenido del repositorio Maven accesible a los usuarios autenticados. Además de el acceso al repositorio Maven habiendo configurado en los proyectos de desarrollo la url en la que está publicado:

```
<project>
  <!-- omitted xml -->
  <distributionManagement>
    <repository>
      <id>archiva.internal</id>
      <name>Internal Release Repository</name>
      <url>http://reposerver.mycompany.com:8080/archiva/repository/internal/</url>
```

²¹<http://en.wikipedia.org/wiki/SFTP>

²²http://en.wikipedia.org/wiki/Secure_Shell

²³<http://archiva.apache.org/index.cgi>


```
</repository>
<snapshotRepository>
  <id>archiva.snapshots</id>
  <name>Internal Snapshot Repository</name>
  <url>http://reposerver.mycompany.com:8080/archiva/repository/snapshots</url>
</snapshotRepository>
</distributionManagement>
<!-- omitted xml -->
</project>
```

El uso de Archiva como gestor de dependencias Maven puede ser público y/o privado dependiendo del repositorio que se cree. De esta forma los equipos de desarrolladores automatizan y centralizan las descargas de las dependencias (librerías) a través de Archiva dentro de la red local, disminuyendo el tráfico de red y agilizando la puesta en marcha de proyectos y los jobs de CI, reduciendo el consumo.

6.3.9. Desarrollador: Eclipse y Maven

El entorno de desarrollo para el desarrollador en el que se integra el proceso iterativo e Incremental consta de dos herramientas y un concepto.

El IDE *Integrated Development Environment* a utilizar es Eclipse, concretamente la distribución del framework Spring *Eclipse STS*²⁴.



Figura 6.20: Spring Tool Suite logo

STS otorga y proporciona un conjunto de herramientas preparadas para trabajar con los componentes de STStack: Redmine, Git, Gerrit, Jenkins, Apache, OpenLDAP, Maven. Se basa en una distribución de Eclipse orientada al desarrollo para trabajar con diferentes componentes externos facilitando la puesta a punto para empezar a desarrollar.

Esta distribución incluye la gestión de un proyecto a través *Maven*²⁵. Maven es un Software

²⁴Eclipse STS - <http://www.springsource.org/sts>

²⁵Apache Maven - <http://maven.apache.org/>

de gestión de proyectos basado en la definición del proyecto POM (Project Object Model). Es el encargado de compilar, ejecutar test, desplegar, gestionar las dependencias (locales y remotas) y generar distribuciones a partir de una configuración definida.



Figura 6.21: Maven logo

Esta herramienta se integra en el ciclo de vida del Software para automatizar los procesos tediosos manuales, proveer una estructura básica inicial a los proyectos (iteración 0) para ir incrementando iteración tras iteración a través de los arquetipos.

Incorpora componentes en la instalación por defecto para facilitar el trabajo del desarrollador con la forja SCSTack:

- *Egit* - Plugin de Eclipse para trabajar con repositorios Git - <http://www.eclipse.org/egit/>.
- *Mylyn* - Plugin para gestionar las tareas del ITS, en este caso Redmine, a través de la interfaz de Eclipse. Automatizando las pruebas, grabando sesiones de trabajo e incluso compartirlas para poder reproducir un procedimiento en otro sistema - <http://www.eclipse.org/mylyn/>
- *Maven* - Plugin *2eclipse* para la gestión de proyectos Maven a través de su ciclo de vida. Ayuda a facilitar la configuración de los builds a través de los pom de Maven. Genera esqueletos de proyectos a través de los arquetipos de una forma intuitiva - <http://www.sonatype.org/m2eclipse>

A través de Maven y Eclipse se definen las líneas del desarrollo de cada iteración facilitando el diseño TDD 4.7.1 a través de una sencilla configuración. El esqueleto de los proyectos Maven proporciona unas sencillas guías estándar para el desarrollo de proyectos de Software. Esta agrupación de funcionalidades se reutiliza en Jenkins-CI a través de los *jobs* de Maven para así poder replicar cualquier entorno durante la integración continua partiendo de una base robusta.

6.4. Pruebas y Validación

Todo proyecto de software necesita ser probado para comprobar que funcione correctamente si existen fallos para así poder atajarlos sin que las sorpresas aparezcan.

El lenguaje utilizado para el desarrollo del API del proyecto ha sido *Java*.



Figura 6.22: SidelabCode Stack Librería API

Para las pruebas del proyecto de la generación del API se ha utilizado el framework *JUnit*²⁶. La cobertura de tests del proyecto en JUnit no es muy halagüeña ya que es del 3,5% 6.21 pero en su defensa se puede justificar debido a que se trata de una librería que accede a APIs de terceros. Las APIs o librerías de terceros no han de desarrollar test unitarios sino test de aprendizaje por lo que no son tan útiles [9].

Por parte de la publicación del API como servicio Web REST se ha utilizado el framework *Restlet* pero en este caso vemos que la cobertura de test es del 0% 6.22. A partir de un proyecto nuevo y dependiente que utiliza la librería API para publicarla como Servicio Web REST ofreciendo las funcionalidades a la Consola de Administración de la Forja.

He utilizado la herramienta *SonarQube*²⁷ para obtener una visión más empírica del estado de la cobertura del proyecto. Utilizando las reglas básicas para poder fijarnos en la cantidad de pruebas definidas para *SidelabCode Stack API Lib* 6.21 y *REST Services* 6.22 ya que para obtener este dato no hace falta un gran refinamiento en la configuración.

²⁶JUnit - <http://junit.org/>

²⁷SonarQube - <http://www.sonarqube.org/>

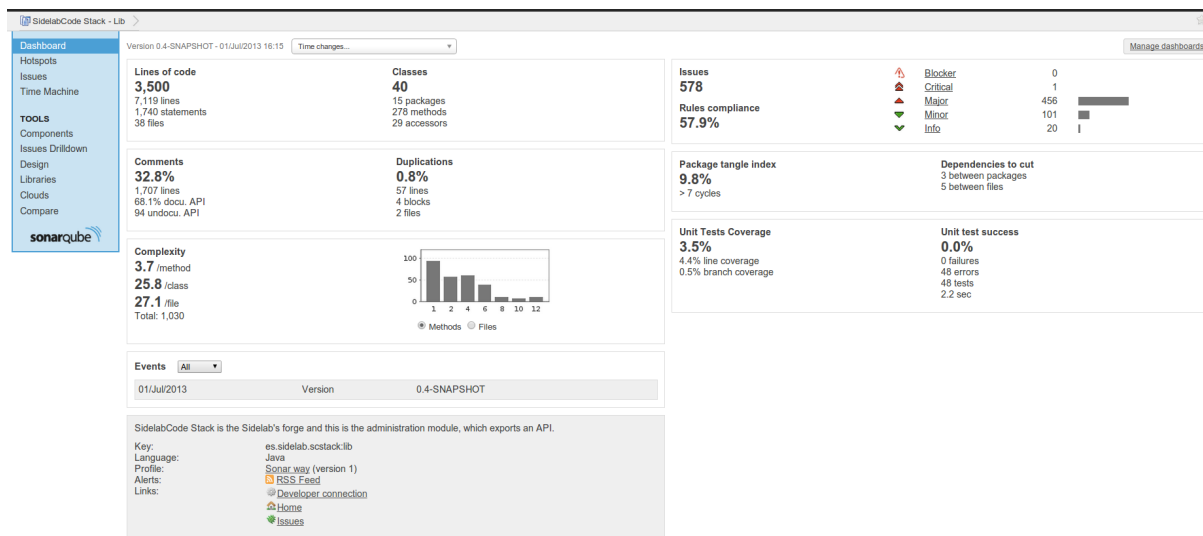


Figura 6.23: SidelabCode Stack API lib Análisis de cobertura con Sonar

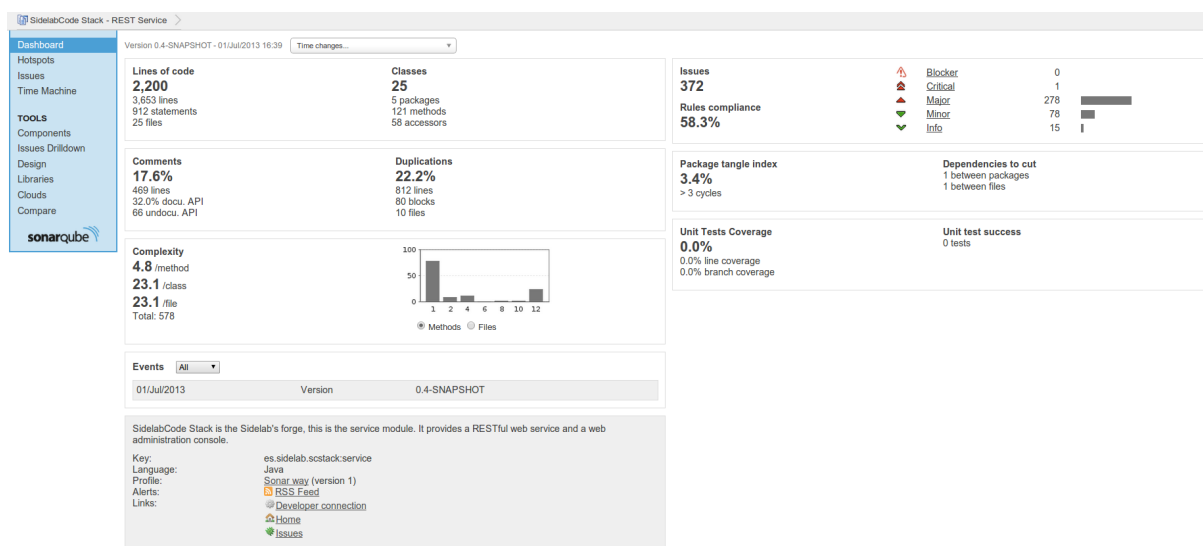


Figura 6.24: SidelabCode Stack REST Service Análisis de cobertura con Sonar

6.4.1. Pruebas instalación

Las pruebas más importantes en el proceso de desarrollo de la forja SidelabCodeStack han sido las de instalación y configuración de la forja que es donde se encuentra el valor del proyecto.

Previamente a la versión de la forja 0.2 la instalación se ejecutaba a partir de un instalador

Java y en consecuencia las pruebas y la replicación de los errores en diferentes entornos acarrearán un trabajo que no podía asumirse con sencillez. A partir de la citada versión (0.2) se introdujo Puppet como herramienta de aprovisionamiento y configuración de los distintos módulos que componen la forja. Este framework ofrece un nuevo camino dentro de las pruebas en el proceso de desarrollo, *la virtualización de las instalaciones*. Puppet nos proporciona un seguimiento completo de la instalación paso a paso a partir de la configuración definida en el proceso. Facilita la ejecución de la instalación de forja en entornos virtuales y seguros para poder asegurar el correcto funcionamiento y descubrir los posibles errores.

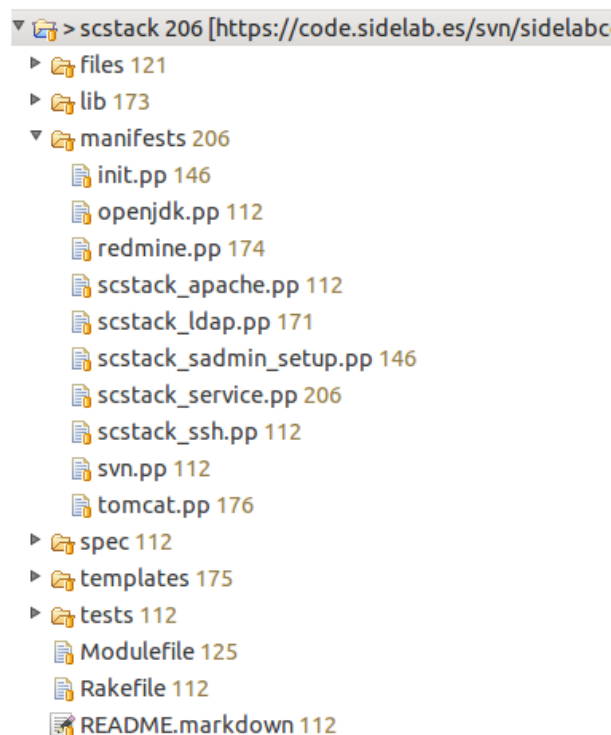


Figura 6.25: Estructura de módulo Puppet de SCStack

Pero no, no es suficiente ya que de esta manera el desarrollador ha de estar preocupado de invertir el tiempo en la gestión de las distintas máquinas virtuales, asignando espacio, potencia, configuración inicial, sobrecarga de la máquina de desarrollo y cumplir con una serie de tareas repetitivas que le alejan del camino marcado, el proyecto SidelabCode Stack.

En este punto aparece *Vagrant*²⁸.



Figura 6.26: Vagrant Logo

Create a single file for your project to describe the type of machine you want, the software that needs to be installed, and the way you want to access the machine. Store this file with your project code.

Vagrant proporciona la gestión de los entornos de desarrollo sobre máquinas virtuales, ligeras, sencillas y replicables. Existe un repositorio de imágenes de virtuales²⁹ preparadas para Vagrant que se han de instalar la máquina del desarrollador con un simple comando:

```
config.vm.box = "precise64"
config.vm.network :hostonly, "192.168.33.10" # Si se desea se puede utilizar el modo bridge y
      asignar una ip por dhcp a la maquina dentro de la red en la que se encuentra el host
config.vm.provision :puppet, :module_path => "modules"
```

Se puede afirmar que Vagrant es la navaja suiza para las pruebas en este proyecto. A través de Vagrant se define el sistema operativo a instalar, la capacidad de la máquina y lo más importante: el **provisionador**, en nuestro caso Puppet y el módulo SidelabCode Stack.

A través de dos comandos se inicia el proceso completo de la instalación de SCStack en una nueva máquina virtual a partir de la configuración de una plantilla con las propiedades necesarias (Apéndice B).

```
$ vagrant box add base http://files.vagrantup.com/precise64.box
$ vagrant init
$ vagrant up
```

²⁸Vagrant - <http://www.vagrantup.com/>

²⁹VagrantBox.es - <http://www.vagrantbox.es/>

Permite la ejecución de pruebas para la instalación de SCStack de una manera eficiente y controlada aunque también es pesada debido a las características del proyecto.

De esta forma se completa el proceso de desarrollo con respecto a las herramientas y módulos utilizados para el desarrollo de la forja SidelabCode Stack ofreciendo las soluciones y herramientas necesarias para poder facilitar el uso del proceso desarrollo Iterativo e Incremental dentro de un marco seguro y fiable.

Capítulo 7

Comunidades FLOSS

El punto diferenciador en el proyecto haciendo hincapié en la interacción con las comunidades de software de cada una de las herramientas.

Capítulo 8

Desarrollo de un proyecto

Cómo llevar a cabo el desarrollo de un proyecto con SCStack. Desarrollo en paralelo de un proyecto mediante github + travis-ci vs gerit + jenkins.

8.1. Alta usuarios

Dar de alta desarrolladores/Project Owners (usuarios de la forja).

8.2. Crear un proyecto

8.2.1. Proyecto en redmine

8.2.2. Repositorio git

Configurar usuario Git del desarrollador del proyecto para la forja:

```
$ cat .gitconfig
[user]
    name = patxigortazar
    email = patxi.gortazar@gmail.com
$ git config --global user.name "ricardogarfe"
$ git config --global user.email "ricardogarfe@gmail.com"
```

```
$ cd [path-to-gitrepo]
[path-to-gitrepo]$ git config user.name "ricardogarfe"
[path-to-gitrepo]$ git config user.email "ricardogarfe@gmail.com"
```

Comprobar las credenciales de Git en el ordenador del desarrollador:

```
user.name=patxigortazar
user.email=patxi.gortazar@gmail.com
```

8.2.3. Configuración de Jenkins

Configuración de Jenkins para realizar determinadas tareas de forma automática:

- Tags
- Construir las versiones vivas

También proveerá tareas para ser ejecutadas manualmente:

- Branches de releases
- Desplegar una versión específica con un clic

Las versiones vivas vivirán en su propia máquina virtual

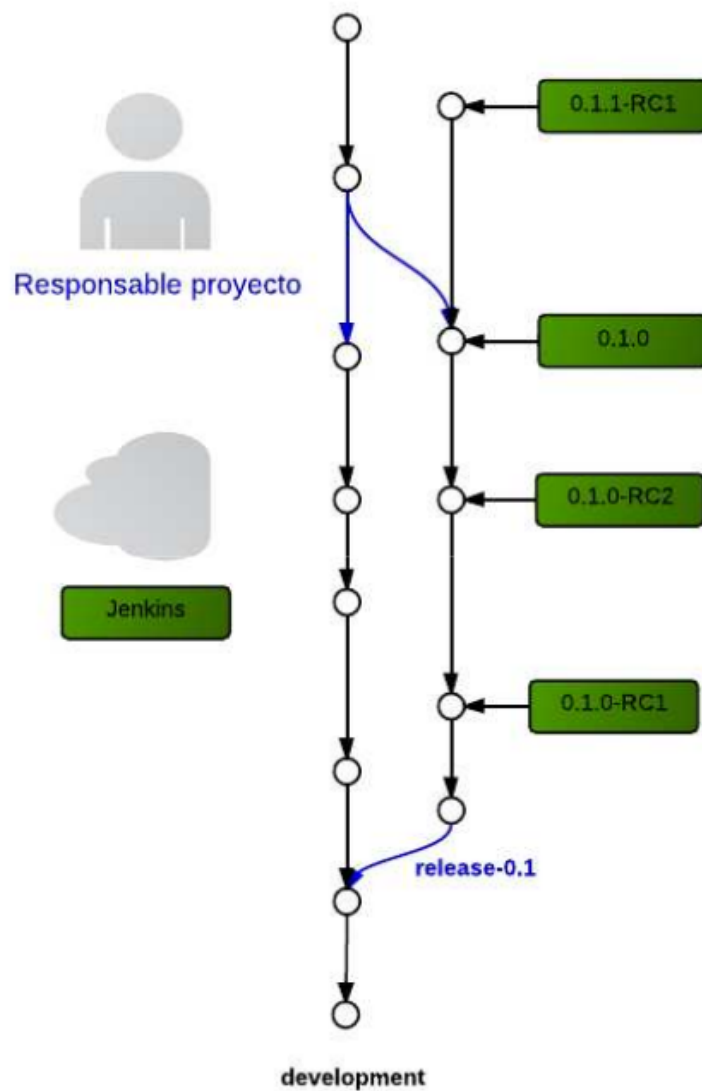


Figura 8.1: Jenkins-Git relación en el proceso de integración

Se mantendrán diferentes versiones vivas a la vez para cumplir unos objetivos con forma al desarrollo iterativo e incremental: *'Release early, release often'*:

- Asegurar la calidad
- Hacer el despliegue ágil
- Minimizar el riesgo

Para orientar la integración continua al proceso de desarrollo se ha de configurar *Jenkins* para acceder a múltiples repositorios Git para esto se han de tener en cuenta los siguientes

requerimientos:

- *Jenkins* construye diferentes proyectos en donde en proyecto puede tener su propio repositorio git.
- *Jenkins* debe tener permisos de lectura o lectura/escritura a todos los repositorios de aquellos proyectos que vaya a construir.
- Por defecto *Jenkins* usa la clave del usuario en \sim *.ssh* para autenticarse
- ¿Con qué usuario se ejecuta Jenkins? con el usuario **tomcat**.

El acceso de Jenkins a los distintos repositorios se configura a partir de un usuario por Jenkins repositorio, aislando los problemas descritos en la sección 6.3.7 del capítulo Procesos de desarrollo.

Configurar *Jenkins* para acceso a *múltiples repositorios git* creando el usuario necesario en la Consola de Administración.

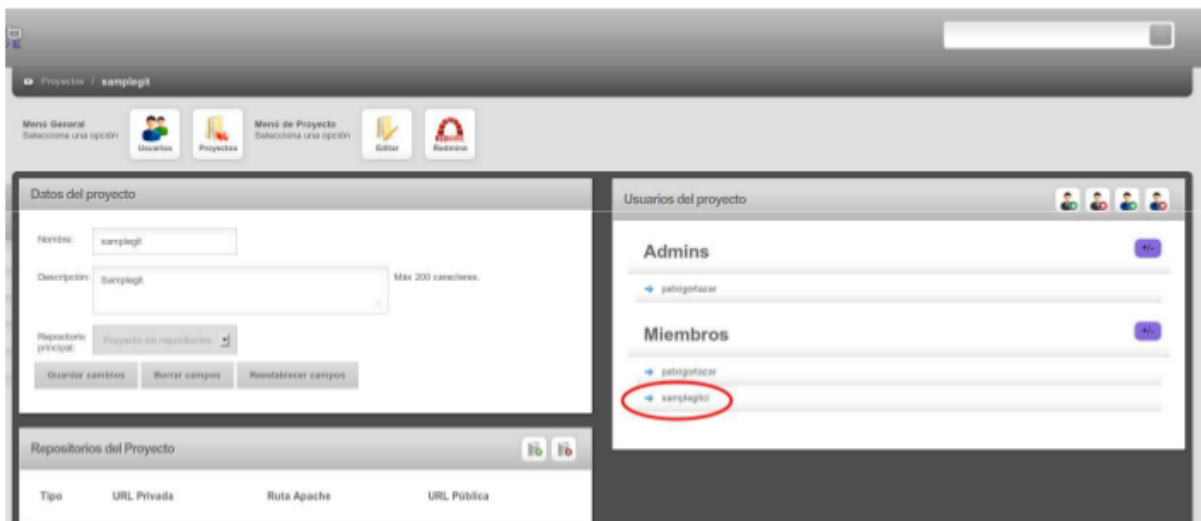


Figura 8.2: Crear usuarios para Jenkins a través de la consola de Administración

Generar un par de claves pública/privada con `ssh-keygen` para cada usuario en diferentes ficheros y acceder a Gerrit con cada usuario creado.

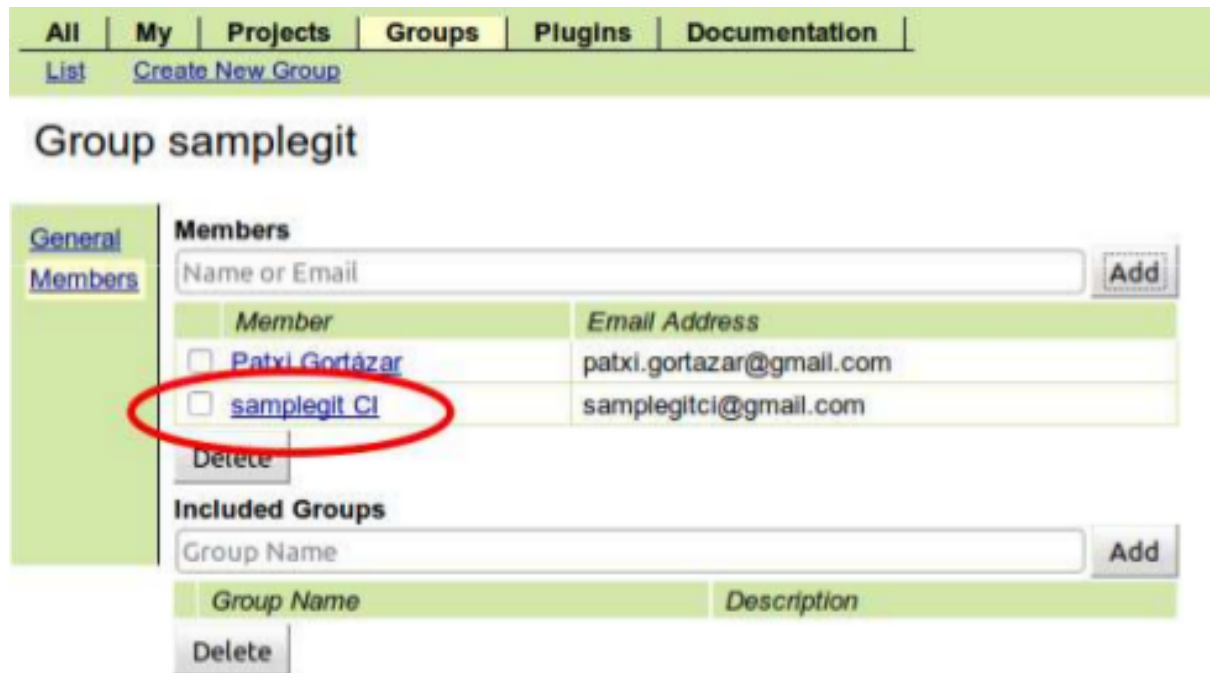


Figura 8.3: Configuración de usuarios Jenkins en Gerrit

Añadir la clave pública para este usuario para copiar las claves al servidor de Jenkins en el directorio '/opt/ssh-keys'.

```
$ git config --list
$ cd /opt/ssh-keys
$ ll
total 24
drwxr-xr-x 2 tomcat tomcat 4096 Jan 4 09:46 ./
drwxr-xr-x 14 root root 4096 Jan 4 09:42 ../
-rw-r--r-- 1 tomcat tomcat 1679 Jan 4 09:46 filetransferci_rsa
-rw-r--r-- 1 tomcat tomcat 398 Jan 4 09:46 filetransferci_rsa.pub
-rw-r--r-- 1 tomcat tomcat 1679 Jan 4 09:44 samplegitci_rsa
-rw-r--r-- 1 tomcat tomcat 396 Jan 4 09:44 samplegitci_rsa.pub
</code>
```

Configurar SSH para que utilice la clave correcta en cada caso creando el fichero /home/tomcat/.ssh/config

```
$ git config --list
$ cd /home/tomcat/.ssh
$ cat config
Host samplegit.ricardogarfe.sidelab.es
    HostName ricardogarfe.sidelab.es
    User samplegitci
    IdentityFile /opt/ssh-keys/samplegitci_rsa
Host filetransfer.ricardogarfe.sidelab.es
```

```
HostName ricardogarfe.sidelab.es
User filetransferci
IdentityFile /opt/ssh-keys/filetransferci_rsa
```

8.2.4. Configuración de builds

Los builds de *Jenkins* funcionan a través de la configuración de **jobs**. Por lo que vamos a definir los jobs necesarios para el proceso de integración continua. Se dividen en tres grupos:

- Jobs de **integración** (read only).
 - Descargan el código (checkout).
 - Construyen.
 - Pasan tests.
 - Despliegan la versión construida en “local”.
- Jobs de **release** (read/write).
 - Realizan los pasos anteriores y además.
 - Tag si los tests pasaron.
 - Push del tag al repositorio remoto.
- Jobs de **despliegue** (read only).
 - Descargan el binario del repositorio de binarios
 - Desplegar

Job de integración

Configurar el job de integración mediante Maven:

- Crear un **job Maven**.

- Configurar el repositorio git:
 - `ssh://filetransferci@filetransferci.code.tscompany.es/filetransfer`
 - Ssh leerá el fichero config y utilizará el fichero de claves correspondiente el host `filetransferci.code.tscompany.es`.
- Añadir las ramas a construir (añadir nuevas ramas con el botón “Add”)
 - development
 - release-0.1.1
- Añadir el `user.email` y `user.name` que usará *Jenkins*.

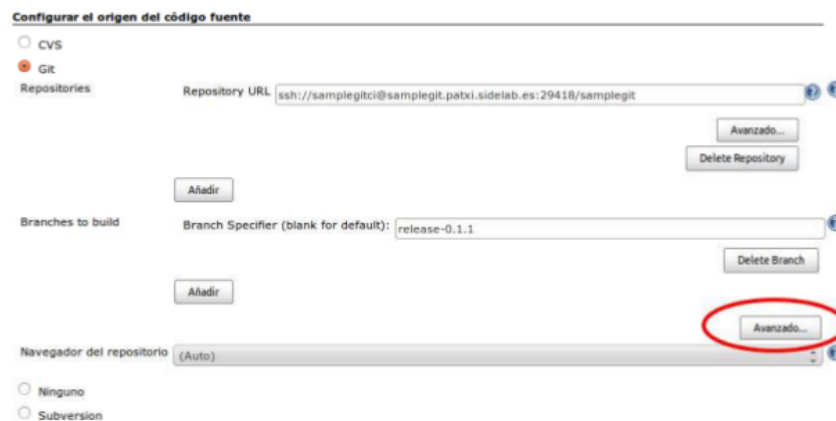



Figura 8.4: Crear Job integración en Jenkins

- Los resultados del build se pueden comprobar en:
 - `/opt/jenkins/jobs/filetransfer/workspace`
 - Si es un proyecto *Maven*, dentro del proyecto en la carpeta target estará el artefacto generado.
 - También se puede acceder vía web y descargar el workspace como un zip.
 - Los **tests** están en la carpeta `surfire-reports` del proyecto *Maven*.
 - También pueden consultarse vía web accediendo al build y seleccionando '*Resultado de los tests*'.



The screenshot shows the Jenkins web interface. The left sidebar contains a list of links: 'Volver al Proyecto', 'Estado', 'Cambios', 'Salida de consola', 'Editar información de compilación', 'Borrar esta ejecución', 'Git Build Data', 'No Tags', 'Resultado de los tests' (circled in red), 'Ver firmas', and 'Ejecución previa'. The main content area is titled 'Resultado del test' and shows '0 Fallidos' and '1 Test'. Below this is a table with the following data:

Módulo	Fallos	(diferencias)	Total	(diferencias)
es.sidelab.scstack.sync:sftp	0		1	+1

At the bottom of the page, there is a footer with the text 'Página generada el: 04-ene-2013 12:01:29', 'BEST API', and 'Jenkins ver. 1.491'.

Figura 8.5: Resultados de los test a través de la interfaz de Jenkins

8.2.5. Maven

Jenkins permite construir proyectos *Maven*.

En determinadas ocasiones los proyectos requieren configuraciones específicas. La información sensible suele ir en el fichero `settings.xml` en el `home` del usuario en su máquina de desarrollo.

- Info de **autenticación para Archiva**.
- **Profiles**

En Jenkins esto se puede gestionar con el plugin “*Config File Provider Plugin*”.



Figura 8.6: Configuración de Maven a través de Jenkins

Podemos añadir cualquiera de los ficheros creados con *Config File Management* en un **job**.



Figura 8.7: Seleccionar archivo de configuración de Maven

Para los deploys, si el certificado es autofirmado es **necesario** generar un **truststore** a

partir del certificado generado por el servidor¹.

Este truststore debe incluirse en todos los `jdk` que utilice *Jenkins* en la ruta indicada en el enlace anterior.

8.3. Proceso de desarrollo basado en ramas

Proceso de desarrollo basado en ramas partiendo de 2 ramas de forma continua:

- **master:** Desarrollo limpio. Sólo versiones estables.
- **develop:** El desarrollo inicial de la versión actual tiene lugar aquí.

Gestión de las ramas para estabilización de las versiones:

- **release-0.1, release-0.2;** una rama de estabilización cada vez

8.3.1. Proceso de estabilización

El proceso de estabilización se gestiona a través de las *ramas de estabilización*:

- Estabilización del código (*RC release candidates*)
- Arreglar bugs (hotfixes)
- Cuando la versión se considera estable se procede al siguiente paso:
 - Tag
 - Mezclar (merge) con development
 - Mezclar (merge) con master
- Si surgen nuevos bugs se vuelve a repetir el **proceso de estabilización**:
 - Se arreglan en la misma rama (release-0.1)
 - Nuevo tag y mezcla

¹<http://www.liferay.com/web/neil.griffin/blog/-/blogs/fixing-suncertpathbuilderexception-caused-by>

8.3.2. Releasing

Para crear una Release se define un proceso de gestión a través las ramas:

- Checkout del tag
- Build (Jenkins)
- Deploy (Jenkins)

8.3.3. Diagrama de desarrollo

El flujo de trabajo a través de las ramas se representa en este diagrama:

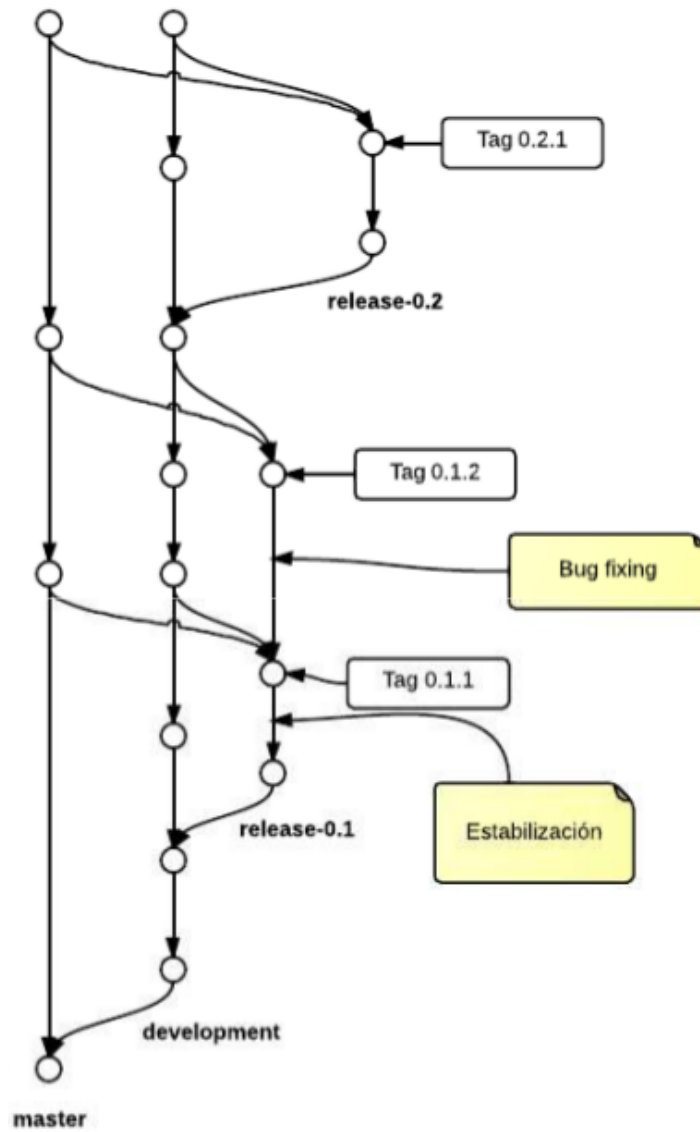


Figura 8.8: Flujo de desarrollo Git.

Capítulo 9

Epílogo

9.1. Conclusiones

El uso de estas herramientas y su incremento de la calidad en el desarrollo, por encima de todo siendo FLOSS debido a eso la versatilidad que otorga en el momento de unificarlas en una herramienta nueva; SidelabCode Stack.

9.2. Lecciones aprendidas

Colaboración entre distintos proyectos y comunidades, interoperabilidad entre herramientas, Forjas de desarrollo y los elementos más comunes de las mismas.

9.3. Trabajo Futuro

Impulso de la comunidad a través de los canales habituales.

Integración y gestión de nuevas herramientas comunes para los desarrolladores.

Centralización de la instalación.

Apéndices

Apéndice A

Puppet

Instalación de Puppet siguiendo la documentación ofrecida en su página web¹.

El proceso automatiza la instalación y configuración manual de un componente, un servidor web.

- Primero se ha de crear la instalación manual del servidor en un entorno definido (Sistema operativo Ubuntu). Configurar los permisos, definir archivos de configuración, directorios a servir, públicos/privados.
- Recapitular todos los pasos seguidos en la instalación y la información sobre la configuración para reescribirla a un módulo Puppet.
- Crear el esqueleto del módulo Puppet.
- Definir las órdenes a ejecutar organizando el módulo en distintas clases y apartados cada una responsable de una funcionalidad.
 - Descargar servidor web.
 - Instalar servidor web.
 - Configurar servidor web a través de una plantilla Ruby basada en los ficheros de configuración base del servidor web. Directorios, usuarios, urls, dominios, web

¹Puppet Documentation - <http://docs.puppetlabs.com/guides/installation.html#debian-and-ubuntu>

de bienvenida, etc.

- Poner en marcha el servidor.

Puppet da la opción de publicar los módulos desarrollados en un repositorio central de módulos para que estén accesibles para los usuarios de Puppet con un sistema de búsqueda para ayudar a los usuarios.

Con este módulo Puppet podemos replicar la instalación en cualquier entorno con tan solo ejecutar el módulo a través de Puppet para la ejecución. Descargar el módulo y ejecutar el commando Puppet.

Código Puppet de ejemplo para la instalación del servidor web Apache a través del módulo Puppet publicado en PuppetForge²:

Instalar Apache importando la clase del módulo:

```
class { 'apache': }
```

Módulo PHP de Apache:

```
class { 'apache::mod::php': }
```

Configurar Host Virtual mediante los distintos parámetros de configuración, partiendo del mínimo un ejemplo sería este:

```
apache::vhost { 'www.example.com':
  priority      => '10',
  vhost_name    => '192.0.2.1',
  port         => '80',
}
```

Se pueden definir más parámetros de configuración:

```
apache::vhost { 'www.example.com':
  priority      => '10',
  vhost_name    => '192.0.2.1',
  port         => '80',
  docroot       => '/home/www.example.com/docroot/',
  logroot       => '/srv/www.example.com/logroot/',
  serveradmin   => 'webmaster@example.com',
  serveraliases => ['example.com'],
}
```

²Puppet Apache Module - <https://forge.puppetlabs.com/puppetlabs/apache>

Apéndice B

Instalación SidelabCode Stack

En la versión 0.4 de SidelabCode Stack se utiliza Puppet en el proceso de instalación. Ahora es extremadamente sencillo instalar SidelabCode Stack usando Puppet o Vagrant con el nuevo instalador.

Puppet nos permite automatizar la instalación en un servidor *Ubuntu* mediante una sencilla configuración.

El uso de *Vagrant* permite probar SidelabCode Stack en una máquina virtual en 5 minutos.

B.1. Requisitos

Para la instalación de SidelabCode Stack es necesario obtener los siguientes recursos.

Dependiendo de la instalación que se utilice |Vagrant o |Puppet se han de seguir instrucciones diferentes.

B.1.1. Descarga SidelabCode Stack

Descargar el instalador de SidelabCode Stack (módulo *scstack*) y sus dependencias de la url:

- `http://code.sidelab.es/public/sidelabcodestack/artifacts/0.3/puppet-installer-0.3-bin.tar.gz`

```
cd $HOME
mkdir tmp
cd $HOME/tmp
wget http://code.sidelab.es/public/sidelabcodestack/artifacts/0.3/puppet-installer-0.3-bin.tar.gz
```

Descomprimir y copiar a la carpeta modules:

```
cd $HOME/tmp
tar xvfz puppet-installer-0.3-bin.tar.gz
```

B.1.2. Configuración de módulos puppet

Creemos un fichero `default.pp` en la carpeta `tmp` con el siguiente contenido:

```
exec { "apt-update":
  command => "/usr/bin/apt-get_update",
}
class { "scstack":
  # Superadmin password. Will be used to access the SidelabCode Stack Console re@lity45
  sadminpass => "re@lity45",
  # Or whatever IP specified in Vagrantfile
  ip => "192.168.33.10",
  domain => "sidelabcode03.scstack.org",
  baseDN => "dc=sidelabcode03,dc=scstack,dc=org",
  # Your company/organization name
  compname => "SidelabCode_stack_version_0.3",
  # A name to be displayed within Redmine
  codename => "SCStack_ALM_Tools",
}
```

B.1.3. Apt Cacher

Dependiendo de la conexión de red, el proceso de instalación puede tardar más o menos. En general es buena idea configurar un proxy para los paquetes debian. Esta opción es recomendable en el proceso de instalación a través de Vagrant. Para ello, simplemente hay que instalar **apt-cacher** en el host:

```
sudo apt-get install apt-cacher
```

Modificar las siguientes líneas del fichero `/etc/apt-cacher/apt-cacher.conf`:

```
daemon_addr = 192.168.33.1 # No podemos usar localhost si queremos que los clientes se puedan
conectar
allowed_hosts = * # Permitir a todos los clientes conectarse a este proxy.
generate_reports = 1 # Generar un informe cada 24h
```

Reiniciar el servicio:

```
$ sudo /etc/init.d/apt-cacher restart
```

Modificar el fichero `/etc/hosts` con la ip del host y el dominio asociado que se define en el fichero de configuración `default.pp` y la ip y el dominio del cliente para la comunicación con apt-cacher definido en el parámetro `daemon_addr` del fichero `/etc/apt-cacher/apt-cacher.conf`:

```
192.168.33.10    sidelabcode03.scstack.org
192.168.33.1    host.scstack.es
```

Añadir el siguiente trozo de código en la primera línea del fichero `default.pp` para que utilice el apt-cacher creado:

```
file { "/etc/apt/apt.conf.d/01proxy":
    content => 'Acquire::http::Proxy_"http://192.168.33.1:3142/apt-cacher";',
}
```

B.2. Vagrant

La instalación a través de Vagrant permite probar SidelabCode Stack en una máquina virtual en 5 minutos.

B.2.1. Descripción del entorno de instalación

Básicamente lo que vamos a hacer es indicar a Vagrant que monte una red privada entre la máquina host y la máquina virtual donde instalaremos SidelabCode Stack. Esto nos

permite tener acceso a la forja desde el host. Normalmente, Vagrant asigna, dentro de esa red privada, la IP 192.168.33.1 al host y la IP 192.168.33.10 a la máquina virtual. Estos valores se pueden cambiar como veremos posteriormente.

B.2.2. Prerequisitos

- Instalar Vagrant y Virtualbox (descargar las últimas versiones de las respectivas páginas web)
- Añadir Vagrant al path

```
$ gedit $HOME/.bashrc
$ PATH=$PATH:/opt/vagrant/bin
```

B.2.3. Preparación de la VM

Utilizaremos Vagrant para provisionar una VM con Java donde poder instalar SidelabCode Stack. Preparamos la carpeta para el proyecto Vagrant.

```
$ mkdir vagrant
$ mkdir vagrant/manifests
$ mkdir vagrant/modules
```

Descargar una imagen vagrant (precise64 es la que elegimos en esta documentación, por ser LTS).

```
$ vagrant box add precise64 http://files.vagrantup.com/precise64.box
```

Si quiere utilizar una versión distinta de una distribución, puede seleccionarla en la lista de **boxes** que proporciona vagrant para virtualbox.

Crear un proyecto Vagrant:

```
$ vagrant init
```

Modificar el Vagrantfile que describe el proyecto (VM, provisioner, etc) para que arranque la vm “precise64” y utilice Puppet:


```
config.vm.box = "precise64"
config.vm.network :hostonly, "192.168.33.10" # Si se desea se puede utilizar el modo bridge y
      asignar una ip por dhcp dentro de la red en la que se encuentra el host
config.vm.provision :puppet, :module_path => "modules"
```

Copiar el instalador de la forja en la carpeta modules del proyecto Vagrant:

```
$ cp -R puppet-installer -0.3/* $HOME/vagrant/modules
```

Copiar el fichero default.pp en la carpeta manifests del proyecto Vagrant:

```
$ cp /tmp/default.pp $HOME/vagrant/manifests
```

Arrancar la vm.

```
$ cd $HOME/vagrant
$ vagrant up
```

B.3. Puppet

TBC: Descripción

B.3.1. Pre requisitos

Puppet se ha de instalar mediante el gestor de paquetes de la distribución, en este caso apt para Ubuntu:

```
$ [sudo] apt-get install puppet
```

La instalación en otras distribuciones se puede consultar en el manual de puppet.

B.3.2. Configuración

Copiar los módulos puppet al directorio de módulos definido:

```
$ mkdir -p $HOME/puppet/modules
$ cp -R puppet-installer -0.3/* $HOME/puppet/modules
```

Ejecutar puppet para el proceso de instalación mediante sudo:

```
$ sudo puppet apply --modulepath=$HOME/puppet/modules default.pp
```

B.4. Post instalación

El proceso de instalación comenzará automáticamente. Una vez finalizado, en la dirección <http://test.scstack.org/redmine> se mostrará Redmine. La consola es accesible a través de la dirección <https://test.scstack.org:5555>. Es posible administrar scstack accediendo con el usuario *sadmin* y la contraseña especificada en el parámetro *sadminpass*.

Nota: La consola de administración se ha comprobado el funcionamiento para los siguientes navegadores:

- Firefox.
- Chrome/Chromium

B.4.1. Primeros pasos

Después de la instalación automatizada del entorno se ha de acceder a las herramientas **Redmine** y **Archiva** para completar el proceso.

Antes de nada se ha de reiniciar la máquina para comprobar que se ejecutan todos los procesos en el inicio.

Redmine

Modificar la fecha de creación de la api key de admin desde mysql:

```
$ mysql -u root -p
> use redminedb;
> update tokens set created_on='2013-01-16 22:16:00' where id='1';
```

Modificar los permisos de las carpetas en `/opt/redmine/tmp` para que sean del usuario de apache:

- `cd /opt/redmine/tmp && chown -R www-data:www-data *`

Acceder a la URL de Redmine, con el usuario `admin` y el password `admin`:

- <https://test.scstack.org/redmine>

Cambiar la contraseña para que no utilice la genérica a través de la ruta:

- Administración > Users > admin > Authentication > actualizar.

Cambiar la API key para securizar el acceso a la API rest (scstack instala una por defecto, pero no es segura):

- Acceder como admin > My account > API access key > Reset > Show -> Copiar la key
- Pegar la key en el fichero `/opt/scstack-service/scstack.conf`
- Reiniciar el servicio scstack: `sudo service scstack-service restart`

Gerrit

El primer usuario que accede a Gerrit obtiene privilegios de administrador. Al instalar la forja, se recomienda crear un usuario “gerritadmin” y password “t0rc0zu310” y acceder con este usuario a Gerrit. Este usuario se convertirá en administrador automáticamente al hacer login. A partir de este momento, este será el usuario con el que crear los grupos y proyectos (repositorios) en Gerrit.

Obtener la clave pública del servidor para asignarla al usuario **gerritadmin**:

1. *Settings > SSH Public Keys> Add.*
2. Copiar la clave del fichero `/opt/ssh-keys/gerritadmin_rsa.pub`.

Configurar permisos para creación de proyectos:

1. Acceder a *Projects > List> All-Projects*.
2. Seleccionar *Access*:
3. *Editar*:

- Bloque **refs/** Add Permission > añadir *Push* el grupo *Administrators*.
- Bloque **refs/meta/config** Add Group > añadir a *Read* el grupo *Administrators*.
- Save Changes.

Archiva

Acceder a la URL de Archiva:

- `https://test.scstack.org/archiva`

La primera vez que se configura archiva pide los datos del administrador. Apuntarlos convenientemente para posteriores necesidades de administración. Se recomienda utilizar la contraseña del administrador de la forja definida en el fichero de configuración `default.pp`.

Repositorios Por defecto, Archiva trae configurado un **repositorio internal** que hace de proxy de *Maven Central* y *java.net*. Si hace falta añadir repositorios remotos adicionales, en Repositories, al final de la página se pueden añadir repositorios remotos.

Archiva trae configurado un **repositorio de snapshots**. Se recomienda crear uno de **releases**.

Para ello accedemos a la administración de Archiva Administration -> Repositories y añadimos uno nuevo con los siguientes parámetros:

```
Identifier*: *releases*
Name*:      *Archiva Managed Releases Repository*
Directory*: */opt/tomcat/data/repositories/releases*
...
Repository Purge By Days Older Than: *30*
```

Se crea el repositorio. Si Tomcat nos muestra un error por pantalla al acceder a la URL `https://test.scstack.org/archiva/admin/addRepository!commit.action` relacionado con **NullPointerException** no nos debemos preocupar ya que el repositorio está creado correctamente siendo accesible y funcional. Se puede comprobar volviendo a visualizar la lista de repositorios de *Archiva*.

Usuarios Archiva no lee los usuarios de *OpenLDAP*, por tanto es necesario añadirlos a mano. En principio, debería ser suficiente con un **usuario de deploy** para *toda la organización*, o como mucho, un usuario por proyecto o grupo de proyectos.

El usuario debe ser **Observer** de los tres repositorios y **manager** de snapshots y releases.

B.5. FAQ

Posibles problemas que se pueden encontrar tras la instalación:

- Error al crear un repositorio Git: Reiniciar el servicio `scstack-service`:

```
$ sudo service scstack-service stop
$ sudo service scstack-service start
```

- Error al acceder a **Archiva** o **Jenkins** 404 No encontrado.

- Configurar el dominio de nombres en el ordenador para que acceda a través de la ip correspondiente.

```
$ sudo vi /etc/hosts
```

- Añadir la línea de conversión entre IP y nombre (elegido en la configuración del fichero `default.pp`).

```
138.100.156.246 sidelabcode03.scstack.org
```


Bibliografía

- [1] Larman, Craig and Basili, Victor R. *Iterative and incremental developments. a brief history*. IEEE, 2003.
- [2] Fowler, Martin. *Feature Branch*. <http://martinfowler.com/bliki/FeatureBranch.html>, September 2009.
- [3] Jose Angel Diaz Diaz Manuel Velardo Pacheco. *INFORME TÉCNICO Forjas: entornos de desarrollo colaborativo. Su integración en el ámbito empresarial..* CENATIC (Centro Nacional de Referencia de Aplicación de las TIC basadas en fuentes abiertas), 2009.
- [4] Dirk Riehle, John Ellenberger, Tamir Menahem, Boris Mikhailovski, Yuri Natchetoi, Barak Naveh, Thomas Odenwald. *Open Collaboration within Corporations Using Software Forges*. IEEE Software, March/April 2009.
- [5] Micael gallego. *Mamá, yo de mayor quiero una forja (de desarrollo software)*. <http://sidelab.wordpress.com/2011/07/05/mama-yo-de-mayor-quiero-una-forja-de-desarrollo-software/>, 2011.
- [6] Jonathan Rosenberg. *The meaning of open*. Google Official Blog <http://googleblog.blogspot.com.es/2009/12/meaning-of-open.html>, December 2009.
- [7] Wikipedia. *Comparison of open-source software hosting facilities*. http://en.wikipedia.org/wiki/Comparison_of_open_source_software_hosting_facilities, 2013.
- [8] Jesus M. Gonzalez-Barahona, *The history of FusionForge and GForge*. <http://blog.bitergia.com/2012/11/16/the-history-of-fusionforge-and-gforge/>, Bitergia's blog, November 16 2012

- [9] Robert C. Martin, *Clean Code*, Chapter 9 Unit Test, Prentice Hall, 2008.