Capítulo 1

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 infiere 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.

1.1. Software de Calidad

Desarrollar Software de Calidad, para ellos nos encontramos con la palabra *Çalidad"* 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.

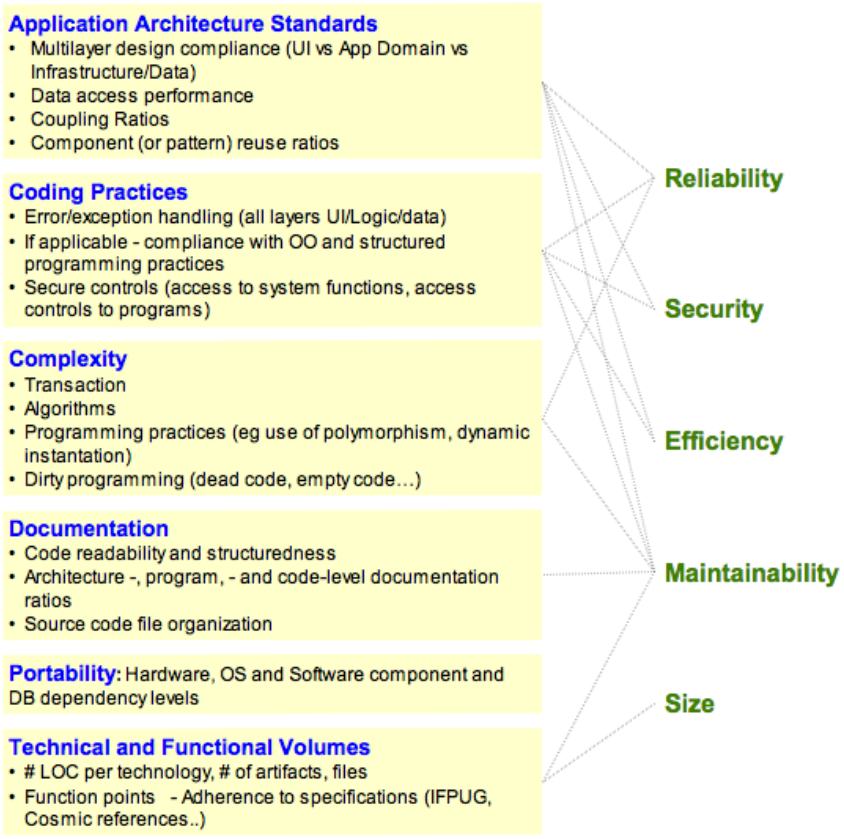

Figura 1.1: Software de Calidad

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.

1.2. 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. Teniendo en cuenta posibles cambios entre iteraciones pero nunca del resultado completo, así controlando 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.

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

[1] Procesos Iterativos e Incrementales - `http://esalas334.blogspot.es/1193761920/`

[2] Proceso de Desarrollo Iterativo - Fernando Soriano - `http://fernandosoriano.com.ar/?p=13`

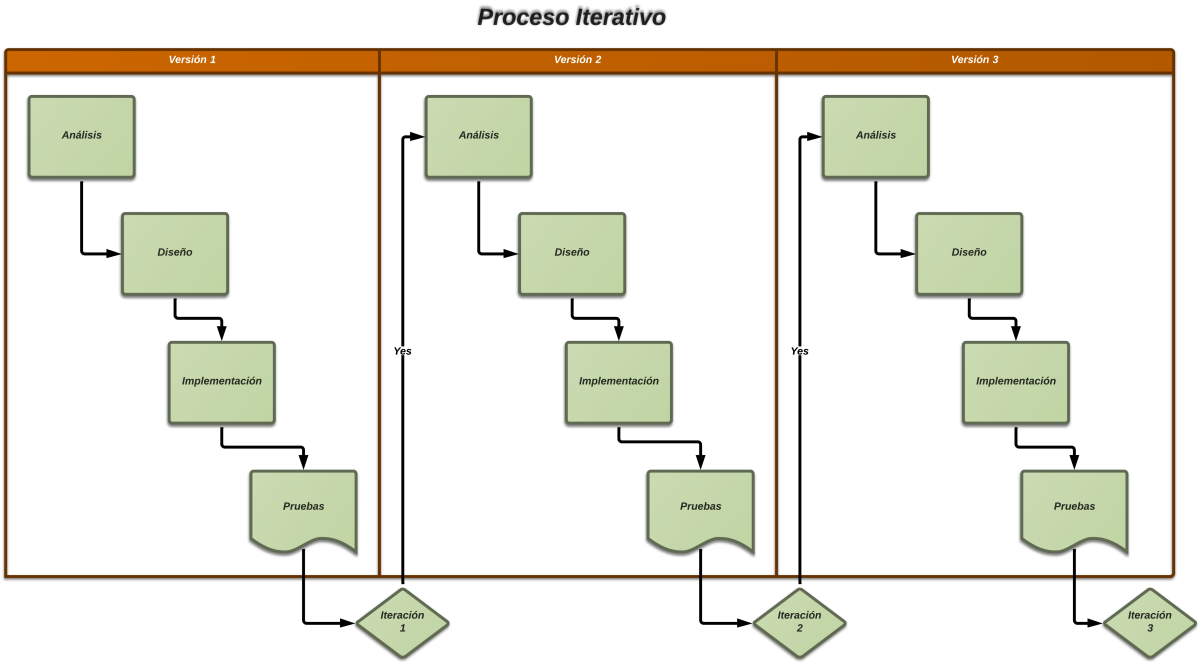

Figura 1.2: Proceso Iterativo

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.

1.3. 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 *ïncremento"* 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 del 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.

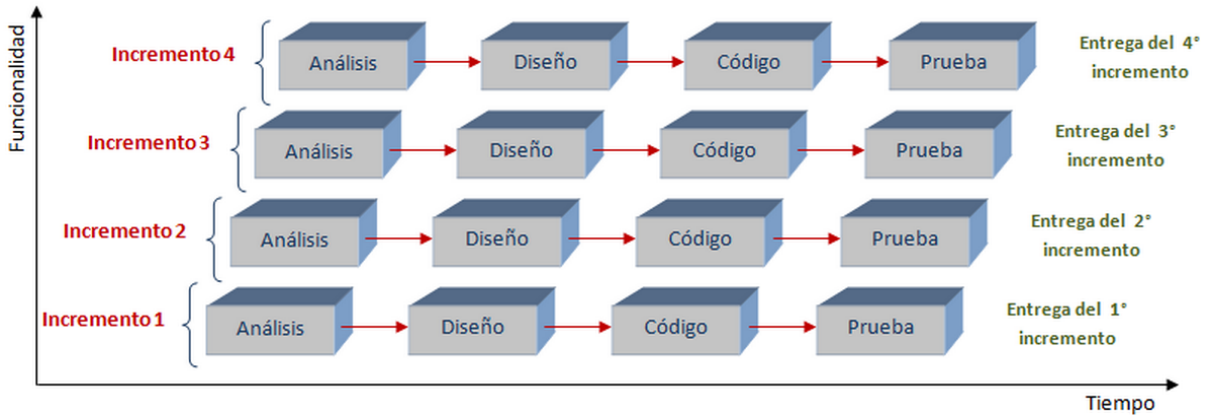

Figura 1.3: Modelo Incremental

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 que el sistema desarrollará.

1.4. Iterativo e Incremental

Desarrollo iterativo e incremental

1.5. Gestión de tareas

Gestión de tareas: ¿qué hay que hacer? ¿quién tiene que hacerlo? ->Sistemas de gestión de tickets

1.6. Código versionado

Código versionado ->repositorios de código

1.7. TDD y CI

TDD ->sistemas de CI para asegurar que los tests se pasan regularmente

1.8. Desarrollo por canales

1.9. Herramientas

Capítulo 2

ALM Tools

* ALM Tools - Forjas de desarrollo * Qué es una forja * Objetivos * Componentes * Estudio del arte de forjas * Problemática ->administración, costes... * Tablas comparativas * Algunos ejemplos y sus limitaciones * Conclusiones del estudio de forjas

ALM Tools significa: *Application Lifecycle Management Tools*. La gestión del ciclo de vida de una aplicación, el conjunto de herramientas encargadas de guíar al desarrollador a través de un camino basado en metodologías para la creación de un Software hacia su estado del arte.

Integrar el proceso de desarrollo de Software a través de las ALM Tools como vehículo, es decir no existe en si mismo una herramienta ALM, sino que la herramienta ALM gobierna a las herramientas incluidas en el proceso de desarrollo.

2.1. 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, la necesidad de interoperabilidad entre la herramientas a través de una comunicación estándar. Aquí encontramos el punto clave de las ALM Tools, la **integración de 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 hacia adelante, no es necesario implementar las mejoras pero sí, dejar un hueco para que casen bien. Un ejemplo que puede ilustrar esta frase es la programación basada en Interfaces mediante Java, ya que las Intefaces ofrecen soluciones para implementar a medida y 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 pero nueva, de esta forma la interoperabilidad entre las clases que utilicen esta Interfaz también se mantiene.

2.2. Qué es una forja

2.3. Objetivos

2.4. Componentes

Este conjunto de herramientas se puede dividir en varios grupos:

- Gestión de Requisitos.
- Arquitectura.
- Desarrollo.

- Test.
- Issue tracking system.
- Continuous Integration.
- Release Management.

Hoy en día las forjas ALM abundan, además de gozar de una gran popularidad entre los proyectos de Software, como podemos ver en los casos de SourceForge, Googlecode y Github (más adelante discutiremos cada proyecto). En este caso ALM Tools as a Service, debido al servicio que ofrecen, pero sólo las que son FLOSS permiten replicar ese mismo entorno en tu propia máquina, un dato muy importante a tener en cuenta, porque siempre se ha de mirar hacia adelante.

2.5. Estado del arte de forjas

- ClinkerHQ `http://clinkerhq.com/` - Privado.
- Github - `http://github.com/` - Privado.
- SourceForge con Allura - `http://sourceforge.net/projects/allura/` - Software Libre pero incompleta (integrated Wiki, Tracker, SCM (svn, git and hg), Discussion, and Blog tools).
- Cloudbees DEV@Cloud `http://www.cloudbees.com/dev.cb` - Privado.
- CollabNet con CloudForge `http://www.cloudforge.com/` - Privado.
- Plan.io - `http://plan.io/en/` - Privado.
- Bitnami - `http://bitnami.com/` - Privado
- GForge
- Collab.net
- Google Code
- Bitbucket - https://bitbucket.org/mswlmanage2013/mswl-bitbucket-alm-tools/wiki/Home

2.5.1. Problemas en algunas forjas

2.5.2. Tablas comparativas

2.6. Conclusiones del estudio de forjas

Capítulo 3

SCStack

* SCStack

3.1. Arquitectura

3.1.1. Diseño e Implementación

Análisis técnico de las Herramientas utilizadas.

3.2. Provisionamiento (puppet)

3.3. Componentes

3.4. Interoperabilidad

Interoperabilidad entre las herramientas que componen la forja. API Rest.

3.5. Pruebas y Validación

Pruebas a través de virtualización de sistemas operativos mediante herramientas libres; Vagrant, kvm, puppet.

Capítulo 4

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 5

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 gerrit + jenkins.

5.1. Estructura de un proyecto

5.1.1. subsection name

5.1.2. Tareas

5.1.3. Binarios

Binarios y/o fuentes empaquetados en lenguajes que no generan binarios.

5.2. Crear un proyecto

5.2.1. Proyecto en redmine

5.2.2. Repositorio git

5.2.3. Jobs en Jenkins

5.2.4. Repositorios Archiva

repositorios de archiva para binarios.

5.3. Alta usuarios

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

5.4. Proceso de desarrollo basado en ramas

Proceso de desarrollo basado en ramas.

Capítulo 6

Epílogo

6.1. 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.

6.2. Lecciones aprendidas

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

6.3. 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.

Apéndice A

Apéndice 1

Bibliografía

[1] Timothy Budd. *Introducción a la programación orientada a objetos*. Addison-Wesley Iberoamericana, 1994.

[2] Mark Lutz. *Programming Python*. O'Reilly, 2001.

[3] Fredrik Lundh. *Python standard library*. O'Reilly, 2001.

[4] George Reese. *Managing and using MySQL*. O'Reilly, 2002.