



**UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA**

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

**Implantación de un sistema de integración
continua en una metodología consolidada**

Julio Alberto Fernández Guerrero

Junio, 2017

**IMPLANTACIÓN DE UN SISTEMA DE INTEGRACIÓN CONTINUA EN UNA
METODOLOGÍA CONSOLIDADA**



**UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA**

Tecnologías y Sistemas de Información

**TECNOLOGÍA ESPECÍFICA DE
INGENIERÍA DEL SOFTWARE**

TRABAJO FIN DE GRADO

**Implantación de un sistema de integración
continua en una metodología consolidada**

Autor: Julio Alberto Fernández Guerrero

Director: Manuel Ángel Serrano Martín

Director: Miguel Ángel Laguna Lobato

Junio, 2017

Julio Alberto Fernández Guerrero

Ciudad Real – Spain

E-mail: JulioAlberto.Fernandez@alu.uclm.es

Teléfono: (+34) 655 398 684

© 2017 Julio Alberto Fernández Guerrero

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Se permite la copia, distribución y/o modificación de este documento bajo los términos de la Licencia de Documentación Libre GNU, versión 1.3 o cualquier versión posterior publicada por la *Free Software Foundation*; sin secciones invariantes. Una copia de esta licencia esta incluida en el apéndice titulado «GNU Free Documentation License».

Muchos de los nombres usados por las compañías para diferenciar sus productos y servicios son reclamados como marcas registradas. Allí donde estos nombres aparezcan en este documento, y cuando el autor haya sido informado de esas marcas registradas, los nombres estarán escritos en mayúsculas o como nombres propios.

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE

VOCAL

SECRETARIO

Fdo.:

Fdo.:

Fdo.:

Resumen

Hoy en día, muchas empresas desarrollan software desde un enfoque clásico pero a la vez intentan adaptarse a los nuevos tiempos intentando introducir las nuevas metodologías en sus proyectos, en especial, las metodologías ágiles. En el enfoque clásico que utilizan las empresas, el desarrollador tan pronto desarrolla como ejecuta tests unitarios al software, es decir, muchas empresas aún no conocen la importancia de separar el trabajo y, sobre todo, el ahorro en costes y tiempo que supone hacer esto.

Madrija Consultoría, S.L. no es una excepción, dado que aplica una metodología propia para gestionar el ciclo de vida de sus desarrollos pero los desarrolladores actúan tanto de desarrolladores como de testers, por lo que pretende introducir una nueva práctica de las metodologías ágiles para mejorar su metodología, en concreto, nacida de la mano de Martin Fowler, la integración continua, surgió con el objetivo de facilitar el trabajo en equipos de desarrollo y automatizar las tareas de integración. La integración continua se basa en la construcción automática de proyectos con frecuencia alta, promoviendo la detección de errores en un momento temprano para poder dar prioridad a corregir dichos errores.

Por lo tanto, se hace necesaria la implantación de un sistema de integración continua dentro de la empresa para alcanzar el mayor grado de automatización posible en sus pruebas de proyectos permitiendo así a los desarrolladores ahorrar tiempo en estas tareas para centrarse en otras, surgiendo así la necesidad de realizar un análisis, diseño e implantación de un sistema de integración continua dentro de la empresa.

Por ello, surge la motivación de desarrollar este trabajo fin de grado, dentro del convenio FORTE, con Madrija Consultoría, S.L.

Abstract

Nowadays, many companies develop software from a classical approach but now they try to adapt it to the new times, they are trying to introduce the new methodologies in their projects, especially the agile methodologies. In the classic approach, the developers do both things develop and executes unit tests, in other words, many companies still don't know the importance of separating the work and, above all, the savings in the costs and the time that means to do this.

Madrija Consultoría, S.L. isn't an exception, since it applies its own methodology for live cycle management but the developers act as much of the developers as of the testers, but they are trying to introduce a new practice of the agile methodologies to improve its methodology, in this case, continuous integration, this practice borned with the aim of facilitating work in development teams and automate integration tasks. The continuous integration is the automatic construction of projects with high frequency, promoting the detection of errors in an early moment to give priority to correct these errors.

Therefore, it is necessary to implement a system of continuous integration within the company to achieve the highest degree of automation possible in their project tests thus allowing developers to save time in these tasks to focus on others, arising in the need to perform an analysis, design and implementation of a system of continuous integration within the company.

For this reason, the motivation to develop this end-of-grade job, within the FORTE agreement, with Madrija Consultoria, S.L.

Agradecimientos

A mis padres y a mi hermana, por creer en mí desde el primer momento que comencé esta aventura, por ser mi motor, darme las fuerzas necesarias en este viaje próximo a finalizar y, sobre todo, por explicarme con su ejemplo el significado de las palabras trabajo y humildad.

A Manuel Ángel Serrano por la oportunidad de aprender contigo e ilustrarme con tus ideas.

A Miguel Ángel Laguna gracias por darme la oportunidad de aprender de ti tanto profesional como personalmente, gracias por tu paciencia conmigo, por tus consejos y el apoyo día a día durante el desarrollo de este trabajo.

A Madrija Consultoría, S.L., por la alegría de cada día en la oficina incluso en los peores momentos.

A Ismael Caballero, Mario Piattini y Macario Polo, gracias por guiarme y aconsejarme cuando tuve dudas en esta aventura.

A mis amigos y amigas de Miguelturra, tanto a esa *Partida* como a esos *Chaveles*, a todos y cada uno de vosotros, por permitirme formar parte de vuestras vidas, y por ser tanto mis apoyos, como mis vías de escape en los malos momentos, con vosotros a mi lado esta aventura ha sido posible.

A todos y cada uno de los amigos que me ha brindado, la que posiblemente sea la mejor etapa de mi vida, desde el primero al último, Alberto, Álvaro, Edu, Jaime, Javi, Jorge, José María, María, Oliva, Pablo y Pedro, por esas *Friki Night's*, tanto de fiesta, como de estudio, que siempre terminan entre risas, por las experiencias y momentos vividos a vuestro lado, no me imagino en esta etapa de mi vida sin vosotros.

Gracias a todos, de corazón.
Julio Alberto Fernández Guerrero

No se sale adelante celebrando éxitos sino superando fracasos.

- Orison Swett Marden -

Índice general

Resumen	V
Abstract	VII
Agradecimientos	IX
Índice general	XIII
Índice de cuadros	XVII
Índice de figuras	XIX
Listado de acrónimos	XXI
1. Introducción	1
1.1. Estructura del documento	4
2. Objetivos	5
2.1. Objetivo principal	5
2.2. Objetivos parciales	5
2.2.1. Analizar y determinar herramientas de integración continua	5
2.2.2. Desarrollar la arquitectura e infraestructura para el sistema de integración continua	6
2.2.3. Adaptarse a la metodología de desarrollo	6
2.2.4. Alcanzar el máximo grado de automatización posible de los tests	6
2.2.5. Proponer mejoras	6
2.3. Justificación	6
3. Estado de la cuestión	7
3.1. Ingeniería del Software	7
3.2. ISO 12207	7
3.3. Inicios de la integración continua	8

3.4. Integración continua	9
3.4.1. Desarrollo ágil	11
3.4.2. DevOps	11
3.5. Características y requisitos de la integración continua	12
3.6. Herramientas	18
3.6.1. Bamboo	18
3.6.2. CircleCI	18
3.6.3. Jenkins	19
3.6.4. Solano CI	19
3.6.5. TeamCity	19
3.6.6. Travis CI	20
3.6.7. Comparativa	20
4. Método de trabajo	25
4.1. Introducción	25
4.2. Heurística IDEAL	25
4.2.1. Identificación del problema	26
4.2.2. Realización del plan	26
4.2.3. Ejecución del plan	26
4.2.4. Analizar la solución	26
4.3. Aplicación del método de trabajo	27
4.4. Planificación de las iteraciones	30
4.5. Marco tecnológico	31
4.5.1. Medios software	31
4.5.2. Medios hardware	33
5. Resultados	35
5.1. Resultado	35
5.2. Iteraciones	39
5.2.1. Iteración 1	39
5.2.2. Iteración 2	42
5.2.3. Iteración 3	43
5.2.4. Iteración 4	46
6. Conclusiones	49
6.1. Análisis de los objetivos del proyecto	49

6.2. Trabajo futuro	51
6.3. Opinión personal	51
Referencias	53
A. Descripción de competencias	59
B. Manual de usuario	63
C. Contrato de propiedad intelectual	67

Índice de cuadros

3.1. Comparativa de tipo de licencia	20
3.2. Comparativa de <i>hosting</i>	21
3.3. Comparativa de envío de notificaciones	21
3.4. Comparativa de <i>builders</i>	22
3.5. Comparativa de integraciones extras	23
4.1. Iteraciones	30
6.1. Objetivo principal	49
6.2. Análisis de los objetivos parciales	50
A.1. Tecnología específica cursada por el alumno	59
A.2. Justificación de las competencias específicas abordadas en el TFG	61

Índice de figuras

1.1.	Logo de Madrija Consultoría, S.L.	3
3.1.	Ciclo de eXtreme Programming (XP)[38]	8
3.2.	Algunos principios del <i>Manifiesto para el Desarrollo de Software Ágil</i> [5] .	10
3.3.	Perfil Develop and Operations (DEVOPS)	12
3.4.	Repositorios y sistemas de control de versiones	13
3.5.	Algunas acciones de Git[13].	15
3.6.	Panel principal de Jenkins	19
3.7.	Herramientas de integración continua	20
4.1.	Ciclo IDEAL	26
4.2.	Imágenes de OpenProject de la iteración 1	27
4.3.	Algunas versiones del proyecto en OpenProject	28
4.4.	Ejemplo de la información que se muestra en la wiki de la iteración 1 (Parte 1)	29
4.5.	Ejemplo de la información que se muestra en la wiki de la iteración 1 (Parte 2)	29
4.6.	Algunas tecnologías utilizadas en este desarrollo	31
5.1.	Arquitectura e infraestructura del sistema de integración continua	37
5.2.	Ejecución y muestra de resultados de “Renombrator”	38
5.3.	Parte del código desarrollado para los tests de “Renombrator”	39
5.4.	Test desarrollado en Java con Selenium	40
5.5.	Indicación de objetivos de un proyecto Maven en Jenkins	41
5.6.	Diagrama de Gantt de la iteración 1	42
5.7.	Diagrama de Gantt de la iteración 2	43
5.8.	Versión inicial de la arquitectura e infraestructura del sistema de integración continua	44
5.9.	Detección de <i>commit</i> en un proyecto Jenkins	45
5.10.	Diagrama de Gantt de la iteración 3	45
5.11.	Diagrama de Gantt de la iteración 4	47

B.1.	Petición de contraseña maestra de Jenkins	63
B.2.	Panel de bienvenida de Jenkins	64
B.3.	Parámetros de configuración de Jenkins	64
B.4.	Parámetros de configuración de un esclavo Jenkins	65
B.5.	Configuración de un proyecto Jenkins	65
B.6.	Resultados tras ejecutar un proyecto Jenkins	66

Listado de acrónimos

BBDD bases de datos

BDD Behavior Driven Development

DEVOPS Develop and Operations

IDE Integrated Development Environment

ISO International Organization for Standardization

MADRIJA Madrija Consultoría, S.L.

TDD Test Driven Development

XP eXtreme Programming

Capítulo 1

Introducción

MAdrija Consultoría, S.L. es una empresa dedicada al desarrollo software, dentro de la empresa destacan dos proyectos, los cuales definen la temática de los desarrollos que se realizan en Madrija Consultoría, S.L. (MADRIJA), una línea dedicada al desarrollo de software sanitario, dicho software es desarrollado para su venta a hospitales y otra línea enfocada al desarrollo de software para la administración de procesos selectivos de las empresas.

MADRIJA ha desarrollado su propia metodología de gestión del ciclo de vida del desarrollo que está en fase de consolidación. Dicha metodología se basa en principios de *frameworks* metodológicos ágiles y las normas International Organization for Standardization (ISO), en concreto, utiliza la ISO 12207[19].

Como se ha dicho antes, MADRIJA basa sus líneas de desarrollo en sus dos proyectos centrales, los cuales poseen una arquitectura y un desarrollo muy complejo, ya que dichos proyectos son multimodulares, con ello se justifica que el producto que se va a vender pueda ser dirigido a tantos clientes como se desee, puesto que la modularidad de sus proyectos permite que se adapte a los diferentes clientes, es decir, permite añadir funciones independientes entre los diferentes módulos de cada cliente sin que una función o un módulo completo dependa de otro.

Dentro del desarrollo de todos sus proyectos destacan dos grandes paquetes centrales:

- El paquete *Core*.
- El paquete *Demo*.

El paquete *Core*, es aquel que contiene la lógica de negocio de cada proyecto, el mapeo de sus datos o las inyecciones de dependencias. Los paquetes *Core* de cada proyecto siguen el principio de dotar a sus proyectos de tener una alta cohesión, pero un bajo acoplamiento con el resto de paquetes *Core*, permitiendo así la posibilidad de que, por ejemplo, el cliente 1 pueda comprar los productos 1, 2 y 3, mientras que un cliente 2 sea capaz de comprar los products 3, 4, 5 y 6 sin que tengan dependencias unos de otros.

El paquete *Demo*, es un proyecto que contiene el código mínimo que configura y hace uso del paquete *Core* respectivo, pero su funcionalidad dentro de los proyectos es ser utilizado para ejecutar las pruebas sobre él. Estas pruebas y sus tests correspondientes son orquestados y ejecutados por Maven[1]. A los diferentes proyectos de MADRIJA se les realizan diferentes tests:

- Tests unitarios.
- Tests de comandos.
- Tests de interfaz de usuario.

Los tests unitarios están dedicados a comprobar el correcto funcionamiento de las unidades básicas del código fuente.

Los tests de comandos están desarrollados con la finalidad de comprobar el correcto funcionamiento de los comandos desarrollados en cada proyecto. Tal y como se han desarrollado durante el transcurso de este proyecto, los tests de comando, son un tipo específico de test de integración.

Tests de interfaz de usuario, estos tests son utilizados para comprobar el correcto funcionamiento de las distintas interfaces de usuario que han sido desarrolladas dentro de cada proyecto, en estos tests se realizan pruebas tales como agregar parámetros a campos específicos o hacer *click* en determinados botones y comprobar que realizan la navegación entre las diferentes interfaces de manera correcta. Tal y como se han desarrollado los tests de interfaz de usuario durante este proyecto, los tests de interfaz de usuario, son un tipo de test específico de los tests de aceptación.

Al tener tantos clientes, MADRIJA debe realizar pruebas específicas para cada uno de ellos, dado que cada producto de cada cliente tiene partes específicas demandadas por ese cliente que no tienen los demás, además, cada producto también posee un desarrollo específico para adaptarse a las diferentes instalaciones que tiene cada cliente.

Por ello, se hace necesario preparar un entorno de pruebas específico para cada cliente. MADRIJA cuenta con tecnología específica que le permite realizar dichas adaptaciones:

- *Liquibase*.
- “Renombrator”.
- *Vagrant*.

Mediante el uso de la herramienta *Liquibase*, MADRIJA consigue que cada cliente tenga almacenados los distintos cambios que se realizan en su base de datos en un fichero XML, estos ficheros también son conocidos como “ChangeSet”.

“Renombrator” es un software desarrollado por MADRIJA que toma un proyecto y revisa todos sus ficheros, con el objetivo de corregir aquellos errores de nombrado para evitar conflictos con los diferentes tipos de bases de datos.

Mediante el uso de la herramienta *Vagrant*, MADRIJA puede crear y configurar entornos virtualizados para la ejecución de los tests y las pruebas anteriormente nombradas, es decir, *Vagrant* permite crear un entorno virtual mediante un comando, realizar las pruebas necesarias en ese entorno y mediante otro comando es capaz de destruir el entorno virtualizado para permitir volver al equipo al estado anterior.

Dada la importancia del software que desarrolla MADRIJA, su principal línea de desarrollo es el software dedicado a sanidad, en concreto a cardiología. MADRIJA debe asegurar de que las funciones básicas en todos sus clientes, independientemente de la versión del producto que tenga cada cliente, funcionen correctamente y no surjan errores.

En lo referente a la gestión de pruebas y automatización de testing de proyecto, MADRIJA utiliza Maven para orquestar sus pruebas como se ha dicho antes, lo que permite un grado de automatización a nivel de proyecto, pero presenta el inconveniente de que su ejecución debe ser manual, lo cual supone pérdidas de tiempo en tareas repetitivas que se pueden automatizar y la posibilidad de que, mediante el factor humano, se cometan errores a la hora de ejecutar las pruebas, pudiendo no realizar siempre las mismas pruebas a un desarrollo o existiendo la posibilidad de que no se realicen pruebas.

Para mejorar los procesos de MADRIJA, se propone la implantación de un entorno de integración continua[12]. Estos entornos, derivados de la práctica de eXtreme Programming[3], de aquí en adelante XP, permiten reducir el tiempo que los desarrolladores invierten en la detección de errores, facilitan la identificación de errores de regresión y permiten automatizar la ejecución de las pruebas.



Figura 1.1: Logo de Madrija Consultoría, S.L.

MADRIJA es consciente de estos beneficios y por esa razón, surge la motivación de realizar este trabajo de fin de grado, dentro del programa FORTE, el cual pretende mejorar y aumentar la automatización de la ejecución de sus tests, aumentando el alcance de las pruebas a nivel de integración entre proyectos.

1.1 Estructura del documento

El presente documento está compuesto por 6 capítulos y 3 anexos. A continuación se describe el contenido de cada uno de ellos.

- **Introducción:** Primer capítulo, se encarga de dar una visión general al lector exponiendo el tema sobre el que versa este trabajo, tanto el problema encontrado, como la solución que se propone, también se incluye la estructura del documento.
- **Objetivos:** Segundo capítulo, en este capítulo se presenta el objetivo principal que se desea alcanzar así como los subobjetivos.
- **Estado de la cuestión:** Tercer capítulo, en este capítulo se realiza una revisión completa sobre el tema a tratar, concretamente, se toma información relativa a los inicios y la evolución de la integración continua.
- **Método de trabajo:** Cuarto capítulo, en este capítulo se explica la metodología aplicada para gestionar el proyecto, así como las iteraciones en las que se ha dividido.
- **Resultados:** Quinto capítulo, en este capítulo se exponen los resultados obtenidos durante el desarrollo del proyecto.
- **Conclusiones:** Sexto y último capítulo, se exponen las conclusiones obtenidas tras finalizar el trabajo y el desarrollo, en dicho capítulo se incluyen también propuestas de trabajos futuros, además de una opinión personal.

Tras estos capítulos se incluye las referencias que han sido consultadas y citadas durante el desarrollo de este documento.

En la parte final del presente documento se presentan los 3 anexos nombrados antes, se incluyen para ampliar y aclarar información sobre el proyecto tratado y que así el lector tenga una mejor comprensión de los temas tratados.

- **Anexo A:** Descripción de competencias.
- **Anexo B:** Manual de usuario.
- **Anexo C:** Contrato de propiedad intelectual.

Capítulo 2

Objetivos

Ado el enfoque y las ideas presentadas en la introducción, este capítulo versa sobre el objetivo principal de este trabajo, así como los objetivos parciales que se pretenden alcanzar durante el desarrollo del mismo. Se definen así una serie de hitos a alcanzar para conseguir el objetivo final.

2.1 Objetivo principal

El objetivo de este trabajo es analizar, diseñar e implementar un entorno de integración continua, que se adapte a la metodología, formas y prácticas de desarrollo que tiene MADRIJA.

Se desea destacar que este proyecto no versa sobre el desarrollo de un producto, ni tampoco sobre el desarrollo de software.

Los subobjetivos a tratar en este trabajo son:

- Analizar y determinar herramientas de integración continua.
- Desarrollar la infraestructura para el sistema de integración continua.
- Adaptarse a la metodología de desarrollo.
- Alcanzar el máximo grado de automatización posible de los tests.
- Proponer mejoras.

2.2 Objetivos parciales

2.2.1 Analizar y determinar herramientas de integración continua

MADRIJA es consciente de los beneficios de implantar un entorno de integración continua, por ello, requiere que se realice un estudio para analizar las herramientas y elegir la que más beneficie y mejor se adapte a la metodología de desarrollo de la empresa.

2.2.2 Desarrollar la arquitectura e infraestructura para el sistema de integración continua

MADRIJA no cuenta con un sistema de integración continua por lo que se deberá analizar, diseñar y desarrollar una arquitectura e infraestructura para el sistema de integración continua.

2.2.3 Adaptarse a la metodología de desarrollo

MADRIJA cuenta con una metodología propia de desarrollo, que será mejorada y complementada por la implantación del sistema de integración continua, para ello, se debe analizar y comprender la metodología utilizada por MADRIJA.

2.2.4 Alcanzar el máximo grado de automatización posible de los tests

MADRIJA realiza de manera manual, utilizando Maven, las pruebas a sus proyectos lo que supone pérdidas de tiempo, además, también influye el factor humano dado que existe la posibilidad de que no se realicen dos pruebas iguales a un mismo desarrollo en un mismo test o, incluso, que haya días en los cuales no se realicen pruebas.

2.2.5 Proponer mejoras

Para la implantación del entorno de integración continua, MADRIJA, deberá realizar una serie de cambios dentro de su metodología para que dicha implantación sea factible, es decir, se debe analizar la forma en la que desarrolla MADRIJA y estudiar la manera más fácil mediante la cual realizar dicha implantación.

2.3 Justificación

Este trabajo tiene varias causas que lo justifican entre las que destacan, el ahorro de tiempo, como se ha dicho antes, MADRIJA realiza las pruebas a sus proyectos de manera manual, utilizando Maven, software mediante el cual se orquestan una parte de las pruebas, pero necesita ser ejecutado por el equipo de desarrollo, por lo que cabe la posibilidad de que sucedan fallos humanos a la hora de realizar la ejecución de dichas pruebas, ocasionando que se cometa el error de no realizar siempre las mismas pruebas al mismo desarrollo.

Se hace necesario un análisis a fondo de la forma de desarrollo de la empresa para que la implantación del entorno de integración continua sea lo más factible posible.

Capítulo 3

Estado de la cuestión

EN este capítulo se presenta una visión general sobre los conceptos teóricos y técnicos que sirven como base para la elaboración de este documento, contribuyendo a su base tecnológica y de conocimiento.

Se desea resaltar que el presente proyecto no versa sobre el desarrollo de una aplicación software, sino que el objetivo principal del mismo, como ya se ha mencionado, es realizar un proceso de análisis y mejora de la metodología de MADRIJA para implantar un sistema de integración continua, por lo tanto, se realizará un recorrido analizando la contribución, posibilidades y aportaciones relevantes a las diferentes áreas de la informática que trata MADRIJA, justificando así la utilidad de desarrollar el presente trabajo.

3.1 Ingeniería del Software

La ingeniería del software es una disciplina, o área, de la informática[30], que ofrece métodos y técnicas para desarrollar y mantener software de calidad que resuelven problemas de todo tipo.

La ingeniería del software trata áreas muy diversas de la informática, tales como construcción de compiladores o sistemas operativos. Y, la que más concierne para este trabajo, la integración continua, abordando todas las fases del ciclo de vida del desarrollo de cualquier tipo de sistema de información.

3.2 ISO 12207

Tal y como se explica en el capítulo 1, MADRIJA utiliza la ISO 12207[19] para gestionar su ciclo de vida de desarrollo, ISO 12207 es el estándar que describe la arquitectura de los procesos de ciclo de vida del software de la organización ISO. La norma no describe detalles sobre como llevar a cabo estos procesos ni las tareas que componen los mismos. Además, establece un marco común para los procesos del ciclo de vida del software, con una terminología bien definida, que puede ser referenciada por la industria del software. Contiene procesos, actividades y tareas que se van a aplicar durante la adquisición de un producto o servicio de software y durante el suministro, desarrollo, operación, mantenimiento y eliminación de productos de software.

Por lo que MADRIJA ha implantando su propia metodología de desarrollo y dentro de dicha metodología se pretende implantar el sistema de integración continua para mejorarla.

3.3 Inicios de la integración continua

XP es una metodología de desarrollo de software que enfoca a todo el equipo en objetivos comunes y alcanzables[3]. Utilizando los valores y principios de XP, los equipos aplican prácticas apropiadas de XP en su propio contexto. Los equipos XP producen software de calidad a un ritmo sostenible.

XP surgió como una nueva manera de encarar proyectos software[20], proponiendo una metodología basada, en esencia, en la simplicidad y agilidad. Las metodologías de gestión de proyectos tradicionales (ciclo de vida en cascada, en espiral, etcétera) aparecen comparadas con XP, como metodologías pesadas y poco eficientes. La crítica más frecuente a estas metodologías “clásicas” es que son demasiado burocráticas.

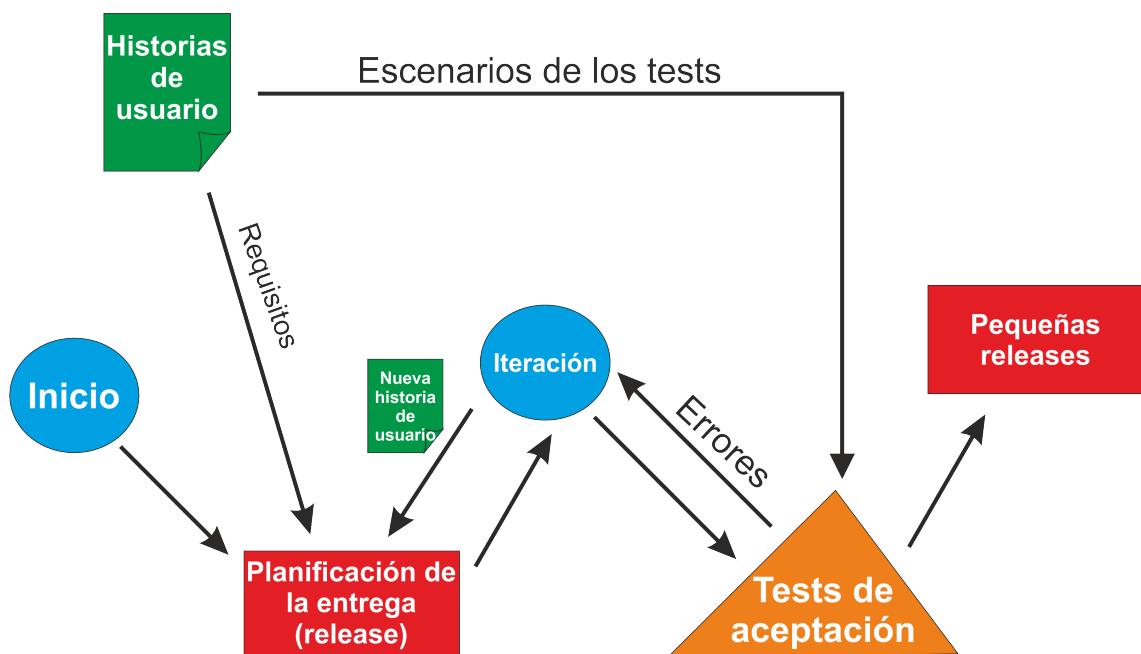


Figura 3.1: Ciclo de XP[38]

Al desarrollarse XP, se desarrollaron una serie de buenas prácticas entre las que destacan Test Driven Development (TDD), *pair programming*, testing continuo o la integración continua, ya que los equipos XP realizan numerosos *commits* varias veces por día. El beneficio de esta buena práctica de XP se observa en esos proyectos, donde la construcción se realiza semanalmente (o incluso mensualmente), y que, con frecuencia, terminan en un “infierno de integración”, donde el código desarrollado no funciona y ningún desarrollador conoce el motivo.

Uno de los principales motivos por los que ocurren estas situaciones es la integración infrecuente de código en el repositorio, es decir, la integración infrecuente consiste en no realizar, al menos diario, la integración de código fuente que posee el desarrollador en local al repositorio donde trabaja todo el equipo de desarrollo, realizar un *build* de ese código y ejecutar las pruebas necesarias para comprobar la integración de los módulos, artefactos y dependencias.

La integración infrecuente lleva a problemas serios en proyectos software[32]:

1. Aunque la integración continua es crítica para desarrollar un buen software, los equipos de desarrollo no la practican a menudo, delegando estas tareas a personas que no están familiarizadas con el sistema completo.
2. La integración infrecuente suele generar código con errores. En el momento de integrar, aparecen problemas que no se detectaron ni ocurrieron cuando se realizaron las pruebas al software sin integrar.
3. Los procesos de integración infrecuentes generan largos períodos de “congelación del código”. La congelación de código consiste en, que por largos períodos de tiempo, los programadores no pueden agregar nuevas características al software, puesto que no van a apreciarse los cambios realizados, dado que no están reparados los errores en el sistema, el sistema no está listo, dado que al realizar *commit* lo más normal es que surjan conflictos y no compile el código.

3.4 Integración continua

Dentro de la ingeniería del software, se encuentra la integración continua, desarrollada de la mano de Martín Fowler[12], integración continua es una práctica de desarrollo de software donde los miembros de un equipo integran su trabajo con frecuencia, en general, cada persona integra su trabajo, al menos una vez al día, lo que conlleva a múltiples integraciones por día. Cada integración se verifica mediante una compilación automatizada, incluidas las pruebas, con el objetivo de detectar errores de integración de manera rápida. Muchos equipos encuentran que este enfoque reduce significativamente los problemas de integración y permite desarrollar software de manera cohesiva y más rápida.

Por consiguiente, un entorno de integración continua permite la reducción de riesgos y la realización de tareas repetitivas, alcanzando un alto grado de automatización de los procesos involucrados en el desarrollo software, permitiendo la generación de software listo para desplegar.

La idea de integración continua nació de la mano de Martin Fowler tal y como se ha dicho antes, pero no fue el único, dado que parte de los autores del manifiesto ágil y otros ingenieros apoyaron la propuesta de Fowler:

- **Martin Fowler:** renombrado autor, consultor de software, orador y precursor de la integración continua, aporta dos décadas de experiencia ayudando a las empresas a utilizar tecnología orientada a objetos para sistemas de información críticos. Es uno de los autores del *Manifiesto para el Desarrollo de Software Ágil*, escritor de libros sobre desarrollo de software y orador de gran prestigio en conferencias internacionales.
- **Kent Beck:** ingeniero de software estadounidense[4], es uno de los creadores de las metodologías de desarrollo de software XP y TDD, incluyó la integración continua como una de las buenas prácticas dentro de XP.
- **Paul Duvall:** CTO en Stelligent. Experto en soluciones de entrega continua e integración continua en Amazon Web Service. Trabaja aplicando la integración continua con clientes como Sony Pictures Entertainment, Nutrisystem, Macy's, grandes compañías financieras, el gobierno de los Estados Unidos de América, Symantec, HP y Cardinal Health.
Autor del libro ganador del premio Jolt Product Excellence 2008 "Integración Continua: Mejorando la Calidad del Software y Reduciendo el Riesgo", orador en las principales conferencias de software sobre integración continua.

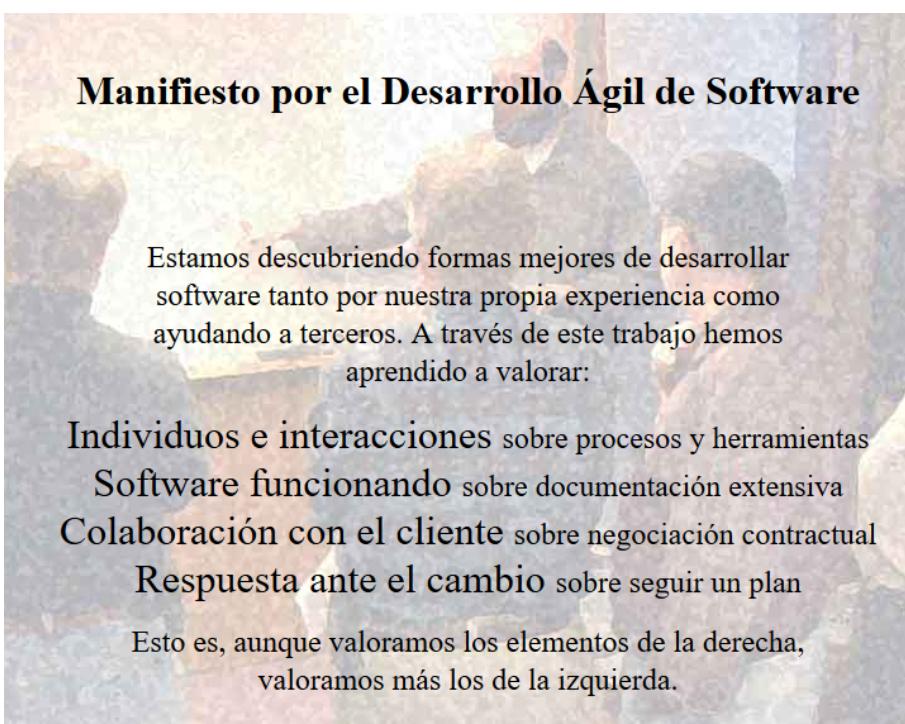


Figura 3.2: Algunos principios del *Manifiesto para el Desarrollo de Software Ágil* [5]

Todos ellos tienen en común que apoyaron el nacimiento y la evolución del desarrollo ágil, utilizando sus prácticas, entre las que destaca la integración continua.

3.4.1 Desarrollo ágil

A través del estudio e investigación, se deduce que una metodología ágil es una alternativa fiable a una metodología tradicional. Las metodologías ágiles desarrollan software de manera iterativa e incremental. Las metodologías ágiles se basan en *El Manifiesto Ágil*[5], documento donde se publicaron “los pilares” del desarrollo ágil, por ejemplo, “*Individuos e interacciones sobre procesos y herramientas*”, “*Software funcionando sobre documentación extensiva*”, “*Respuesta a un cambio sobre seguir un plan*”, etcétera.

A menudo, las experiencias reportadas, se refieren a la adopción de prácticas ágiles individuales o ciertos fundamentos de desarrollo ágil para complementar los procesos existentes de una organización, es decir, los equipos no toman todas las propuestas indicadas en las metodologías ágiles, sino que, toman e incorporan las que más les convienen para mejorar sus actuales formas y metodologías de trabajo.

Los equipos que utilizan SCRUM como metodología para gestionar sus equipos de trabajo, en casos puntuales, toman otras medidas para complementar sus metodologías, por ejemplo, en el caso del desarrollo de software incorporan integración continua. La integración continua no forma parte de la metodología de trabajo de SCRUM, pero sí en XP, por lo que se incorpora para ser utilizada en SCRUM.

Los equipos ágiles buscan con estas incorporaciones a sus metodologías de trabajo evitar los errores que cometen los equipos que trabajan con metodologías tradicionales, dado que no dictan con qué frecuencia se deben integrar todas las fuentes de un proyecto.

Esto supone que los miembros del equipo trabajen por separado durante horas, días e incluso semanas con un código fuente que contiene errores. Los equipos ágiles, debido a que generan código robusto y estable en cada iteración con alta frecuencia, manifiestan antes los errores que los equipos que trabajan con metodologías tradicionales. Por lo tanto, los equipos ágiles reducen el tiempo de sus desarrollos y de entrega de sus proyectos.

Para la realización de estas tareas (por ejemplo, reducción de tiempos en la entrega de productos) ha surgido un nuevo perfil, el cual permite reducir las diferencias entre los departamentos de desarrollo y operaciones, el perfil DevOps.

3.4.2 DevOps

DEVOPS consiste en un conjunto de prácticas que tratan de reducir las diferencias entre los departamentos de desarrollo y de operaciones, aunque su principal objetivo consiste en cubrir todos los aspectos que ayudan en la entrega de software rápida, optimizada y de alta calidad. DEVOPS es un conjunto de principios hacia la entrega de software donde el enfoque clave es la velocidad de entrega y las pruebas continuas, es decir, estar en estado de envío a producción en cualquier momento con retroalimentación continua, teniendo así la capacidad de reaccionar a cambios más rápidamente.

Los equipos que trabajan para lograr un DEVOPS heredan de los principios ágiles. Aunque los principios de DEVOPS se aplican a todo el ciclo de vida del proyecto, la zona de motivación clave o zona de enfoque que desencadenó todo esto, es asegurarse de que el equipo de operaciones pueda ejecutar a lo largo de todo el ciclo de vida del desarrollo con equipos de desarrollo.

DEVOPS también tiene en cuenta que se desarrolle un software de calidad, siguiendo los principios de XP, es decir, no se basa sólo en desarrollar un software que esté disponible para desplegar en cualquier momento o en que se realicen muchos *commits* a diario, se centra en que el software desarrollado sea de calidad.

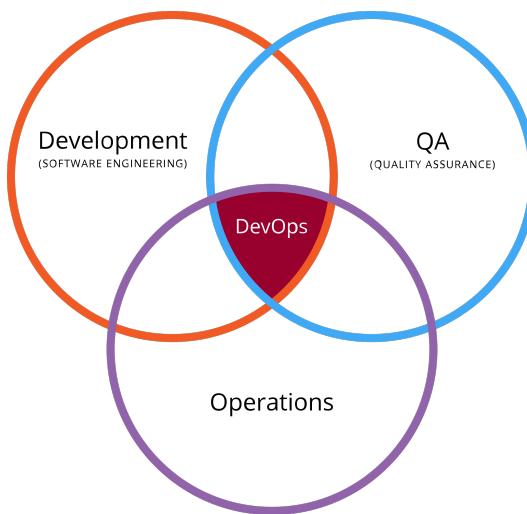


Figura 3.3: Perfil DEVOPS

Por lo que DEVOPS es utilizado para conseguir realizar una entrega continua rápida y de calidad al cliente, para ello, utiliza la integración continua para realizar integraciones frecuentes procurando así obtener siempre un código robusto y estable, tras comprobar que se realiza de manera correcta el *build* del proyecto y se superan las pruebas.

Para conseguir obtener una entrega continua de calidad, tal y como se ha dicho antes, se utiliza la integración continua, pero la integración continua contiene una serie de características y requisitos para que sea realizada de manera correcta.

3.5 Características y requisitos de la integración continua

La integración continua consta de una serie de propuestas (o buenas prácticas) a seguir para que se desarrolle de la mejor manera posible. Entre todas estas propuestas de buenas prácticas destaca las nombradas por Martin Fowler[12], precursor de la integración continua.

1. Mantener un único repositorio fuente.

Los desarrollos de software trabajan sobre archivos que necesitan orquestarse entre sí para realizar el *build* de un producto.

Rastrearlos supone un esfuerzo, y más, si cada desarrollador tiene fragmentos de código en su equipo local, en distintas carpetas o lugares de almacenamiento lo cual es un error.

Cuando el proyecto es desarrollado por un equipo de desarrollo formado por varias personas en vez de por una sola, esta situación es inaceptable.

Con lo cual, no supone una sorpresa que con el paso del tiempo los equipos de desarrollo de software hayan construido herramientas que solucionen estos problemas, estas herramientas son conocidas como sistemas de control de versiones, las cuales, en poco tiempo, se han convertido en una parte imprescindible en la gestión de los proyectos de desarrollo de software.



Figura 3.4: Repositorios y sistemas de control de versiones

Otro error común en equipos que sí utilizan repositorios, consiste en que no ponen todo dentro de él, es decir, contienen parte del código en local y parte del código en repositorios. Es decir, se debe colocar todo lo que sea necesario para hacer un *build* en el repositorio, incluyendo: *scripts* de prueba, archivos de propiedades, esquemas de bases de datos, *scripts* de instalación o bibliotecas de terceros.

2. Automatizar el *build*.

Convertir el código fuente en sistemas de ejecución puede ser un proceso complicado que involucra, entre otros muchos pasos, compilación de código, mover archivos o cargar esquemas dentro de las bases de datos[17].

Sin embargo, como la mayoría de las tareas en esta parte del desarrollo de software, se puede y se debe automatizar. Ya que realizar estas tareas mediante comandos y de manera manual no es nada más que una pérdida de tiempo y, además, supone una alta probabilidad de cometer errores.

Los entornos automatizados para *builds* son una herramienta común de los sistemas. El mundo UNIX los ha hecho por décadas; la comunidad Java desarrolló Ant, Maven y Gradle, la comunidad .NET desarrolló NAnt y ahora dispone de MSBuild.

Un error típico al desarrollar consiste en no incluir todo en el *build* automatizado. El *build* debe poder obtener del repositorio el esquema de la base de datos y ponerlo en marcha en el entorno de ejecución.

Los *build scripts* se obtienen de diferentes formas y a menudo son particulares para una plataforma o comunidad. Hacer un gran *build* en los *script* lleva tiempo, y no se pretenden realizar todos esos pasos sólo para hacer un pequeño cambio; entonces, una buena herramienta para hacer un *build* analiza lo que necesita cambiar como parte del proceso. La forma más común de hacer esto es comprobar las fechas de la fuente y archivos de objetos y sólo compilar si la fecha de la fuente es posterior.

Dependiendo de las necesidades, puede resultar adecuado fragmentar la construcción del software en diferentes artefactos o partes del proyecto. Algunos componentes se pueden construir de forma autónoma. Un *build script* debe permitir hacer un *build* en destinos alternativos para casos diferentes.

3. Crear *builds* testeables.

Tradicionalmente, *build* significa compilar y el resto de pasos que se necesiten para lanzar un programa. Un programa puede ejecutarse, pero eso no significa que lo haga de forma correcta. Los lenguajes modernos, en especial los tipados de manera estática, pueden atrapar muchos *bugs*, pero otros *bugs* permanecen ocultos.

Una forma de encontrar *bugs* de manera más rápida y eficiente es incluir pruebas automatizadas en el proceso de *build*. Estas pruebas no son perfectas al 100 %, pero pueden descubrir muchos *bugs*; por lo que son útiles.

Un código autotestable es aquel que tiene un número representativo de pruebas que permiten comprobar una gran parte del código. Las pruebas deben poder dispararse desde un comando simple y autocomprobarse. El resultado de la ejecución del conjunto de pruebas debe indicar si alguna prueba ha fallado.

Por otra parte, se afirma que mediante estas pruebas no se descubrirán todos los *bugs*. Por lo que, estas pruebas no implican la ausencia de *bugs* en el código fuente.

Sin embargo, la perfección a la hora de realizar las pruebas no es el único punto que justifica la realización de un *build* que sea capaz de testearse.

Tal y como dice Martin Fowler[12] de manera coloquial: “*La ejecución de pruebas que no manifiestan todos los errores de un código con frecuencia, son mucho mejores que la ejecución de aquellas pruebas perfectas que nunca se escribieron*”.

4. Todos hacen *commit* al *mainline* todos los días.

La integración continua es, ante todo, comunicación. La integración continua permite a los desarrolladores contar al resto de desarrolladores los cambios que han realizado. La comunicación continua permite que el resto del equipo conozca los cambios que se han realizado.

El único prerequisito para un desarrollador que hace *commit* al *mainline* es que el resto de desarrolladores puedan hacer un *build* correcto del código. Esto, por supuesto, incluye superar de manera correcta la pruebas que contenga el *build*.

Al realizar el ciclo completo para hacer *commit* de manera frecuente, los desarrolladores descubren rápidamente si existe algún conflicto entre el código desarrollado por dos o más desarrolladores distintos. La clave para solucionar problemas de forma rápida es encontrar dicho problema de manera rápida. Con desarrolladores que hacen *commit* cada pocas horas, el conflicto se puede detectar dentro de las primeras horas, es decir, el error no ha tenido tiempo de propagarse por lo que, de manera rápida, se soluciona. Los conflictos y errores que no son detectados rápidamente son difíciles de resolver cuando se descubren.

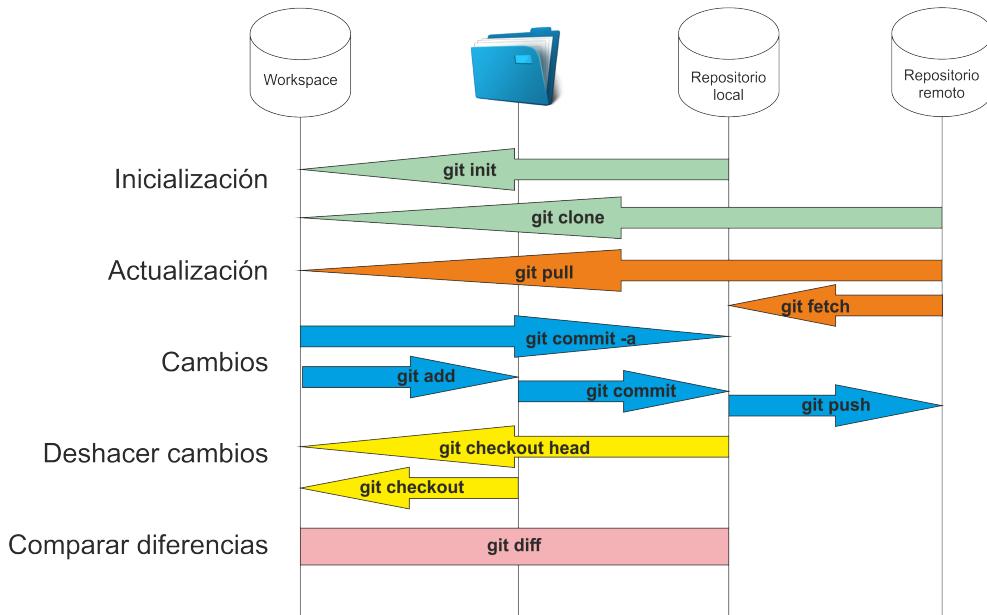


Figura 3.5: Algunas acciones de Git[13].

El hecho de realizar un *build* cuando se actualiza la copia de trabajo local significa que se han detectado conflictos en la compilación. Como el *build* tiene la capacidad de testearse asimismo, también detecta conflictos en la ejecución del código.

Como sólo hay un par de horas de cambios entre *commits*, no hay muchos lugares donde pueda estar escondido el problema, por lo que se encuentra el error y se soluciona de manera rápida. Además, el proyecto que se está desarrollando no ha cambiado mucho dado que ha transcurrido poco tiempo, por lo que se puede *debuggear* para ayudar a encontrar el error.

5. Hacer *commit* debe construir el *mainline* en una máquina de integración.

Haciendo *commits* a diario, se obtienen *builds* probados con mucha frecuencia. Por lo que el *mainline* permanece en estado saludable. Sin embargo, aún el desarrollo no es perfecto y siguen surgiendo errores. Una razón es la disciplina, el desarrollador debe actualizar y hacer un *build* antes de hacer *commit*, para comprobar que el proceso de hacer *commit* se ha realizado de manera correcta hasta el final.

Como resultado se debe asegurar que los *builds* se realicen en una máquina de integración y sólo si este *build* de integración se realiza con éxito se debe hacer *commit* al *mainline*. Se debe monitorizar el *build* del *mainline* para poder repararlo en caso de que se rompa.

Hay dos maneras para asegurar que se ha realizado este proceso de manera correcta: utilizar un *build* manual o un servidor de integración continua.

Utilizar un *build* manual es simple. Esencialmente, consiste en un *build* que se realiza en el repositorio antes que el *commit*.

Un servidor de integración continua actúa como un monitor del repositorio. Cada vez que finaliza un *commit* contra el repositorio, el servidor automáticamente realiza un *checkout* de las fuentes en la máquina de integración, inicia un *build*, y notifica los resultados obtenidos del *build* al desarrollador que ha realizado el *commit*, el *build* no termina hasta que se obtiene la notificación.

Muchas organizaciones realizan *builds* regulares en un cronograma programado, por ejemplo, todas las noches. Pero no se obtienen los mismos resultados, puesto que no es igual que un *build* continuo y no es suficiente para ser considerado como integración continua.

La finalidad de la integración continua es encontrar problemas tan pronto como sea posible. Un *build* nocturno significa que los *bugs* estuvieron alojados durante todo el día en el código sin ser detectados. Una vez que están en el sistema por tanto tiempo, lleva mucho tiempo encontrarlos y solucionarlos.

6. Mantener rápido el build.

El objetivo de la integración continua es proporcionar un *feedback* rápido.

Para aquellos proyectos, según XP, en los que su *build* tarde un máximo de 10 minutos en realizar todo sus pasos, son considerados *builds* con un tiempo óptimo de construcción, actualmente, la mayoría de proyectos logran no superar ese tiempo.

Se considera de gran valor para un proyecto realizar un esfuerzo para lograr reducir el tiempo de construcción del *build*, puesto que cada minuto que se reduzca en la realización del *build* es un minuto más para cada desarrollador cada vez que haga *commit*. Puesto que la integración continua demanda hacer *commits* con alta frecuencia esto implica ahorrar una gran cantidad de tiempo.

Por lo tanto, realizar un *build* que se ejecute en poco tiempo supone reducir el tiempo en el cual un proyecto finalice.

7. Prueba en un clon del entorno de producción.

Cuando se realizan pruebas se persigue un objetivo, que consiste en eliminar, bajo condiciones controladas, cualquier problema que surja durante la ejecución del *build*.

Para conseguir este objetivo, obtiene un gran peso el entorno dentro del cual el *build* se ejecutará. Como resultado, se pretende configurar el entorno de prueba para que sea lo más parecido a un clon del entorno de producción donde será ejecutado ese *build*, utilizando el mismo software de base de datos (incluyendo las mismas versiones), la misma versión de sistema operativo, colocando todas las bibliotecas o recursos que sean necesarios donde se encontrarían en el entorno de producción dentro del entorno de prueba, aunque no sean necesarias también se recomienda utilizar las mismas direcciones IP, en los mismos puertos y ejecutándose en el mismo hardware.

Cualquier desarrollador de software, conoce que no es posible probar en un clon de cada una de las posibles computadoras con todo el software de terceros que diferentes personas estén ejecutando. Ocurre lo mismo con algunos entornos de producción que pueden ser muy costosos de duplicar. Más allá de estos límites, el objetivo debe seguir siendo duplicar el entorno de producción tanto como sea posible, y entender los riesgos que aceptas para cada diferencia entre la prueba y la producción.

8. Facilitar la obtención del último archivo ejecutable.

Desarrollar software no es sencillo, una de las partes más difíciles del desarrollo de software consiste en asegurar que se está realizando un *build* correcto.

Cualquier desarrollador involucrado en un proyecto de software debería poder obtener el archivo ejecutable más reciente y poder ejecutarlo: para demostraciones, pruebas, u observar qué ha cambiado en el desarrollo sin tener que realizar ningún cambio en su máquina o en el *build*.

9. Todo el equipo debe observar los cambios que están sucediendo.

La integración continua, como ya se ha dicho en varias ocasiones, es comunicación, así que se debe asegurar de que todos los miembros del equipo puedan ver fácilmente el estado del sistema y los cambios que se han realizado en él[10].

Si se utiliza la integración continua manual, la visibilidad sigue siendo esencial. El monitor de la máquina física del *build* debe mostrar el estado del *build* del *mainline*.

Por ello, surge la ventaja de utilizar un sitio web, ya que así, aquellos que no están en el mismo lugar, pueden conocer el estado del proyecto accediendo a una página web.

10. Automatizar el *deployment*.

Para ser capaz de realizar una integración continua de manera correcta se necesitan múltiples entornos: un entorno para ejecutar pruebas de *commit*, uno o más entornos para ejecutar pruebas secundarias, etcétera.

Además, si se necesitan mover archivos ejecutables entre los diferentes entornos varias veces al día y de manera repetitiva, no se puede realizar de manera manual dado que es ineficiente y supone pérdidas de tiempo, por lo que se hace necesario automatizar el proceso.

En consecuencia, también se necesitan *scripts* que permitan realizar el *deployment* en el entorno de producción con facilidad. No se realiza el *deployment* de una producción cada día, pero el *deployment* automático permite acelerar el proceso y reducir la capacidad de cometer errores.

3.6 Herramientas

3.6.1 Bamboo

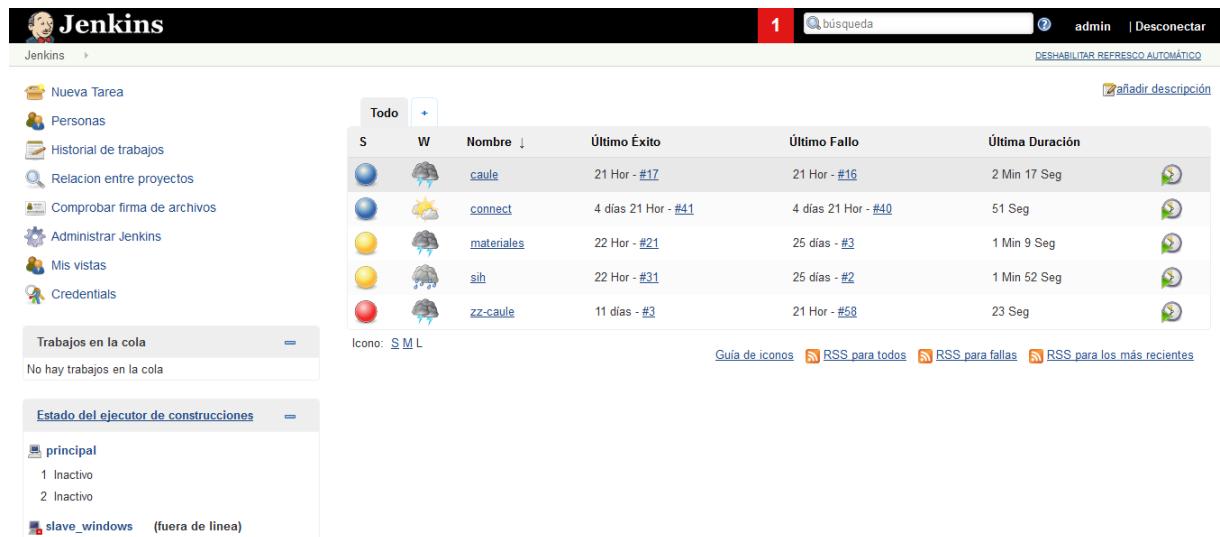
Bamboo es una herramienta contenida en un contenedor web, posee licencia de propietario, tiene compatibilidad con varios *builders* de Windows como MSBuild, NAnt y Visual Studio, además, también cuenta con *builders* de Java, tales como Ant y Maven, permite el envío de notificaciones vía email, Google, XMPP y RSS. Integrado con varios Integrated Development Environment (IDE) como IntelliJ IDEA, Eclipse y Visual Studio. Destaca también su integración con Bitbucket y GitHub.

3.6.2 CircleCI

CircleCI[7] es un software de integración continua que está contenido en un *hosted*, con licencia “*Creative Commons*”, tiene compatibilidad con Python, Ruby y Java entre otros, se encuentra disponible para Ubuntu y macOS. Se integra con repositorios como GitHub y Bitbucket o con herramientas como Azure y Docker, permite enviar notificaciones mediante Jira y Slack.

3.6.3 Jenkins

Al igual que Bamboo esta herramienta está contenida en un contenedor web, sin embargo, es una herramienta de software libre ya que cuenta con licencias de “*Creative Commons*” y “*MIT*”, contiene MSBuild y NAnt, *builders* de Windows y Ant, Maven y Kundo, *builders* de Java, Ruby, Python y shell script. Permite enviar notificaciones vía Twitter, Google, email, Android o Slack. Posee compatibilidad con varios IDE, Eclipse, IntelliJ IDEA y NetBeans, además también cuenta con la integración de Google Code, Jira, Bitbucket, GitHub, GitLab y Redmine entre otras muchas.



The screenshot shows the Jenkins dashboard. At the top, there's a navigation bar with links for 'Nueva Tarea', 'Personas', 'Historial de trabajos', 'Relacion entre proyectos', 'Comprobar firma de archivos', 'Administrar Jenkins', 'Mis vistas', and 'Credentials'. Below the navigation bar is a search bar and user information ('admin | Desconectar'). A red notification badge with the number '1' is visible. On the left, there are two expandable sections: 'Trabajos en la cola' (No hay trabajos en la cola) and 'Estado del ejecutor de construcciones' (principal: 1 Inactivo, 2 Inactivo; slave_windows: fuera de linea). The main area displays a table of projects with columns: S (Status), W (Weather icon), Nombre (Name), Último Éxito (Last Success), Último Fallo (Last Failure), and Última Duración (Last Duration). The table lists five projects: 'caule' (status green, last success 21 Hor - #17, last failure 21 Hor - #16, duration 2 Min 17 Seg), 'connect' (status green, last success 4 días 21 Hor - #41, last failure 4 días 21 Hor - #40, duration 51 Seg), 'materiales' (status yellow, last success 22 Hor - #21, last failure 25 días - #3, duration 1 Min 9 Seg), 'sih' (status yellow, last success 22 Hor - #31, last failure 25 días - #2, duration 1 Min 52 Seg), and 'zz-caule' (status red, last success 11 días - #3, last failure 21 Hor - #68, duration 23 Seg). Below the table are links for 'Guia de iconos', 'RSS para todos', 'RSS para fallas', and 'RSS para los más recientes'. There are also icons for 'añadir descripción' and 'Deshabilitar REFRESCO AUTOMÁTICO'.

Figura 3.6: Panel principal de Jenkins

3.6.4 Solano CI

Esta herramienta se encuentra albergada en un *hosted*, en la nube privada o sobre software multiplataforma, posee licencia de propietario, no cuenta con *builders* para Windows, posee *builders* de Java para Ant, Andriod, Maven y Gradle y otros *builders* para C, C++, JavaScript, PHP o Ruby. Cuenta con notificaciones vía email, pero no cuenta con integración para IDE, sin embargo, sí se encuentra integrada con GitHub o Bitbucket.

3.6.5 TeamCity

Al igual que Bamboo y Jenkins está contenida en un contenedor web, posee licencia de propietario, cuenta con *builders* de Windows tales como MSBuild, NAnt y VisualStudio, *builders* de Java tales como Ant, Maven y Gradle, notificaciones vía email, XMPP y RSS, compatible con IDE como Eclipse, Visual Studio o IntelliJ IDEA.

3.6.6 Travis CI

Travis CI se alberga en un *hosted*, su licencia es "*MIT*", no posee *builders* para Windows pero sí para Java, donde destacan Ant, Maven y Gradle, además, soporta *builders* para C, C++, Go, Groovy, Python, Ruby o PHP. Permite enviar notificaciones vía email o Slack entre otros medios, pero no posee integración con IDE, pero sí con GitHub.



Figura 3.7: Herramientas de integración continua

3.6.7 Comparativa

Herramienta	Tipo de licencia
Bamboo	Propietario
CircleCI	<i>Creative Commons</i>
Jenkins	<i>Creative Commons</i> y <i>MIT</i>
Solano CI	Propietario
TeamCity	Propietario
Travis CI	<i>MIT</i>

Cuadro 3.1: Comparativa de tipo de licencia

Herramienta	<i>Hosting</i>
Bamboo	Contenedor web
CircleCI	<i>Hosted</i>
Jenkins	Contenedor web
Solano CI	Contenedor web
TeamCity	Contenedor web
Travis CI	<i>Hosted</i>

Cuadro 3.2: Comparativa de *hosting*

Herramienta	Notificaciones
Bamboo	Email, Google, XMPP y RSS
CircleCI	Jira y Slack
Jenkins	Twitter, Google, email, Android y Slack
Solano CI	Email
TeamCity	Email, XMPP y RSS
Travis CI	Email y Slack

Cuadro 3.3: Comparativa de envío de notificaciones

Herramienta	<i>Builders</i>
Bamboo	MSBuild, NAnt, Vistual Studio, Ant y Maven
CircleCI	Python, Ruby, Ant y Maven
Jenkins	MSBuild, NAnt, Ant, Maven, Gradle, Android, Kundo, Ruby, Python y Shell script
Solano CI	Ant, Android, Maven, Gradle, C, C++, JavaScript, PHP y Ruby
TeamCity	MSBuild, NAnt, Visual Studio, Ant, Maven y Gradle
Travis CI	Ant, Maven, Gradle, C, C++, Go, Groovy, Python y Ruby

Cuadro 3.4: Comparativa de *builders*

Herramienta	Otras integraciones
Bamboo	IntelliJ IDEA, Eclipse, Visual Studio, Bitbucket y GitHub
CircleCI	Azure y Docker
Jenkins	Eclipse, IntelliJ IDEA, NetBeans, Google Code, Jira, Bitbucket, GitHub, GitLab y Redmine
Solano CI	GitHub y Bitbucket
TeamCity	Eclipse, Visual Studio y IntelliJ IDEA
Travis CI	GitHub

Cuadro 3.5: Comparativa de integraciones extras

Capítulo 4

Método de trabajo

A Lo largo de este capítulo se explica la forma en que se ha llevado a cabo este trabajo. Para ello, se justifica la metodología utilizada y se explican variaciones respecto a ésta.

En consecuencia, también se explica el desarrollo desde el punto de vista software y hardware, nombrando las principales tecnologías utilizadas.

4.1 Introducción

En este proyecto no es posible utilizar metodologías de gestión de proyectos como proceso unificado de desarrollo o SCRUM[25], dado que este proyecto no versa sobre el desarrollo de un producto, ni tampoco sobre el desarrollo de un software y que, en cualquier caso, será realizado por una persona.

En su lugar, se procederá a desarrollar iterativamente utilizando una heurística, en concreto, la heurística IDEAL, y a partir de los resultados obtenidos en la última fase de cada iteración, se procede a incrementar el prototipo obtenido.

4.2 Heurística IDEAL

IDEAL fue formulada por Bransford y Stein en 1984[6], incluye 5 pasos de los cuales surgen el nombre de esta metodología, **I**dentificar el problema, **D**efinir y presentar el problema, **E**xplorar las estrategias viables, **A**vanzar en las estrategias y **L**ograr la solución.

La heurística IDEAL se define como iterativa e incremental, dado que repite varias veces el proceso hasta que el problema ha sido resuelto por completo, e incremental puesto que cada iteración y nuevo inicio por los pasos del ciclo va incrementando las partes del proyecto a evaluar.

Para su estudio y aplicación, realmente, en vez de en cinco fases se suelen agrupar las dos primeras en una única fase[29], “*Identificar el problema*” y “*Definir y presentar el problema*” tratan aspectos muy parecidos por lo que para evitar redundancias a la hora de realizar tareas se agrupan en una única fase.

4.2.1 Identificación del problema

En esta primera fase del ciclo a realizar, se procede a observar y analizar el problema y se determina el resultado que se debe obtener. Para esta primera fase de la heurística IDEAL, se toma como entrada la última salida del ciclo, en este caso, el prototipo obtenido en la última fase.



Figura 4.1: Ciclo IDEAL

4.2.2 Realización del plan

Se determinan los pasos y operaciones a realizar para resolver el problema planteado. Por lo general, se descompone el problema en subproblemas más pequeños para facilitar la manera de resolver dicho problema. Para el desarrollo de este proyecto, tomaremos los test que debe superar la aplicación para comprobar si se han conseguido los objetivos marcados y se analizará la manera mediante la cual se deben superar dichos tests, o si se deben provocar los fallos de los tests en busca de errores.

4.2.3 Ejecución del plan

Se procede a la ejecución de cada paso del plan, siguiendo la especificación de pasos a seguir en la fase anterior “*Realización del plan*”, tras realizarse la descomposición en subproblemas más pequeños.

4.2.4 Analizar la solución

Una vez resuelto el problema identificado en la primera fase, se confirma la nueva versión del prototipo del entorno de integración continua, además, se toman técnicas de SCRUM y se realizan reuniones al final de cada iteración con MADRIJA y el equipo de desarrollo para ver la evolución del prototipo, se inicia de nuevo el ciclo IDEAL desde la primera fase, tomando como entrada la salida de esta última fase, es decir, la última versión del prototipo.

4.3 Aplicación del método de trabajo

A continuación, se presenta cómo ha sido aplicada la metodología heurística IDEAL para el desarrollo de este proyecto. Cabe destacar que lo presentando en este apartado es sólo la planificación inicial de este proyecto, por lo que más adelante, en el capítulo 5, se desglosarán los pasos seguidos.

Para la aplicación de la metodología de trabajo se va a utilizar OpenProject[26] como software para gestionar el desarrollo de este proyecto.

Dentro de OpenProject, cada iteración de este desarrollo se corresponde con una versión, es decir, por cada iteración durante el desarrollo de este proyecto se realizará una nueva versión del objetivo final que se persigue. Mediante la visualización de cada versión podemos observar el número de tareas que se llevan a cabo y el estado en el que se encuentran.

The screenshot shows the OpenProject interface for 'Iteración 1'. On the left is a sidebar with links: Paquetes de trabajo (selected), Líneas de tiempo, Backlogs, Calendario, Wiki, Reportes de costo, Miembros, Presupuestos, and Configuración del proyecto. The main content area has a header 'Iteración 1' and a sub-header 'Objetivo'. It contains text about the goal of obtaining a prototype and implementing a continuous integration system. Below is a section 'Fases y tareas' with a note that tasks are published on the wiki. A list of tasks follows:

- [Task #718 Closed](#): Identificación del problema - Iteración 1. 2017-03-13 – 2017-03-13
- [Task #719 Closed](#): Realización del plan - Iteración 1. 2017-03-13 – 2017-03-13
- [Task #727 Closed](#): Paso 1 del plan. 2017-03-14 – 2017-03-14
- [Task #728 Closed](#): Paso 2 del plan. 2017-03-14 – 2017-03-15
- [Task #729 Closed](#): Paso 3 del plan. 2017-03-15 – 2017-03-22
- [Task #731 Closed](#): Paso 4 del plan. 2017-03-22 – 2017-03-30
- [Task #721 Closed](#): Análisis de la solución - Iteración 1. 2017-03-31 – 2017-04-03

Below is a section 'Errores y cosas a tener en cuenta' with a note about a conflict between libraries and a bug fix:

- No contabiliza los tests, se ejecutan correctamente pero no los contabiliza. [Error solucionado](#) (conflicto entre librerías), solucionado en la tarea [Bug #1004 Closed](#): Errores Selenium y Cucumber 2017-04-19 – 2017-04-20

Figura 4.2: Imágenes de OpenProject de la iteración 1

A su vez, en las versiones de OpenProject, se desglosa en tareas el objetivo que se pretende conseguir en cada iteración, también se anotan los errores surgidos durante el desarrollo de la iteración para no cometerlos en futuras iteraciones.

Además, OpenProject consta de una wiki donde se indican las tareas que se van a realizar en cada versión, el estado de las mismas o algunos comentarios sobre, por ejemplo, errores que se han cometido para que no se realicen en el futuro.

Por lo cual, para la primera fase de la metodología heurística IDEAL, la fase de “Identificación del problema”, se crea una nueva versión del proyecto con su correspondiente wiki, en dicha wiki se indica el propósito de esa versión y se crea una tarea de tipo “Meeting”, la cual indica que se va a realizar una reunión con MADRIJA para tratar los problemas que se desean solucionar y para obtener los requisitos de esta nueva versión.

VERSIÓN	FECHA DE INICIO	FECHA DE VENCIMIENTO	DESCRIPCIÓN	ESTADO	COMPARTIENDO	PÁGINA WIKI
Iteración 1	13/03/2017	03/04/2017	Implantación de un entorno de integración continua en zz-caule.	abrir	No compartido	Wiki_Iteración_1
Iteración 2	03/04/2017	17/04/2017	Definición de la arquitectura para el sistema de integración continua.	abrir	No compartido	Wiki_Iteración_2
Iteración 3	24/04/2017	08/05/2017	Determinar una estrategia para definir cuando debe actuar o no el sistema de integración continua dentro de Madrid Consultoría, S.L.	abrir	No compartido	wiki_iteración_3

Figura 4.3: Algunas versiones del proyecto en OpenProject

Dado que en esta primera fase se debe realizar la captura de los requisitos, se ha optado por utilizar varias técnicas de captura de requisitos para conseguir obtener, con la mayor firmeza posible, los requisitos que se deben cumplir en cada versión de cada iteración. Se han utilizado las siguientes técnicas:

- **Entrevistas[36]:** técnica que consiste en mantener una o más reuniones entre dos o más personas, en las que se plantean una serie de preguntas para obtener las correspondientes respuestas en el contexto de un determinado dominio de problemas.
- **Cuestionarios[24]:** esta técnica consiste en la elaboración de preguntas con el propósito de obtener información de los usuarios encuestados.
- **Brainstorming:** es una técnica basada en reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios.

Posteriormente, en la segunda fase del de la heurística IDEAL, se realiza otra tarea en OpenProject, en la cual se describe el plan a seguir para cumplir el objetivo marcado en el paso anterior.

Una vez finalizada la fase de “*Realización del plan*”, se crean las tareas correspondientes a cada paso a seguir del plan elaborado, no hay un número fijo de tareas, dado que cada plan de cada versión puede tener un número distinto de pasos a seguir.

Iteración 1

Fecha de inicio 13/03/2017 Fecha de vencimiento 03/04/2017

Implantación de un entorno de integración continua en zz-caule.

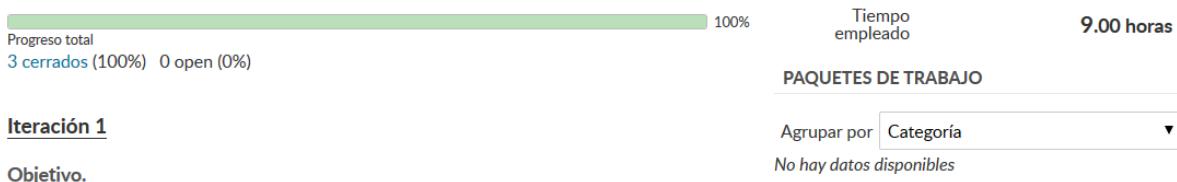


Figura 4.4: Ejemplo de la información que se muestra en la wiki de la iteración 1 (Parte 1)

Por último, se realiza una tarea llamada “Análisis de la solución”, en la cual tras realizar todos los pasos del plan, se evalúan los resultados obtenidos así como los errores encontrados (tanto los resueltos, como los no solucionados) y se cierra esta versión.

Fases y tareas.

Por cada fase de la iteración se realizan una o varias tareas, el resultado obtenido se publicará en esta wiki.

Las tareas a ejecutar, por orden, son las siguientes (algunas tareas pueden realizarse en paralelo):

- [Task #718 Closed](#): Identificación del problema - Iteración 1. 2017-03-13 – 2017-03-13
- [Task #719 Closed](#): Realización del plan - Iteración 1. 2017-03-13 – 2017-03-13
- [Task #727 Closed](#): Paso 1 del plan. 2017-03-14 – 2017-03-14
- [Task #728 Closed](#): Paso 2 del plan. 2017-03-14 – 2017-03-15
- [Task #729 Closed](#): Paso 3 del plan. 2017-03-15 – 2017-03-22
- [Task #731 Closed](#): Paso 4 del plan. 2017-03-22 – 2017-03-30
- [Task #721 Closed](#): Análisis de la solución - Iteración 1. 2017-03-31 – 2017-04-03

Errores y notas a tener cuenta.

- No contabiliza los tests, se ejecutan correctamente pero no los contabiliza. [Error solucionado](#) (conflicto entre librerías), solucionado en la tarea [Bug #1004 Closed](#): Errores Selenium y Cucumber 2017-04-19 – 2017-04-20
- Indica un error al enviar e-mails pero los e-mails se reciben correctamente. [Error solucionado](#), simplemente estaba el puerto cerrado.
- Se debe conocer la ruta donde se encuentra la instalación de Git (incluyendo en la ruta hasta git.exe) en cada esclavo de Jenkins.
- Necesidad de conocer la ruta de MAVEN_HOME y JAVA_HOME en cada esclavo de Jenkins.
- Necesidad de disponer en el equipo de trabajo esclavo de los proyectos padres, si fueran necesarios para la compilación de algún proyecto debido a que Jenkins no descarga desde el repositorio del proyecto hijo los proyectos padres.
- En equipos Windows, se necesita tener instalado SSH en su cmd, si ya estuviese SSH instalado por tener MSYS o similares en el equipo, se debe indicar la ruta de Git en \$PATH.

Figura 4.5: Ejemplo de la información que se muestra en la wiki de la iteración 1 (Parte 2)

4.4 Planificación de las iteraciones

Se estima que para la realización de este trabajo serán necesarias 5 iteraciones, la duración de cada iteración será de dos semanas.

Iteración	Propósito
Iteración 1.	En esta primera iteración se pretende desarrollar una prueba de concepto como prototipo del sistema de integración continua.
Iteración 2.	En esta segunda iteración se pretende obtener como resultado un prototipo de una infraestructura y arquitectura para el entorno de integración continua.
Iteración 3.	En la tercera iteración se pretende obtener una estrategia que active las ejecuciones del entorno de integración continua, permitiendo así obtener el mayor rendimiento y beneficio posible del mismo.
Iteración 4.	En la cuarta iteración se pretende acoplar al sistema de integración continua el software "Renombrator" que permite encontrar palabras claves que no pueden ser utilizadas para nombrar columnas en las bases de datos y cambiarlas por nombres válidos.
Iteración 5.	En la quinta iteración se pretende añadir SonarQube al sistema de integración continua para que mida la calidad del código de MADRIJA.

Cuadro 4.1: Iteraciones

4.5 Marco tecnológico

4.5.1 Medios software

Debido a que este trabajo se realiza mediante un convenio FORTE, las herramientas a utilizar son impuestas por la empresa.

■ Sistemas operativos

- **Windows 10 Home x64:** Estación de trabajo.
- **Ubuntu Server x64:** Servidor de la empresa.

■ Herramientas y tecnologías para el desarrollo software

- **Jenkins:** servidor de integración continua de código abierto[34].
- **Java:** lenguaje de programación y una plataforma informática comercializada por Sun Microsystems[33].
- **JUnit:** conjunto de bibliotecas creadas por Erich Gamma y Kent Beck que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java[2].
- **Maven:** es una herramienta que se utiliza para construir y gestionar cualquier proyecto, y sus dependencias, basado en Java.
- **Eclipse:** IDE, es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar software[11].
- **Selenium:** entorno de pruebas de software para aplicaciones basadas en la web[31].
- **Cucumber:** herramienta de automatización de pruebas[8] para Behavior Driven Development (BDD).



Figura 4.6: Algunas tecnologías utilizadas en este desarrollo

■ Control de versiones

- **Git:** sistema de control de versiones de código abierto[37].
- **GitLab:** plataforma de desarrollo colaborativo para el alojamiento de proyectos que utilizan el sistema de control de versiones Git[14].

■ Modelado y documentación

- **LATeX:** es un sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica[21].
- **Inkscape:** software para edición de imágenes basado en gráficos vectoriales[35].

■ Herramientas para la virtualización

- **Oracle VM VirtualBox:** Oracle VM VirtualBox es un software de virtualización para arquitecturas x86/amd64[18].
- **Vagrant:** herramienta para la creación y configuración de entornos de desarrollo virtualizados[15].

■ Herramientas de bases de datos

- **HSQLDB:** Hyperthreaded Structured Query Language DataBase, es un sistema gestor de bases de datos, libre escrito en Java[16].
- **Liquibase:** es una biblioteca independiente de bases de datos de código abierto para el seguimiento, la gestión y la aplicación de los cambios de esquema de base de datos[22].
- **DBeaver:** herramienta para la gestión de bases de datos[9].

■ Herramientas para la gestión del proyecto

- **OpenProject:** software dedicado a la gestión de proyectos[26].
- **Microsoft Project:** software dedicado a la gestión de proyectos, en especial, para diagramas de Gantt[23].

■ Herramientas para la comunicación entre el servidor y los clientes

- **Java Web Start:** permite iniciar aplicaciones Java que se encuentran en un servidor web de aplicaciones, pero antes, comprueba que el cliente posee la última versión de la aplicación, si no la tiene, se debe actualizar la versión del cliente e inicia la comunicación. El inicio de las aplicaciones se realiza mediante enlaces a una página web o a través de enlaces de escritorio desde el cliente[28].
- **SSH:** herramienta para administración remota o acceso a equipos privados a través de una puerta trasera, denominada *backend*, permite manejar un equipo por completo a través de la línea de comandos[27].

4.5.2 Medios hardware

- **PC:** Ordenador portátil HP NOTEBOOK 15-r213ns con las siguientes características hardware:
 - Procesador Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz.
 - 8 GB RAM.
 - HDD 1 TB.
- **Servidor:** HP ProLiant DL360 G7 con las siguientes características:
 - 12 CPUs x Intel(R) Xeon(R) CPU X5650 @ 2.67 GHz.
 - 32 GHz de CPU.
 - 16 GB RAM.
 - HDD 400 GB.
 - VMWare ESXi versión 6.5.0.

Capítulo 5

Resultados

El objetivo de este capítulo es presentar de manera detallada los resultados obtenidos tras aplicar la metodología descrita en el capítulo anterior. El capítulo se encuentra estructurado por iteraciones y los resultados obtenidos en cada una de ellas.

5.1 Resultado

Tras finalizar el desarrollo de este proyecto se ha obtenido:

Un sistema de integración continua, implantado con éxito en la metodología de gestión del ciclo de desarrollo de MADRIJA que permite gestionar el ciclo de pruebas de cualquier proyecto desde el principio hasta el final. Por lo tanto, se confirma haber alcanzado el objetivo principal definido en el capítulo 2 de este proyecto.

Dicho sistema de integración continua tiene la capacidad de realizar diferentes tareas:

- Configura las variables de entorno necesarias para que se realice de manera correcta la compilación de los proyectos.
- Descargar el código fuente desde el repositorio donde trabaja el equipo de desarrollo.
- Utiliza las credenciales necesarias para poder descargarse el código fuente del repositorio.
- Una vez descargado el código fuente del proyecto, es capaz de elegir el *branch* en el cual desea realizar las pruebas, es decir, en el proyecto “X” es capaz de elegir la rama “Y” correspondiente con el cliente “Z” y realizarle las pruebas que sean necesarias sólo a ese cliente.
- Es capaz de detectar cambios en el repositorio para comenzar la ejecución de los *builds*.
- Establece una conexión *maestro-esclavo* mediante el uso de *Java Web Start* para permitir la comunicación entre el servidor de integración continua (maestro) y el equipo donde se van a realizar las ejecuciones (cliente).

- El sistema de integración continua tiene la capacidad de realizar las compilaciones necesarias para cada proyecto, permitiendo así la integración de los diferentes artefactos e infraestructuras. Es decir, tiene la capacidad de compilar los proyectos padres necesarios para que el proyecto sobre el que se desea trabajar se compile de manera correcta.
- Mediante el uso de SSH es capaz de comunicar el servidor con los diferentes equipos.
- Es capaz de utilizar *Vagrant* para crear máquinas virtuales de los diferentes sistemas operativos de cada cliente, incluyendo a cada sistema operativo su configuración correspondiente, permitiendo crear un clon lo más exacto posible del entorno de producción y tener así la capacidad de realizar las pruebas necesarias en un entorno con la mayor similitud posible al entorno real donde se ejecutará el software desarrollado.
- Es capaz de orquestar diferentes tecnologías como *HSQLDB* o *Liquibase* para preparar las bases de datos de prueba necesarias para la realización de pruebas sobre ellas.
- El sistema de integración continua es capaz de empaquetar y mover hacia un contenedor de aplicaciones, en este caso *Apache Tomcat*, los archivos necesarios para “levantar” por completo el código fuente.
- Lanza la ejecución de los diferentes tipos de tests, tests unitarios, tests de comandos, tests de compatibilidad con los diferentes tipos de bases de datos, mediante el uso de “Renombrador” y tests de interfaz de usuario.
- Una vez han finalizado las pruebas, es capaz de destruir los elementos que ha creado para ejecutarlas, es decir, es capaz de destruir la máquina virtual nombrada antes para que el equipo vuelva a su estado anterior.
- Cuando el equipo ha vuelto al estado anterior, notifica mediante correo electrónico a todos los desarrolladores de cada proyecto, el estado en el cual se encuentra el proyecto, es decir, si el proyecto ha superado todos los tests o ha fallado en alguno test.

Por lo tanto, se ha desarrollado con éxito un sistema de integración continua capaz de realizar el ciclo completo de pruebas a cualquier proyecto de MADRIJA. El sistema de integración continua cumple con todos los requisitos que han sido demandados por parte de la empresa durante las distintas reuniones en las distintas iteraciones.

En consecuencia, tras la implantación del sistema de integración continua, MADRIJA ha obtenido una arquitectura e infraestructura donde se han implantado el sistema de integración continua, desde la cual se orquestan a las diferentes máquinas donde se realizan las pruebas necesarias.

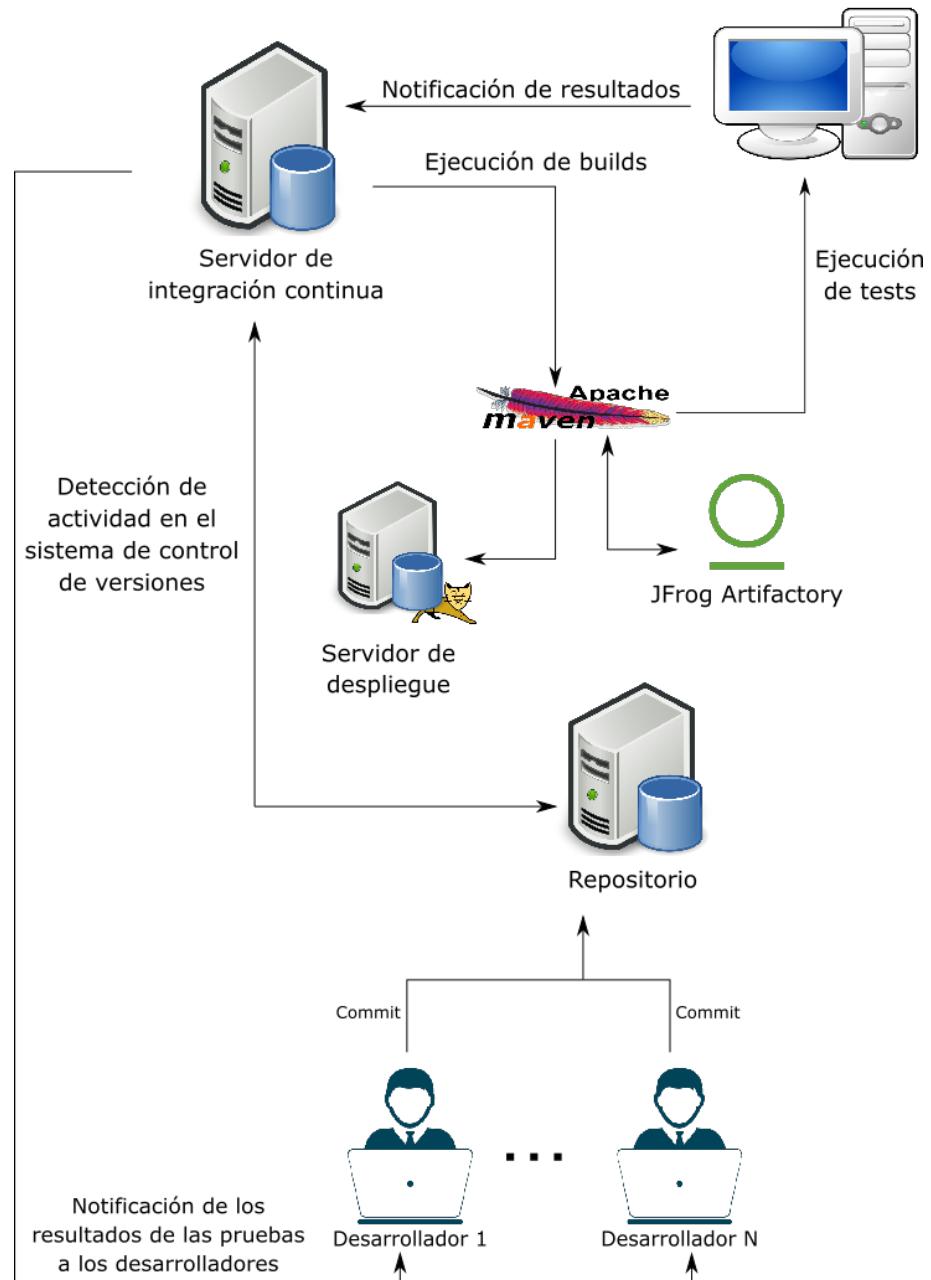


Figura 5.1: Arquitectura e infraestructura del sistema de integración continua

La arquitectura e infraestructura permite al sistema de integración continua comunicarse con los diferentes equipos y servidores que utiliza MADRIJA.

Por una parte, el desarrollador permanece aislado de lo que ocurre, simplemente desarrolla software y coloca todo lo necesario para realizar un *build* en el repositorio (código fuente, *scripts* de bases de datos, etcétera), dicho repositorio se encuentra en constante comunicación con el servidor de integración continua, en cuanto detecta un cambio en el repositorio (*commit*, *push*, etcétera), el servidor lanza las ejecuciones necesarias para realizar el ciclo completo de pruebas, orquesta dichas pruebas con Maven, descarga las librerías que necesita de *JFrog Artifactory*, realiza las compilaciones necesarias para integrar los diferentes artefactos e infraestructuras y lanza la aplicación en el servidor de aplicaciones, finaliza las pruebas y envía los resultados al equipo de desarrollo.

Por otra parte, se ha obtenido una estrategia para lanzar las ejecuciones del servidor de integración continua, es decir, el servidor de integración continua detecta un cambio en el código fuente y lanza las ejecuciones para comprobar que esos cambios no violen ninguna restricción y mantenga el funcionamiento del código fuente de manera correcta.

El software “Renombrator” ha sido mejorado e integrado en el sistema de integración continua, este software permite migrar una base de datos de un tipo a otro.

```

jafer@DESKTOP-SF92852 M5YS ~/repos/Madrija/renombrator
$ mvn test -Dpath=C:\msys64\home\jafer\repos\connect
[INFO] Scanning for projects...
[INFO]
[INFO] Building renombrator 0.0.1-SNAPSHOT
[INFO]
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ renombrator ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:/msys64/home/jafer/repos/Madrija/renombrator/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ renombrator ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ renombrator ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:/msys64/home/jafer/repos/Madrija/renombrator/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ renombrator ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ renombrator ---
[INFO] Surefire report directory: C:/msys64/home/jafer/repos/Madrija/renombrator/target/surefire-reports

-----  

TESTS  

-----  

Running renombrator.Rrenombrator_Test  

C:\msys64\home\jafer\repos\connect\connect-core\src\main\resources\changelogs\connect-core-1.0.xml: Coincidencia palabra reservada, name -> name  

Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.198 sec <<< FAILURE!
```

Figura 5.2: Ejecución y muestra de resultados de “Renombrator”

Ese decir, comprueba que el código y los *scripts* desarrollados para crear las bases de datos sean compatibles con bases de datos de PostgreSQL, Microsoft SQL Server, HSQLDB y Oracle. “Renombrator” es capaz de realizar pruebas como:

- Comprobar que no se viole la restricción del número máximo de caracteres que puede tener un campo de una base de datos, es decir, si la longitud máxima es de 30 caracteres, que no se viole esa restricción.
- Comprueba que no se hace uso de palabras reservadas en cada tipo de base de datos.
- Comprueba que no se haga un uso incorrecto de mayúsculas y minúsculas.

```

private static boolean hasAnyUpper(String name){
    boolean result = false;

    for(int i = 0; i < name.length() && !result; i++){
        char aux = name.charAt(i);

        result = Character.isUpperCase(aux) && !Character.isDigit(aux);
        assertFalse(Character.isUpperCase(aux) && !Character.isDigit(aux));
    }

    return result;
}

/**
 * Comprueba coincidencias con palabras reservadas de Oracle y Postgres
 *
 * @param name nombre original
 * @return nombre corregido
 */
private static String checkReservedWords(File file, String name){
    String result = name;

    if("id".equals(name)){
        System.out.println(file + ": Coincidencia palabra reservada, " + name + " -> " + result);
        assertFalse("id".equals(name));
    }else if(ReserverWords.WORDS.contains(name.toUpperCase())){
        System.out.println(file + ": Coincidencia palabra reservada, " + name + " -> " + result);
        assertFalse(ReserverWords.WORDS.contains(name.toUpperCase()));
    }

    return result;
}

```

Figura 5.3: Parte del código desarrollado para los tests de “Renombrator”

5.2 Iteraciones

5.2.1 Iteración 1

En esta primera iteración se pretende obtener un prototipo del análisis, diseño, desarrollo e implantación del sistema de integración continua para varios proyectos reales de MADRIJA obteniendo así una primera prueba de concepto, se va a seguir una metodología heurística, en concreto la metodología heurística IDEAL.

Identificación del problema

Se realiza una reunión con MADRIJA y mediante el uso de diferentes técnicas de captura de requisitos, se capturan los requisitos que se persiguen en esta iteración y se aprueban estos requisitos mediante otra reunión con miembros de MADRIJA, en esta iteración el objetivo principal consiste en desarrollar una prueba de concepto de un sistema de integración continua para proyectos reales de la empresa en un plazo de dos semanas.

Realización del plan

Una vez obtenidos los requisitos del sistema que desea implantar MADRIJA, se procede al análisis, diseño y desarrollo de un plan que permita abordar y solucionar los problemas identificados.

Entre los requisitos que MADRIJA desea que tenga el sistema de integración continua destacan:

- Capacidad para descargar el código fuente desde el repositorio donde trabaja el equipo de desarrollo.
- Capacidad para realizar las compilaciones necesarias, permitiendo así la integración entre los diferentes artefactos y la infraestructura.
- Capacidad de comunicación con alguna herramienta para crear entornos virtuales.
- Capacidad para ejecutar los tests necesarios para el código fuente descargado.
- Capacidad de destruir los entornos virtuales y los elementos que ha creado para la ejecución de las pruebas.
- Capacidad de comunicar a los miembros del equipo de desarrollo el resultado de las pruebas ejecutadas.

Ejecución del plan

En esta fase de la heurística IDEAL se llevan a cabo las tareas que se han marcado en la fase anterior para llegar al objetivo marcado.

```
/* Driver for browser*/
WebDriver driver = new ChromeDriver(); //new FirefoxDriver();

@Given("^Open_Browser$")
public void openBrowser() throws Throwable {
    /* Firefox */
    // System.setProperty("webdriver.gecko.driver", "geckodriver.exe");

    /* Google Chrome */
    System.setProperty("webdriver.chrome.driver","chromedriver.exe");

    /* URL */
    driver.get("http://localhost:8081/enigma-caule/");

    /* Wait for 1 second */
    Thread.sleep(1000);
}

@When("^Insert_Information$")
public void insertInformation() throws Throwable {
    /* Find the inputs */
    WebElement user = driver.findElement(By.name("j_idt9:j_idt13"));
    WebElement password = driver.findElement(By.name("j_idt9:j_idt15"));

    /* Enter datas */
    user.sendKeys("admin");
    password.sendKeys("1234");

    /* Wait for 1 second */
    Thread.sleep(1000);
}
```

Figura 5.4: Test desarrollado en Java con Selenium

Finalizados todos los pasos desglosados del plan, se realizan tests y pruebas mediante el sistema de integración continua, obteniendo los mismos resultados que sin su implantación, es decir, los resultados obtenidos de las pruebas realizadas a los proyectos con o sin sistema de integración continua son iguales.

Análisis de la solución obtenida

Se realiza una reunión con miembros de MADRIJA para presentarles la solución obtenida, se comentan los problemas que han surgido, MADRIJA acepta la solución propuesta y se cierra esta primera iteración del proyecto. Como se ha dicho explicado antes, se toma el prototipo obtenido de esta iteración y se toma como entrada de la siguiente iteración.

The screenshot shows the Jenkins project configuration interface for a Maven project. The top navigation bar includes tabs for General, Configurar el origen del código fuente, Disparadores de ejecuciones, Entorno de ejecución, Pasos previos (selected), and Proyecto. Below the tabs, there are sections for 'Proyecto' and 'Pasos posteriores'. In the 'Proyecto' section, the 'Fichero POM raíz' is set to 'pom.xml' and the 'Goles y opciones' is set to 'clean install -DskipTests -U'. A 'Avanzado...' button is visible. In the 'Pasos posteriores' section, there are three execution criteria radio buttons: 'Ejecutar sólo en caso de éxito', 'Ejecutar sólo cuando la ejecución fué buena o inestable.', and 'Ejecutar siempre (sea cual sea el resultado)'. A 'Criterios de ejecución' link is provided. An 'Añadir un paso posterior' button is also present. The 'Propiedades del proyecto' section includes a checkbox for 'Notificación por E-mail' which is checked, and a recipient field containing 'jafernandezg9@hotmail.com'. There are 'Guardar' and 'Apply' buttons, along with two checkboxes for email notifications: 'Send e-mail for every unstable build' (checked) and 'Send separate e-mails to individuals who broke the build'.

Figura 5.5: Indicación de objetivos de un proyecto Maven en Jenkins

Elección de la herramienta de integración continua

Debido a la gran variedad de herramientas de integración continua en el mercado se realizó un estudio e investigación para elegir la opción que permita obtener más beneficio para MADRIJA, en consecuencia, se elige Jenkins, debido a que es la herramienta que mejor se adapta a las tecnologías y herramientas utilizadas por MADRIJA, contiene muchos plugins y, además, como plus añadido es una herramienta *open source* y gratuita por lo que no supone ningún coste extra para la empresa. Jenkins superó la primera prueba de concepto.

Además, Jenkins cuenta con una serie de dibujos junto a los proyectos que contiene para informar de las estadísticas de los mismos.

- **Sol:** No hay ejecuciones recientes con fallos.
- **Sol con nubes:** Una de las cinco últimas ejecuciones ha fallado.
- **Nubes:** Dos de las cinco últimas ejecuciones han fallado.
- **Lluvia:** Tres de las últimas cinco ejecuciones han fallado.
- **Tormenta:** Cuatro de las cinco últimas ejecuciones han fallado.

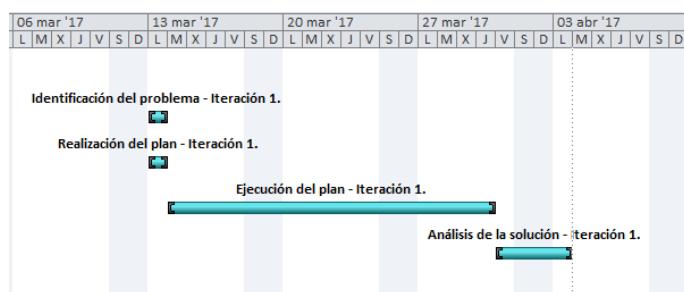


Figura 5.6: Diagrama de Gantt de la iteración 1

Problemas surgidos

Debido a conflictos iniciales entre algunas librerías y herramientas el sistema de integración continua no podía contar el número de tests que ejecutaba, es decir, era capaz de determinar si se realizaban todos bien o si por el contrario algún test fallaba pero no era capaz de suministrar una cifra de tests bien ejecutados y de tests fallidos.

Por lo que debido al desconocimiento de estos problemas que surgieron durante el desarrollo, esta primera iteración duró tres semanas en lugar de las dos semanas que se estimaron inicialmente.

5.2.2 Iteración 2

En esta segunda iteración mediante el análisis, diseño, desarrollo e investigación sobre sistemas de integración continua, se pretende definir un prototipo de una infraestructura y arquitectura para el sistema de integración continua que desea implantar MADRIJA, para realizar esta segunda iteración, se va a utilizar una metodología heurística, en concreto la metodología heurística IDEAL.

Identificación del problema

Se lleva a cabo una reunión con MADRIJA, donde se abordan la necesidad de determinar como debería ser la infraestructura y arquitectura que será necesaria para implantar un sistema de integración continua dentro de MADRIJA y que así el sistema de integración continua permita obtener el mayor rendimiento y la mayor eficiencia.

Realización del plan

Una vez obtenidos los objetivos que demanda MADRIJA que se deben cumplir en esta reunión mediante una reunión y el uso de distintas técnicas de capturas de requisitos, se capturan los requisitos, se presentan a la empresa y son aceptados por ellos, se obtiene como objetivo principal de esta segunda iteración que se hace necesario el análisis, diseño y desarrollo de una arquitectura para el servidor de integración continua.

Pasos a seguir en el plan

Se procede al análisis y estudio de infraestructuras y arquitecturas de sistemas de integración continua, así como sus diseños o proveedores. Derivando así un estudio e investigación sobre arquitecturas de sistemas de integración continua, teniendo en cuenta las ventajas y desventajas que ofrecen cada una de las posibilidades.

Análisis de la solución obtenida

Se presenta un prototipo de infraestructura y arquitectura la cual permite implantar el sistema de integración continua y obtener el mayor beneficio posible para MADRIJA, la empresa acepta y se cierra esta segunda iteración del proyecto.

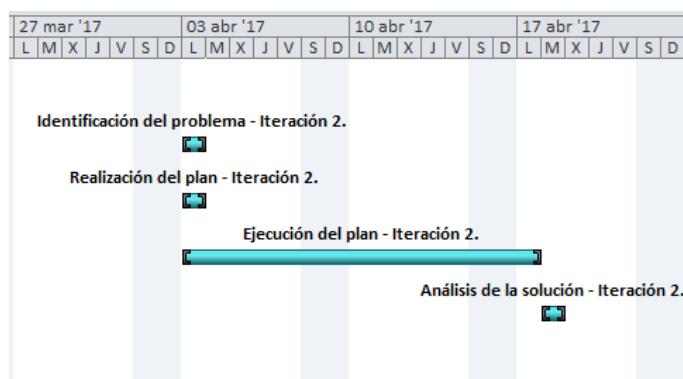


Figura 5.7: Diagrama de Gantt de la iteración 2

5.2.3 Iteración 3

En esta tercera iteración se pretende determinar una estrategia para definir cuando debe actuar el sistema de integración continua y activar las ejecuciones correspondientes, para resolver los problemas detectados en la reunión con miembros de MADRIJA, se va a seguir una metodología heurística, en concreto, la heurística IDEAL.

Identificación del problema

Se realiza una reunión con miembros de MADRIJA para realizar la captura de requisitos, se validan estos requisitos y se obtiene como objetivo principal que se debe proponer una estrategia, o patrón a seguir, mediante el cual el sistema de integración continua inicie sus *builds* y ejecuciones.

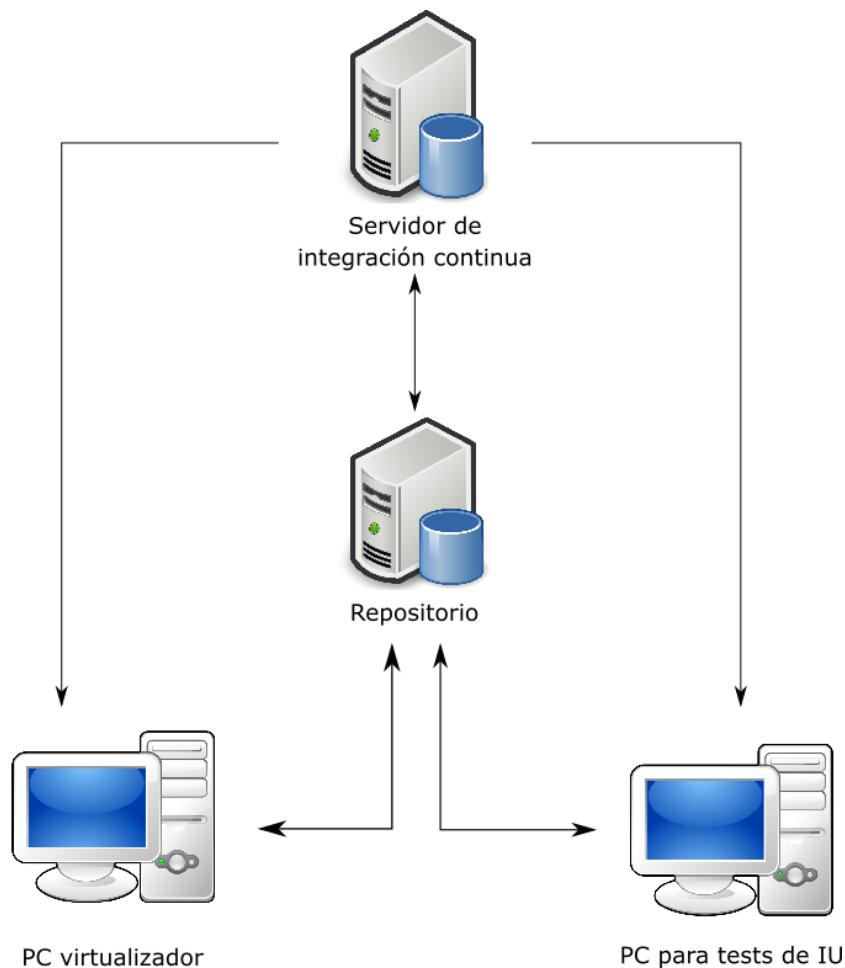


Figura 5.8: Versión inicial de la arquitectura e infraestructura del sistema de integración continua

Realización del plan

Como paso previo a la realización del plan mediante el cual abordaremos los problemas detectados en la reunión con Madrija, se debe realizar un análisis sobre la metodología de trabajo que sigue la empresa, en especial, a su estrategia de ramificación dado que utilizan *git flow*.

Una vez realizado ese estudio, se procede al análisis, diseño y desarrollo de un plan que permita abordar y solucionar los problemas identificados.

Pasos a seguir en el plan

En primer lugar se procede a analizar e investigar sobre las alternativas que ofrece la herramienta utilizada en el sistema de integración continua, con ello y teniendo en cuenta que MADRIJA utiliza *git flow*, encontrar así la mejor alternativa.

Una vez realizado el estudio y la investigación sobre las alternativas ofrecidas por la herramienta se procede a la elección de la mejor alternativa, tras realizar el estudio se detallaron dos alternativas, por una parte, mediante la detección de *commits* y cambios en repositorio, es decir, se elegiría una rama, y cualquier cambio que se realizase en esa rama activaría el circuito de integración continua, por otro lado, surgió la alternativa de activar de manera nocturna todos los días el sistema de integración continua, esta segunda opción era menos convincente dado que recordaba a un “*cron*” de Linux por lo que se ha optado por la primera opción.

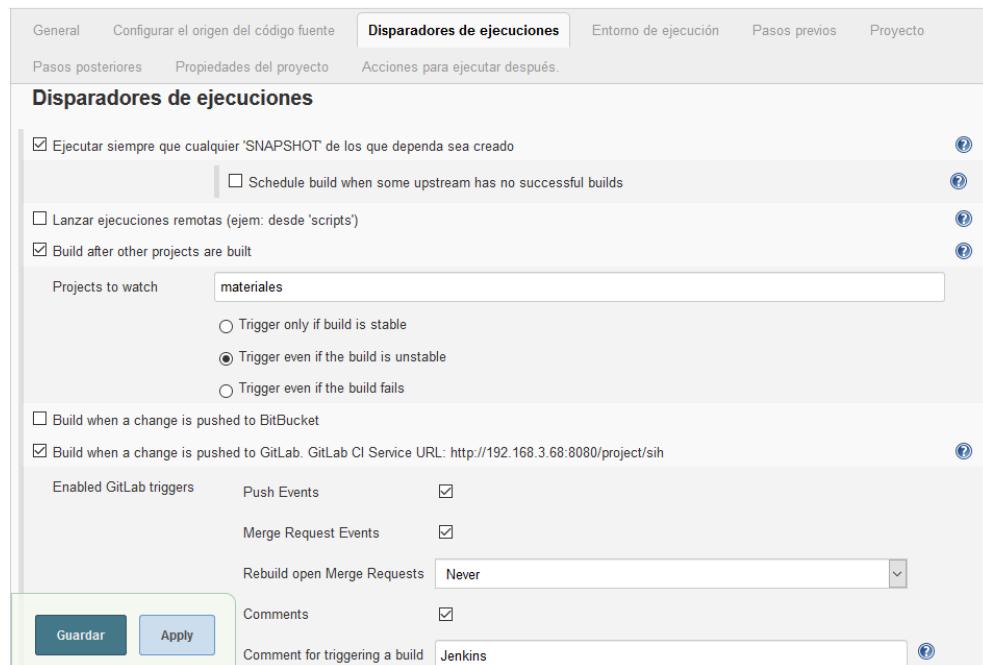


Figura 5.9: Detección de *commit* en un proyecto Jenkins

Análisis de la solución obtenida

Se había decidido implantar una estrategia mediante la detección de *commits*, es decir, si un desarrollador realizaba un *commit*, de manera automática, el sistema de integración continua realizaba todas las ejecuciones involucradas, pero el plugin que realiza dicha acción tiene *issues* por lo que no ha sido posible realizar esa estrategia dado que no se han solucionado los problemas.

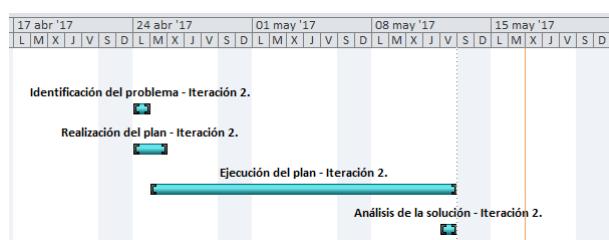


Figura 5.10: Diagrama de Gantt de la iteración 3

Por lo tanto, mientras se mantenga el problema, se ha optado por otra estrategia, que consiste en realizar de manera nocturna y a diario ejecuciones automáticas de todos los proyectos y enviando así correos a los desarrolladores para que a la mañana siguiente tengan un *feedback* de los cambios que realizaron el día anterior en el código fuente.

5.2.4 Iteración 4

Se pretende incorporar al sistema de integración continua la ejecución del software “Renombrator” el cual permite identificar y corregir palabras reservadas en el desarrollo de bases de datos (BBDD).

Para realizar esta cuarta iteración, se va a seguir una metodología heurística, en concreto, la heurística IDEAL.

Identificación del problema

Se realiza una reunión con miembros de MADRIJA para proceder a la captura y validación de requisitos.

Se obtiene como objetivo de esta cuarta iteración que MADRIJA desarrolle software en los que trabaja con BBDD, pero dichas BBDD no son siempre las mismas, dado que un desarrollo puede estar destinado a trabajar con BBDD de Oracle, otro desarrollo con BBDD PostgreSQL, etcétera. Por lo que MADRIJA ha desarrollado “Renombrator”, un software encargado de revisar el software desarrollado referente a BBDD y evitar el uso de palabras reservadas o caracteres incompatibles, entre otros usos.

Pero si por cada cambio que se realiza en el código, el desarrollador debe lanzar “Renombrator”, es un trabajo repetitivo y que no produce beneficios para MADRIJA, únicamente produce pérdidas en el tiempo de desarrollo.

Por lo tanto, tras una reunión con miembros de MADRIJA se desea implantar “Renombrator” en el sistema de integración continua para que cuando algún desarrollador realice algún cambio en el código y haga *commit*, el sistema de integración continua lance la ejecución de “Renombrator” e indique si dicha ejecución ha sido realizada de manera correcta, o si, en caso contrario, se han detectado errores que deben ser solucionados por el desarrollador.

Realización del plan

Una vez obtenidos los requisitos necesarios para esta cuarta iteración, se procede a estudiar la posibilidad y compatibilidad de implantar “Renombrator” en el sistema de integración continua.

Pasos a seguir en el plan

Dado que “Renombrator” es un software que no se ejecuta como test para detectar fallos, se ha decidido utilizar diferentes tecnologías para desarrollar tests mediante los cuales cuando se realice una ejecución del sistema de integración continua se verifique si el código desarrollado es correcto o si, por el contrario, necesita ser revisado dado que se haya encontrado algún error.

Análisis de la solución obtenida

Una vez finalizada la ejecución del plan de esta cuarta iteración, se realiza otra reunión con miembros de MADRIJA para verificar el correcto funcionamiento de “Renombrator”, se valida y, por lo tanto, se concluye esta cuarta iteración.

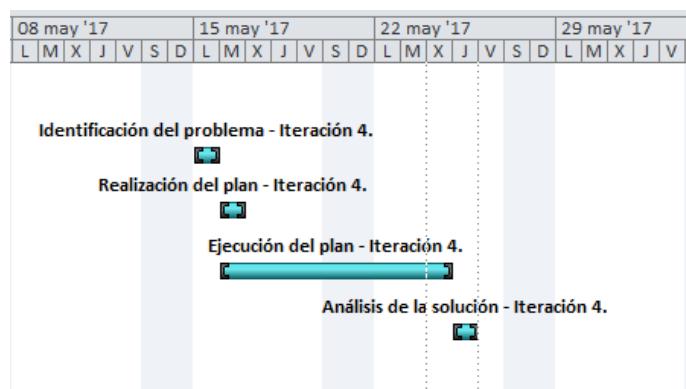


Figura 5.11: Diagrama de Gantt de la iteración 4

Capítulo 6

Conclusiones

EN este último capítulo se determina si se han conseguido los objetivos parciales detallados en el segundo capítulo de este documento, incluyendo el objetivo principal también.

Para concluir, se presentan algunas propuestas de trabajo futuro relacionadas con la temática sobre la que versa este trabajo, se incluye al final una pequeña opinión personal del autor.

6.1 Análisis de los objetivos del proyecto

El objetivo principal de este trabajo era analizar, diseñar e implementar un entorno de integración continua, que se adapte a la metodología, formas y prácticas de desarrollo que tiene MADRIJA. El objetivo ha sido cumplido mediante la implantación de un sistema de integración continua dentro de la empresa así como su instalación y configuración dentro de la arquitectura que fue diseñada para dicho propósito, obteniendo así la implantación del sistema de integración continua y su arquitectura en el servidor de la empresa.

Por lo tanto, el objetivo principal de este trabajo ha sido conseguido y a su vez los objetivos parciales, todo ello documentado a lo largo de este documento.

Objetivo	¿Conseguida?
Analizar, diseñar e implementar un entorno de integración continua, que se adapte a la metodología, formas y prácticas de desarrollo que tiene MADRIJA	

Cuadro 6.1: Objetivo principal

Objetivo	Justificación
Analizar y determinar herramientas de integración continua.	Se justifica haber superado este objetivo parcial en la iteración 1, debido a la prueba de concepto inicial realizada con Jenkins y superada con éxito, se elige a Jenkins como la herramienta que será utilizada para desarrollar el sistema de integración continua.
Desarrollar la infraestructura para el sistema de integración continua.	Se justifica haber superado este objetivo parcial en la iteración 2, dada la aprobación por parte de MADRIJA del prototipo de infraestructura presentado a la empresa para el desarrollo del sistema de integración continua.
Adaptarse a la metodología de desarrollo.	Se justifica haber superado este objetivo parcial en las iteraciones 1 y 3, debido a la prueba de concepto que demostró la capacidad de adaptación de Jenkins y en la iteración 3, debido a la propuesta de cambios para mejorar su metodología de gestión del ciclo de vida de sus desarrollos.
Alcanzar el máximo grado de automatización posible de los tests.	Se justifica haber superado este objetivo en las iteraciones 1 y 4, dado que en la primera iteración como se describe en el documento se consiguen automatizar tests con Maven e incorporarlos en Jenkins, y en la iteración 4, se incorpora a Jenkins “Renombrator”, tests que realizaba la empresa de manera manual y que ahora se realizan de manera automática por parte de Jenkins cuando detecta algún cambio en el código.
Proponer mejoras.	Se justifica haber superado este objetivo parcial dado que en la iteración 3, se proponen mejoras y cambios en la metodología propia de la empresa para gestionar el ciclo de vida de sus desarrollos.

Cuadro 6.2: Análisis de los objetivos parciales

6.2 Trabajo futuro

Tras finalizar este trabajo surgen nuevas cuestiones para un trabajo futuro entre las que destacan:

- **Solucionar el problema del plugin de GitLab:** Para poder realizar otra estrategia para realizar las ejecuciones del sistema de integración continua.
- **Implantación de SonarQube:** Para controlar y medir la calidad del código.
- **Mejorar la seguridad:** El sistema de integración está únicamente protegido por medio de “*usuario y contraseña*” por lo que se podría aportar algún tipo de seguridad extra.

6.3 Opinión personal

La realización de este trabajo me ha servido para adquirir nuevos conocimientos tanto teóricos sobre integración continua y calidad del código fuente desarrollado, como aspectos tecnológicos (configuración de máquinas virtuales, sistemas UNIX, Jenkins, gestión y construcción de proyectos, etcétera).

Durante estos meses he adquirido conocimientos relacionados con integración continua y he reforzado otros conceptos de aspectos tecnológicos.

Concluido este trabajo fin de grado, debo decir que no es un punto y final. El desarrollo de este trabajo y el paso de las dificultades, me han sido útiles para saber qué únicamente conozco una pequeña parte de este gran mundo, la informática, y, por ello, seguiré en formación y buscando nuevos retos que me apasionen tanto o más como la realización de este trabajo.

Por lo tanto, como resultado global de este trabajo me encuentro muy satisfecho y con ganas de seguir mejorando, evolucionando e innovando en este mundo.

Ciudad Real, a 27 de junio de 2017
Fdo. *Julio Alberto Fernández Guerrero*

Referencias

- [1] APACHE SOFTWARE FOUNDATION. Maven. <https://maven.apache.org/what-is-maven.html>, 2002.
- [2] BEACK, K., AND GAMMA, E. Junit. <http://junit.org/>.
- [3] BECK, K. *eXtreme Programming eXplained: Embrace Change*. Addison-Wesley Professional, 2000.
- [4] BECK, K., AND FOWLER, M. *eXtreme Programming Planning*. Addison-Wesley Professional, 2000.
- [5] BECK, K., AND HIGHSMITH, J. The agile manifesto. *Software Development. The lifecycle starts here*. (aug 2001).
- [6] BRANSFORD, J. D., AND STEIN, B. S. The ideal problem solver.
- [7] CIRCLECI. <https://circleci.com/docs/>.
- [8] CUCUMBER. <https://cucumber.io/>.
- [9] DBEAVER. <http://dbeaver.jkiss.org/>.
- [10] DYBÅ, T., AND DINGSØYR, T. Empirical studies of agile software development: A systematic review. *Information and software technology 50*, 9 (2008), 833–859.
- [11] ECLIPSE. <https://eclipse.org/>.
- [12] FOWLER, M., AND FOEMMEL, M. Continuous integration. *Thought-Works* (2006), 122.
- [13] GIT. <https://git-scm.com/doc>.
- [14] GITLAB INC. Gitlab. <https://about.gitlab.com/about/>.
- [15] HASHICORP. <https://www.vagrantup.com/>.
- [16] HSQLDB. <http://hsqldb.org/>.

- [17] HUMBLE, J., AND FARLEY, D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education, 2010.
- [18] INNOTEK GMBH. Virtualbox. <https://www.oracle.com/virtualization/virtualbox/index.html>, jan 2007.
- [19] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION & INTERNATIONAL ELECTROTECHNICAL COMMISSION. *ISO/IEC 12207:2008, Systems and software engineering – Software life cycle processes*, 2008.
- [20] JOSKOWICZ, J. Reglas y prácticas en extreme programming. *Universidad de Vigo*, 22. (aug 2008).
- [21] LAMPORT, L. Latex. <http://www.latex-project.org/about/>, 1989.
- [22] LIQUIBASE. <http://www.liquibase.org/>.
- [23] MICROSOFT. <https://products.office.com/es-es/project/project-and-portfolio-management-software?tab=tabs-1>.
- [24] MONSALVE, E., WERNECK, V., AND LEITE, J. Evolución de un juego educacional de ingeniería de software a través de técnicas de elicitación de requisitos. In *Proceedings of XIII Workshop on Requirements Engineering (WER'2010), Cuenca, Ecuador* (2010), pp. 12–23.
- [25] NONAKA, I. Scrum. <http://www.scrumguides.org/>, 1986.
- [26] OPENPROJECTFOUNDATION. Openproject. <https://www.openproject.org/>, 2010.
- [27] OPENSSH. <https://www.openssh.com/>.
- [28] ORACLE. <http://docs.oracle.com/javase/8/docs/technotes/guides/javaws/>.
- [29] PALACIOS, F. J. P. Enseñanza-aprendizaje de una heurística en la resolución de problemas de física: un estudio cuasiexperimental. *Revista interuniversitaria de formación del profesorado*, 21 (1994), 201–209.
- [30] PRESSMAN, R. S., AND TROYA, J. M. *Ingeniería del software*. McGraw Hill, 1988.
- [31] SELENIUM. <http://www.seleniumhq.org/>.
- [32] SETA, L. D. Una introducción a extreme programming. *Dos Ideas. Personas y Software* (feb 2010).
- [33] SUNMICROSYSTEMS. Java. <https://www.java.com/es/download/faq/whatis.java.xml>, 1995.

- [34] SUNMICROSYSTEMS. Jenkins. <https://jenkins.io/>, 2011.
- [35] TED GOULD, B. H. Inkscape. <https://inkscape.org/en/>, 2003.
- [36] TORO, A. D., AND JIMÉNEZ, B. B. Metodología para la elicitation de requisitos de sistemas software. *Informe Técnico LSI-2000-10. Facultad de Informática y Estadística Universidad de Sevilla* (2000).
- [37] TORVALDS, L. Git. <https://www.atlassian.com/git/tutorials/what-is-git>.
- [38] WELLS, J. D. extreme programming: A gentle introduction.

ANEXOS

Anexo A

Descripción de competencias

Tecnologías de la Información

Computación

Ingeniería del Software

Ingeniería de Computadores

Cuadro A.1: Tecnología específica cursada por el alumno

Competencia	Justificación
Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la Ingeniería del Software.	Este trabajo consiste en la implantación de un entorno de integración continua que garantice un alto grado de automatización de las pruebas, de manera fiable y confiable, debido a que se prevé que este entorno de integración continua trabaje con tareas críticas, y dado que se deben obtener los mismos resultados ejecutando los tests mediante el entorno de integración continua que si se realizaran de manera manual, se justifica que se hará uso de la capacidad de desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, que sean asequibles tanto en su desarrollo como en su mantenimiento, y que apliquen normas de calidad mediante teorías, principios y prácticas de la Ingeniería del Software.
Capacidad para valorar las necesidades del cliente y especificar los requisitos software para satisfacer estas necesidades, reconciliando objetivos en conflicto mediante la búsqueda de compromisos aceptables dentro de las limitaciones derivadas del coste, del tiempo, de la existencia de sistemas ya desarrollados y de las propias organizaciones.	Se requiere analizar y comprender la metodología propia de desarrollo utilizada en Madrija, dado que se debe incorporar en la misma un entorno de integración continua. Esta modificación a la metodología, podrá suponer cambios que generen conflictos y que deban ser resueltos considerando un equilibrio entre los objetivos buscados con la integración continua y la metodología existente. Se justifica así la capacidad indicada.

Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales.	Dadas las necesidades software que se deben satisfacer, y la aparición de conflictos durante la implantación del entorno de integración continua, se deben implementar, verificar y documentar las soluciones software que han sido desarrolladas, para resolver los conflictos surgidos, durante el desarrollo del proyecto para su correcta implantación, justificando así que mediante la implantación del entorno de integración continua, se adquiere la capacidad de identificar y analizar problemas y diseñar, así como de desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas habituales.
Capacidad de dar solución a problemas de integración en función de las estrategias, estándares y tecnologías disponibles.	Mediante la implantación e integración del entorno de integración continua en la metodología de desarrollo de MADRIJA junto con todas sus herramientas, Vagrant o VirtualBox entre otras muchas, siendo necesaria la capacidad para dar solución a problemas de integración en función del conocimiento de estrategias, estándares y tecnologías disponibles.

Cuadro A.2: Justificación de las competencias específicas abordadas en el TFG

Anexo B

Manual de usuario

Tal y como se especifica en el capítulo 4, la máquina virtual que alberga Jenkins es Ubuntu Server x64 por lo que su manual de usuario e instalación se realizará en base a Ubuntu Server x64 y a Windows 10 como estación de trabajo que alberga la máquina virtual.

En la siguiente dirección se encuentra el repositorio de paquetes de Jenkins para automatizar su instalación <https://pkg.jenkins.io/debian-stable/> y actualización. Para poder utilizar este repositorio, primero se debe añadir la clave al sistema:

```
$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
```

Después, debemos añadir la siguiente línea al final del archivo /etc/apt/sources.list

```
$ deb https://pkg.jenkins.io/debian-stable binary/
```

Actualizamos los paquetes e instalamos Jenkins.

```
$ sudo apt-get update  
$ sudo apt-get install jenkins
```

Una vez que se han realizado estos pasos, obtenemos la dirección IP de la máquina virtual y mediante un navegador web entramos a la dirección IP y al puerto 8080, dicho puerto viene por defecto para Jenkins pero se puede cambiar en la configuración.

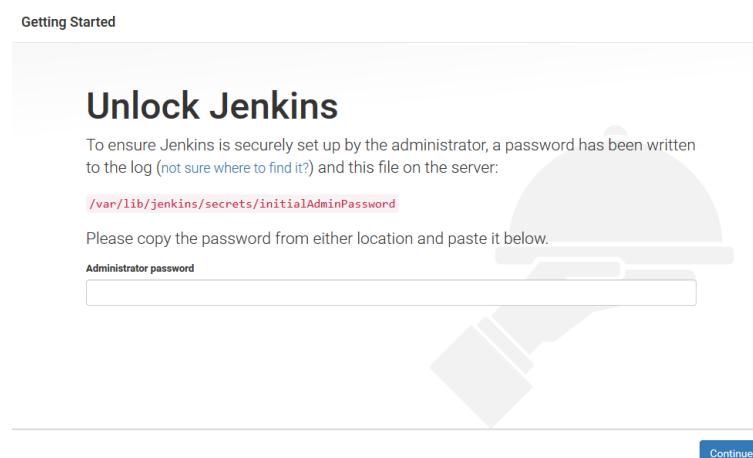


Figura B.1: Petición de contraseña maestra de Jenkins

Una vez introducimos la contraseña maestra, indicamos los plugins que queremos instalar e indicamos los datos del usuario administrador nos aparece el panel principal de Jenkins vacío listo para comenzar a crear proyectos.

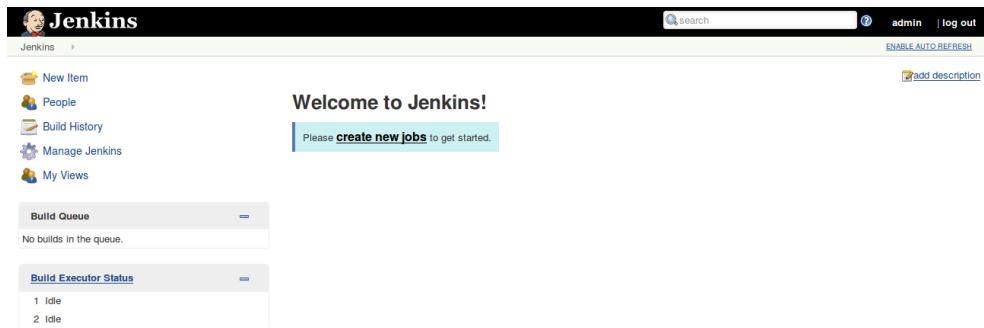


Figura B.2: Panel de bienvenida de Jenkins

Debido a la posibilidad de poder crear proyectos Maven y compilarlos con Java debemos instalar tanto en la máquina virtual como en el huésped ambas herramientas por lo que para la máquina virtual de Ubuntu Server instalaremos Maven y Java de la siguiente manera:

```
$ sudo apt install maven
$ sudo apt install openjdk-8-jdk-headless
```

En el caso de Windows 10 para Java nos dirigimos a su página oficial, descargamos el archivo e instalamos y de igual manera para Maven, tanto en Ubuntu Server como en Windows 10 se deben configurar las variables de entorno `$MAVEN_HOME` y `$JAVA_HOME`.

En el caso de Ubuntu Server, sistema operativo donde se alberga Jenkins, hay que indicarle la ruta de ambas variables para su correcta configuración, además de la ruta donde está instalado Git, sistema de control de versiones utilizado.

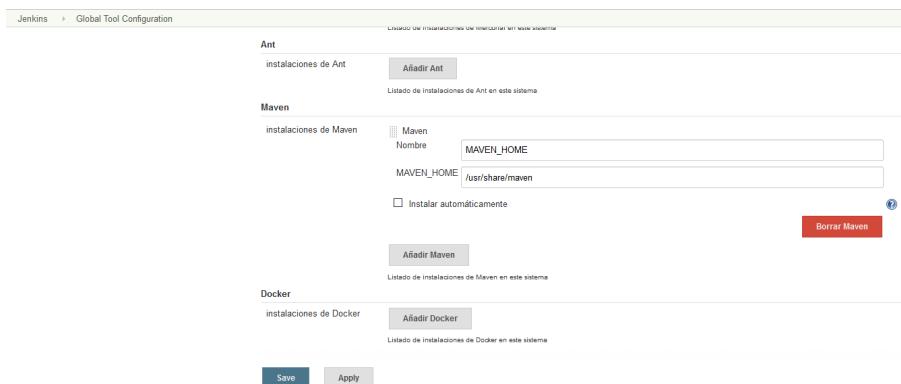


Figura B.3: Parámetros de configuración de Jenkins

Con la configuración actual, podemos realizar ejecuciones en local, es decir, en Ubuntu Server pero lo que queremos para realizar el sistema de integración continua de manera correcta, es que Ubuntu Server sea el orquestador y se realicen las ejecuciones en otros computadores esclavos, por lo que ahora vamos a crear un esclavo con la configuración de Windows 10, al cual debemos indicarle la información dicha anteriormente.

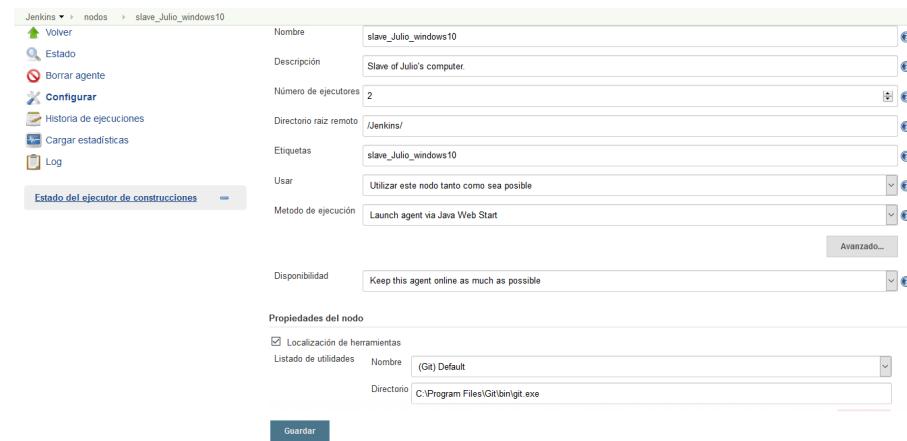


Figura B.4: Parámetros de configuración de un esclavo Jenkins

Una vez realizada esta instalación, podemos crear un proyecto en Jenkins, en este caso un proyecto Maven, donde indicamos la dirección del repositorio donde se encuentra el proyecto para que lo instale, el esclavo donde se va a realizar la ejecución, proyectos padres, las acciones anteriores y posteriores que se deben realizar en la ejecución...

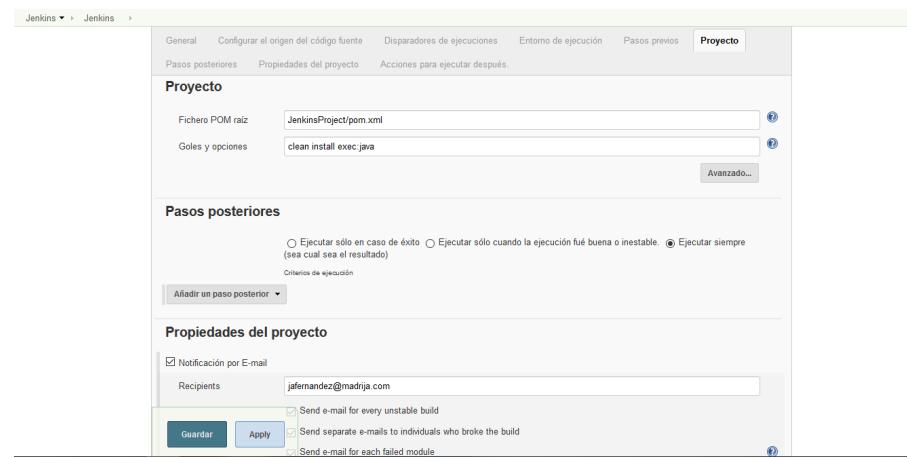


Figura B.5: Configuración de un proyecto Jenkins

Una vez realizadas todas las configuraciones necesarias, se puede ejecutar el proyecto y obtener los resultados de su ejecución.

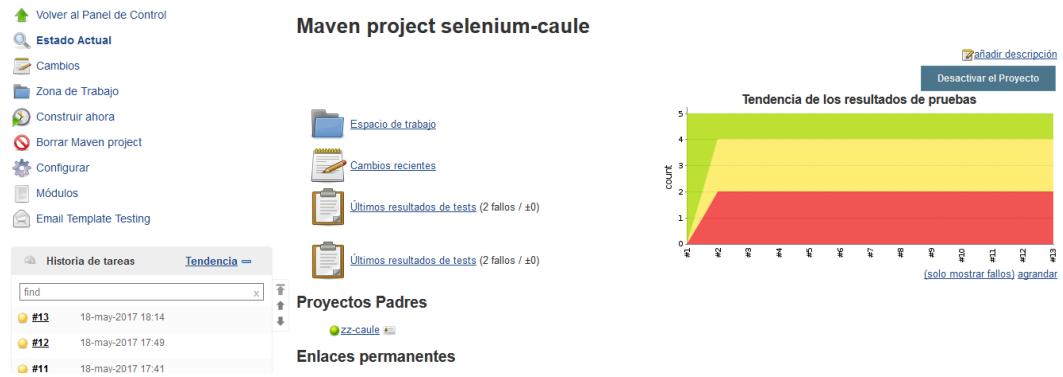


Figura B.6: Resultados tras ejecutar un proyecto Jenkins

Anexo C

Contrato de propiedad intelectual

CONTRATO DE CONFIDENCIALIDAD Y PROPIEDAD INTELECTUAL DE TRABAJO DE FIN DE GRADO (TFG)

En Ciudad Real, a 03 de marzo de 2017

REUNIDOS

DE UNA PARTE, D. **Julio Alberto Fernández Guerrero** mayor de edad, con D.N.I. número **05719150-Q**, alumno de la Escuela Superior de Informática, y en adelante el “ALUMNO DEL TFG”.

DE OTRA PARTE,

- D. **Miguel Ángel Laguna Lobato** mayor de edad, con D.N.I. número **05682598-B**, Director de I+D de Madrija Consultoría, S.L. en adelante “DIRECTOR DEL TFG”,
- D. **Manuel Ángel Serrano Martín** mayor de edad, con D.N.I. número **05655675-K**, Profesor de la Escuela Superior de Informática, en adelante “TUTOR DEL TFG”,
- D. **Ismael Moreno Fernández**, mayor de edad, con D.N.I. número **05680243-W**, representante legal de **Madrija Consultoría, S.L.** en adelante “MADRIJA”.

Todos ellos reconociéndose mutuamente capacidad suficiente para la celebración del presente Contrato,

EXPONEN

PRIMERO: El objeto del presente contrato es la determinación de la propiedad intelectual del TFG titulado “**Implantación de un sistema de integración continua en una metodología consolidada**”, y del resultado de la integración del mismo con las herramientas usadas por MADRIJA así como, establecer el compromiso de la confidencialidad que el ALUMNO DEL TFG contraerá con MADRIJA.

SEGUNDO: La propiedad intelectual del producto software resultante del TFG integrado con las herramientas y entornos de la empresa MADRIJA pertenecerá única y exclusivamente a MADRIJA, no pudiendo el ALUMNO DEL TFG utilizarlas sin el consentimiento expreso de la empresa. Sin embargo, la propiedad intelectual del TFG de manera aislada, así como de los documentos, diagramas, modelos, etc. que lo acompañen será compartido a partes iguales entre el ALUMNO DEL TFG y el DIRECTOR DEL TFG.

TERCERO: El ALUMNO DEL TFG trabajará para la realización de su TFG con documentación, productos software y entornos de la empresa MADRIJA y por ello, por medio de la presente, se compromete de forma irrevocable ante MADRIJA a:

1. Asumir las políticas, los objetivos y las directrices establecidas por MADRIJA, y particularmente la política de calidad, las directrices y procedimientos incluidos o referenciados en el manual de calidad de MADRIJA.
2. Actuar de forma que se preserve una imagen positiva y eficaz de MADRIJA guardando al debida diligencia en relación con todas las labores efectuadas, actuando en todo momento de buena fe y en aras al leal cumplimiento de mis obligaciones para con MADRIJA.
3. Evitar intervenir en cualquier actividad que pueda disminuir la confianza en la competencia, imparcialidad, juicio o integridad operativa de MADRIJA.
4. Asegurar la imparcialidad y estar libre de toda presión indebida, comercial, financiera o de otra índole, que pueda influir en el juicio técnico empleado en la realización de sus actividades en MADRIJA.
5. No alegar su calidad de colaborador de MADRIJA con fines ajenos a las misiones que se le hayan confiado por MADRIJA. No admitir ningún tipo de retribución ajena a MADRIJA por cualquier actividad realizada en nombre de MADRIJA o en calidad de persona cualificada por MADRIJA.
6. En particular, durante la realización de las actividades del TFG:
 - Detectar desviaciones o no conformidades basadas en evidencias objetivas y no en opiniones personales o valoraciones subjetivas.
 - No adoptar nunca actitudes de prepotencia ante la empresa.
 - No tener inconveniente en modificar una opinión si la empresa le demuestra que estaba equivocado.
 - No ceder ante presiones o coacciones que pueda recibir si está convencido de sus criterios y no le demuestran con contrario.
7. Participar, en la medida de lo posible, en las actividades, reuniones o cursos de armonización de criterios siempre y cuando sea requerido por MADRIJA. Compartir con el personal de MADRIJA, su conocimiento, experiencia, ideas y sugerencias.
8. Avisar a MADRIJA de cualquier cambio o interrupción en sus actividades profesionales.
9. Durante su relación con MADRIJA y tras ella, guardar el más estricto secreto sobre todos los asuntos, documentos y registros confidenciales de carácter técnico, organizativo, comercial y económico de MADRIJA y aquella que pudiera tener conocimiento dentro de marco de sus actividades. Tratando como documentación confidencial:
 - Solicitud, formal o informal, de un proyecto. Queda incluido el hecho de la existencia de la solicitud.
 - La entrega por una organización que ha solicitado o ha realizado un proyecto con MADRIJA.
 - Los datos de los productos o servicios o de las organizaciones que ha solicitado o ha realizado un proyecto con MADRIJA.
 - Toda la correspondencia entre MADRIJA y las organizaciones con que trabaja o tiene relación.
 - Toda la correspondencia entre MADRIJA y sus subcontratistas y viceversa.
 - La que los documentos del sistema señalen como confidencial.
 - Los documentos del sistema de gestión de MADRIJA.

10. Adoptar todas las precauciones para evitar que se divulguen, directa o indirectamente, por causa propia o de las personas de las que es responsable, documentos o informaciones de los que pudiera tener conocimiento dentro del marco de sus actividades en los proyectos de MADRIJA. Estas precauciones incluyen:
- Manipular y mantener debidamente archivada la documentación.
 - No dejar incontrolada documentación confidencial en salas de reuniones y visitas.
 - Evitar que personas ajenas accedan a lugares de trabajo y archivos.
11. En caso de acceso y tratamiento de datos de carácter personal, por exigencias el trabajo que desempeñe, asegurará la confidencialidad de los mismos respetando y cumpliendo en todo momento los procedimientos y directrices instaurados en MADRIJA al efecto y la legislación vigente aplicable.
12. Respetar la titularidad de MADRIJA sobre cualquier proyecto, informe, documento e imágenes que realice en el desempeño de las tareas que se le encomiendan durante el TFG (respetando y protegiendo los derechos de propiedad intelectual o industrial de MADRIJA).

Y en prueba de cuando antecede, las Partes suscriben el Contrato, en tres ejemplares y a un solo efecto, en el lugar y fecha señalados en el encabezamiento.

El Alumno del TFG  Fdo: Julio A. Fernández Guerrero	El Director del TFG  Fdo: Miguel A. Vojeme Loboete
El Tutor Académico del TFG  Fdo: Manuel Ángel Serrano Martín	El Representante Legal de MADRIJA  Ctra. B4509 km 133, Edificio P - Oficina 100 Tel.: 925 260 939 www.madrija.com

Este documento fue editado y tipografiado con L^AT_EX empleando la clase **esi-tfg** (versión 0.20170627) que se puede encontrar en:

<https://bitbucket.org/arco-group/esi-tfg>

