



UNIVERSIDAD DE VALPARAÍSO  
Facultad de Ingeniería  
Escuela de Ingeniería Civil en Informática

# **IMPLEMENTACIÓN DE UNA INFRAESTRUCTURA PARA EL DESARROLLO COLABORATIVO DE SOFTWARE CON FINES DOCENTES.**

TRABAJO REALIZADO PARA OPTAR AL TITULO  
PROFESIONAL DE  
**INGENIERO EN INFORMÁTICA**

**Luis Alberto Maldonado Gonzalez**  
Profesor Guía: Gabriel Astudillo  
Enero 2017

Certifico que he leído este documento y que, en mi opinión, es adecuado en ámbito y calidad como trabajo para optar al título de Ingeniero Civil en Informática.

---

Gabriel Astudillo Profesor Guía

Certifico que he leído este documento y que, en mi opinión, es adecuado en ámbito y calidad como trabajo para optar al título de Ingeniero Civil en Informática.

---

Rene Noel Profesor Co-Referente

Aprobado por la Escuela de Ingeniería Civil en Informática, UNIVERSIDAD DE VALPARAÍSO.

# Índice general

<b>1. Introducción</b>	<b>1</b>
<b>2. Marco conceptual</b>	<b>3</b>
2.1. Ingeniería de software . . . . .	3
2.2. Método ágil . . . . .	4
2.3. Nube de computo . . . . .	5
2.3.1. Virtualización . . . . .	5
2.4. Control de versiones . . . . .	6
2.4.1. Esquemas de almacenamiento . . . . .	7
2.5. Marco Técnologico . . . . .	8
<b>3. Estado del arte</b>	<b>9</b>
3.1. Sistemas, técnicas y/o métodos actuales . . . . .	9
<b>4. Definición del problema y solución propuesta</b>	<b>12</b>
4.1. Contexto . . . . .	12
4.2. Solución propuesta . . . . .	13
4.3. Naturaleza del cambio . . . . .	14
4.4. Objetivos . . . . .	14
4.4.1. Objetivo general . . . . .	14
4.4.2. Objetivos específicos . . . . .	14
4.5. Aporte . . . . .	15
4.6. Metodología de investigación y desarrollo . . . . .	15
4.6.1. Investigación . . . . .	15
4.6.2. Desarrollo . . . . .	15
<b>5. Análisis</b>	<b>16</b>
5.1. Especificación de requerimientos . . . . .	16
5.1.1. Requerimientos funcionales . . . . .	16
5.1.2. Requerimientos no funcionales . . . . .	16
5.2. Diagrama de casos de uso (DCU) . . . . .	17

<b>6. Diseño</b>	<b>18</b>
6.1. Diseño arquitectónico . . . . .	18
6.2. Descomposición arquitectural . . . . .	19
6.2.1. Nivel Físico . . . . .	19
6.2.2. Nivel virtual . . . . .	20
6.3. Nivel conceptual . . . . .	21
6.3.1. Modelo general . . . . .	22
6.3.2. Lista de características . . . . .	23
6.3.3. Planear por característica . . . . .	24
6.3.4. Diseño por característica . . . . .	25
6.3.5. Construcción de las características . . . . .	26
6.4. Roles y responsabilidades . . . . .	27
<b>7. Herramientas</b>	<b>29</b>
7.1. Comunicación - Slack . . . . .	30
7.2. Repositorio de contenidos - GIT . . . . .	30
7.3. Gestión del flujo de trabajo - SourceTree . . . . .	31
7.4. Integración continua - Jenkins . . . . .	31
7.5. Diagrama final de la solución . . . . .	31
7.6. Diseño de pruebas . . . . .	32
7.6.1. Pruebas unitarias . . . . .	32
<b>8. Implementación</b>	<b>33</b>
8.1. Hardware de desarrollo . . . . .	33
8.2. Herramientas para el desarrollo . . . . .	34
8.2.1. Nivel virtual - Proxmox Virtual Enviorment . . . . .	34
8.2.2. Servidores virtuales - Estructura operacional . . . . .	35
8.2.3. Herramientas del sistema - Versión . . . . .	37
<b>9. Vistas</b>	<b>38</b>
9.1. Proxmox Virtual Enviorment . . . . .	38
9.2. Gitlab . . . . .	39
9.3. Jenkins . . . . .	41
9.4. Navegabilidad del sistema . . . . .	44
9.4.1. Proxmox . . . . .	44
9.4.2. Gitlab . . . . .	45
9.4.3. Jenkins . . . . .	45

<b>10.Pruebas</b>	<b>46</b>
10.1. Pruebas de requerimientos . . . . .	46
10.2. Pruebas funcionales . . . . .	48
10.2.1. Detalle de las pruebas funcionales . . . . .	49
10.2.2. Análisis de resultados . . . . .	56
10.3. Pruebas de sistema . . . . .	57
10.3.1. Pruebas de carga . . . . .	58
10.3.2. Pruebas de estrés . . . . .	59
10.3.3. Análisis de resultados . . . . .	61
10.4. Pruebas de validación . . . . .	61
10.4.1. Análisis de resultados . . . . .	61
<b>11.Implantación</b>	<b>63</b>
11.1. Características del ambiente . . . . .	63
11.2. Actividades previas a la implantación . . . . .	63
11.3. Actividades propias de la implantación . . . . .	64
<b>12.Conclusión</b>	<b>65</b>
<b>A. Manual de implantación</b>	<b>66</b>
A.1. Instalar Proxmox VE 3.4 . . . . .	66
A.2. Instalar JDK Development Environment 1.8 . . . . .	67
A.2.1. Instalar PHP 5.6 . . . . .	67
A.2.2. Instalar Gitlab 8.10 . . . . .	68
A.3. Instalar Jenkins . . . . .	68
A.4. Administrador Proxmox . . . . .	68
A.5. Administrador Jenkins . . . . .	68
A.6. Desarrollador Gitlab . . . . .	69
A.7. Flujo de trabajo . . . . .	69
<b>Bibliografía</b>	<b>70</b>

# Índice de tablas

7.1. Modelo general de pruebas . . . . .	32
10.1. Tabla de evaluación de requerimientos . . . . .	47
10.2. Lista de requerimientos . . . . .	47
10.3. Métricas de evaluación . . . . .	48
10.4. Resumen de resultados . . . . .	49
10.5. Resumen de resultados . . . . .	49
10.6. Resumen de resultados . . . . .	50
10.7. Resumen de resultados . . . . .	50
10.8. Resumen de resultados . . . . .	51
10.9. Resumen de resultados . . . . .	51
10.10Resumen de resultados . . . . .	52
10.11Resumen de resultados . . . . .	52
10.12Resumen de resultados . . . . .	52
10.13Resumen de resultados . . . . .	53
10.14Resumen de resultados . . . . .	53
10.15Resumen de resultados . . . . .	53
10.16Resumen de resultados . . . . .	54
10.17Resumen de resultados . . . . .	54
10.18Resumen de resultados . . . . .	54
10.19Resumen de resultados . . . . .	55
10.20Resumen de resultados . . . . .	55
10.21Resumen de resultados . . . . .	56
10.22Resumen de resultados . . . . .	56

# Índice de figuras

2.1.	Virtualización en tres niveles . . . . .	5
2.2.	Sistema manual de control de versiones . . . . .	6
2.3.	Esquema simplificado de un sistema de control de versiones centralizado	7
2.4.	Esquema simplificado de un sistema de control de versiones distribuido	8
3.1.	Abstracción del plan de estudio de la Universidad . . . . .	10
4.1.	Modelo de la situación actual . . . . .	13
4.2.	Arquitectura de alto nivel de la solución . . . . .	14
4.3.	Metodología de trabajo . . . . .	15
5.1.	Diagrama de casos de uso . . . . .	17
6.1.	Diagrama general de la solución . . . . .	19
6.2.	Nivel Físico . . . . .	19
6.3.	Arquitectura hosted v/s hypervisor . . . . .	20
6.4.	Esquema del nivel virtual . . . . .	20
6.5.	Diagrama de bloques del proceso . . . . .	21
6.6.	Diagrama de flujo de la primera etapa. . . . .	22
6.7.	Diagrama de flujo de la segunda etapa. . . . .	23
6.8.	Diagrama de flujo de la tercera etapa . . . . .	25
6.9.	Diagrama de flujo de la cuarta etapa . . . . .	26
6.10.	Diagrama de flujo de la quinta etapa. . . . .	27
6.11.	Modelo jerárquico de roles . . . . .	28
7.1.	Ejemplo de funcionamiento . . . . .	29
7.2.	Ejemplo de funcionamiento . . . . .	31
7.3.	Modelo de la solución . . . . .	31
8.1.	Modelo de red del sistema . . . . .	35
9.1.	Login Proxmox VE . . . . .	38
9.2.	Login Proxmox VE . . . . .	38

9.3.	Pantalla de carga de imágenes y plantillas . . . . .	39
9.4.	Pantalla de inicio Gitlab . . . . .	39
9.5.	Creación nuevo proyecto . . . . .	39
9.6.	Creación nuevo proyecto . . . . .	40
9.7.	Creación nuevo proyecto . . . . .	40
9.8.	Creación nuevo proyecto . . . . .	41
9.9.	Login Jenkins . . . . .	41
9.10.	Pantalla de inicio Jenkins . . . . .	42
9.11.	Crear/Configurar proyecto . . . . .	42
9.12.	Crear/Configurar proyecto . . . . .	43
9.13.	Construir proyecto . . . . .	43
9.14.	Salida de consola . . . . .	43
9.15.	Flujo general del sistema . . . . .	44
9.16.	Diagrama de navegabilidad de la web GUI de Proxmox . . . . .	44
9.17.	Diagrama de navegabilidad de la web GUI de Gitlab . . . . .	45
9.18.	Diagrama de navegabilidad de la web GUI de Jenkins . . . . .	45
10.1.	Diagrama de entrada y salida . . . . .	48
10.2.	Diagrama de entrada y salida . . . . .	49
10.3.	Diagrama de entrada y salida . . . . .	50
10.4.	Diagrama de entrada y salida . . . . .	51
10.5.	Diagrama de entrada y salida . . . . .	53
10.6.	Diagrama de entrada y salida . . . . .	54
10.7.	Diagrama de entrada y salida . . . . .	55
10.8.	Diagrama de entrada y salida . . . . .	55
10.9.	Diagrama de entrada y salida . . . . .	56
10.10.	Detalle del plan de pruebas . . . . .	57
10.11.	Detalle del plan de pruebas . . . . .	57
10.12.	Diagrama de pruebas . . . . .	58
10.13.	Grafico de resultados . . . . .	58
10.14.	Diagrama de pruebas . . . . .	58
10.15.	Grafico de resultados . . . . .	59
10.16.	Diagrama de pruebas . . . . .	59
10.17.	Grafico de resultados . . . . .	59
10.18.	Diagrama de pruebas . . . . .	60
10.19.	Grafico de resultados . . . . .	60
10.20.	Diagrama de pruebas . . . . .	60
10.21.	Grafico de resultados . . . . .	61
10.22.	Diagrama de validación . . . . .	61
10.23.	Desempeño del repositorio central . . . . .	62

A.1. Selector de lenguaje . . . . .	66
A.2. Formatear disco duro . . . . .	67
A.3. Configurar red . . . . .	67
A.4. Flujo general del sistema . . . . .	69

# **Capítulo 1**

## **Introducción**

Resultado de la masificación de uso de TIC en la industria, existe gran demanda por especialistas preparados en el área de las ciencias de la computación. En el estudio de esta materia, el aprendizaje de la programación, es uno de los temas más difíciles de aprender. En la resolución de problemas se requiere de elevadas habilidades analíticas y la aplicación de conceptos abstractos. Por lo que estudiar estas ideas, es a menudo, un desafío [3]. En el ámbito del desarrollo de software, adjunto a las destrezas en programación, son necesarias el uso de metodologías, enfoques y herramientas que ayuden a ordenar el proceso de elaboración del software, con el fin de asegurar la producción de aplicaciones y sistemas de calidad, en donde las operaciones participantes de los procesos computarizados sean usables, confiables y mantenibles. Algunas universidades, en respuesta al actual funcionamiento de la industria, han implementado infraestructuras más realistas para apoyar a los alumnos en su desarrollo [10], en el contexto del trabajo colaborativo, apoyado con herramientas estándar [7] en conjunto con estrategias integradoras, diseñadas con el propósito de mejorar el desempeño académico, que luego se traduzca en graduados habilidosos, seguros y competentes. A continuación, se ilustra una breve descripción de los capítulos y sus contenidos, donde se pueden apreciar las etapas de desarrollo de este trabajo. El Capítulo 2 se definen los principales conceptos que serán abordados durante el desarrollo del trabajo. Luego en el Capítulo 3 se definen el problema específico detectado y la solución propuesta. En la etapa de análisis, se especifican la definición de requerimientos, las funcionalidades que debe poseer la infraestructura. En el Capítulo de Diseño, se define se dividen las tareas y se definen las el modelo que se implementara. En el Capítulo 8 se presenta la implementación de cada uno de los niveles que forman parte de sistema. Posteriormente, en el Capítulo 10, se realizan las pruebas correspondientes, que nos permitan asegurar con un grado aceptable de confianza y evidencia documentada, el funcionamiento correcto y constante del sistema, en condiciones normales y límites de operación. El Capítulo 11 detalla los pasos que se llevaron a cabo para la implantación

del sistema en su ambiente de producción. Finalmente, en el Capítulo 11, se exponen las conclusiones de este trabajo de título.

# **Capítulo 2**

## **Marco conceptual**

En este capítulo se definen los conceptos que formarán parte del dominio de la solución propuesta.

### **2.1. Ingeniería de software**

Las ciencias de la computación e informática es un área compleja [3] que requiere entendimiento de las matemáticas, fundamentos de la computación y la aplicación de metodologías en el desarrollo de soluciones innovadoras. La enseñanza de esta práctica a tomado diferentes acercamientos durante el tiempo, dependiendo de cuales métodos se adapten mejor a un ambiente académico y/o a las necesidades de un determinado entorno.

El desempeño académico de los alumnos, normalmente, es medido por medio de pruebas, exámenes, quizzes, trabajos, etc. La principal misión en cada semestre, en gran parte de las instituciones que imparten carreras relacionadas a las ciencias de la computación e informática, es ofrecer educación de calidad en orden de producir hábiles y competentes graduados.

Aprender técnicas y lenguajes de programación es uno de los requerimientos para obtener el título de Ingeniero Civil en Informática en la Universidad de Valparaíso. Esta habilidad requiere el desarrollo del pensamiento creativo, que será de utilidad, en la construcción de software más complejo, donde es necesaria la aplicación de conceptos relacionados a la ingeniería de software para ordenar las ideas de los alumnos y optimizar la aplicación de las habilidades y competencias que progresivamente adquieren.

Las prácticas actuales en la enseñanza de Ingeniería de software no están preparando adecuadamente a los estudiantes para el mundo real [16]. Se resume el problema en los siguientes puntos.

- No hay producto: Los alumnos desarrollan proyectos en lugar de productos.
- Baja duración: Un semestre o dos, no hay continuidad.
- Alto recambio: Los alumnos no se están desarrollando en base a experiencias.
- Baja complejidad: Debido al tiempo que requiere entrenar las habilidades.
- No hay cliente: La mayoría de los proyectos no tiene contacto con un cliente.

Existe mucha literatura sobre acercamientos en la enseñanza de la Ingeniería de software, por muchos años fue enseñado el modelo en cascada, ya que era considerado la mejor estructura para el desarrollo de software, luego fue el modelo incremental [6]. Más recientes se tiene la programación extrema y los métodos ágiles [12].

## 2.2. Método ágil

Los métodos de desarrollo ágil están intentado ofrecer respuestas a las organizaciones que buscan ciclos cortos de desarrollo con rápido proceso de elaboración del producto [15]. Los métodos ágiles se centran en resolver requerimientos a través de procesos colaborativos de diseño, donde el software es continuamente liberado para que los desarrolladores y el usuario final puedan experimentar con las funcionalidades que se implementan a lo largo de este ciclo. Los desarrolladores implementan requerimientos de usuario en periodos cortos y los usuarios pueden revisar las funcionalidades creadas, para así dar su opinión y sugerencias (Feedback). Luego, los desarrolladores pueden tomar en cuenta estas opiniones y sugerencias en un ciclo futuro del desarrollo. Esto genera un ambiente altamente interactivo entre el desarrollador y el usuario del sistema, en consecuencia se tiene un rápido cumplimiento de los requerimientos de usuario. Estos métodos tienden a separar los requerimientos de un proyecto en segmentos más pequeños. Asegurando retroalimentación constante con el usuario en cada tarea e iteración del proyecto. Los segmentos son planeados, implementados y testeados individualmente, para mantener altos estándares de calidad.

## 2.3. Nube de computo

Puede ser definido como el uso de hardware computacional nuevo o existente y tecnologías de virtualización para formar una estructura compartida que proporciona servicios basados en la web. La 'Nube de computo' ofrece muchos beneficios a las organizaciones. Posibilita la colaboración entre comunidades y grupos de trabajo dispares, ha superado las dificultades que han afectado a soluciones comerciales existentes y facilitado el acceso a gran cantidad de recursos computacionales de forma eficiente. Los tres modelos predominantes son software, plataforma e Infraestructura como servicio.

### 2.3.1. Virtualización

La virtualización ya existe en el ecosistema de las tecnologías de la información y comunicación. Este concepto esconde complejidad subyacente que permite a múltiples clientes, con diversas aplicaciones y demandas, utilizar servidores de forma simultánea.

Los tres modelos dominantes son presentados en la [Figura 2.1]. En la capa superior, se encuentra el software como servicio (SaaS). En la segunda capa, se tiene Plataforma como servicio (PaaS). Por ultimo Infraestructure como servicio (IaaS).

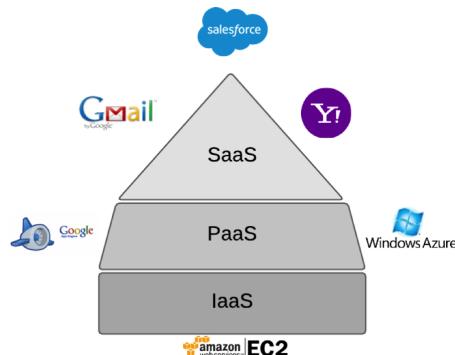


Figura 2.1: Virtualización en tres niveles

- Software como servicio: Permite a usuarios acceder a software y servicios que residen en la 'Nube' y no en el dispositivo de usuario. Los consumidores de aplicaciones 'SaaS' solo requieren de un pequeño cliente para hacer uso de estos recursos. Esto reduce los requerimientos de hardware para los usuarios finales.

Algunos ejemplos populares son Gmail y Google Apps.

- Plataforma como servicio: Provee acceso a API's, lenguajes de programación y capas intermedias, que permite a los suscriptores desarrollar aplicaciones personalizadas sin necesidad de instalar o configurar el entorno de desarrollo. Usando las herramientas incluidas en esta plataforma, desarrolladores pueden construir aplicaciones y servicios tomando ventaja del hardware virtualizado, redundancia de datos y alta disponibilidad. Una vez el desarrollo es completado, la aplicación puede ser distribuida a los usuarios vía internet. Google App Engine, Microsoft Azure and SalesForce.com son ejemplos de PaaS.
- Infraestructura como servicio: Puede ser definido como el uso de servidores, almacenamiento y virtualización para facilitar servicios a los usuarios. La infraestructura consiste en las instalaciones, redes de comunicación, nodos físicos de computo y lote de recursos computacionales virtualizados y administrados por los proveedores de servicios.

## 2.4. Control de versiones

Se denomina versión, al estado en el que se encuentra un producto en el tiempo. El control de versiones tiene que ver con la gestión de los cambios generados en los elementos que componen un producto. Si bien, este control puede ser realizado manualmente [Figura 2.2].

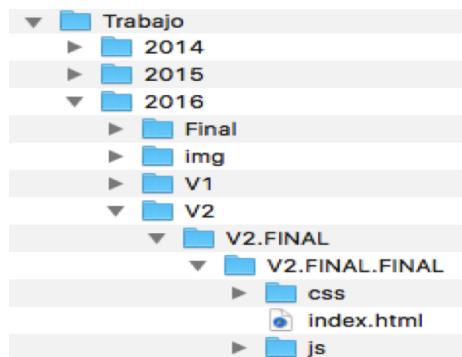


Figura 2.2: Sistema manual de control de versiones

En la industria informática son utilizados los sistemas de control de versiones [SCV] para la gestión del código fuente de proyectos. Estas herramientas facilitan

principalmente el almacenamiento de archivos, la recuperación de cada uno de ellos y el registro histórico e identificación de las modificaciones realizadas en las sucesivas versiones. En general, es beneficioso el uso de SCV debido a su capacidad de potenciar el trabajo en paralelo y distribuido, facilitando la realización de cambios y respectiva unión entre ellos. Cuando un elemento es bloqueado, impidiendo que otros usuarios puedan hacer uso de éste, nos encontramos bajo un esquema de funcionamiento exclusivo. Por otro lado, si grupos de usuarios pueden trabajar sobre un mismo elemento, estamos en presencia de un esquema colaborativo.

#### 2.4.1. Esquemas de almacenamiento

- Centralizados: Estos sistemas, como CVS, Subversion, y Perforce, tienen un único servidor que contiene todos los archivos versionados, y varios clientes que descargan las versiones desde ese repositorio central. La edición de un archivo específico no soporta múltiples usuarios y se requiere estar conectados constantemente al repositorio central [Figura 2.3].

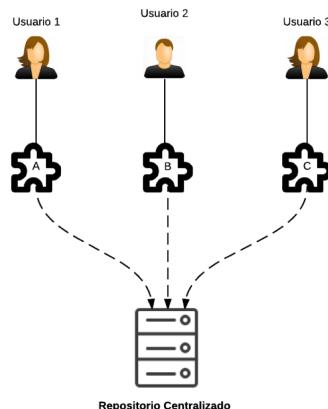


Figura 2.3: Esquema simplificado de un sistema de control de versiones centralizado

#### Principales características de los sistemas centralizados

1. Control central del avance del proyecto.
  2. La mayoría de las operaciones requieren conexión con el repositorio central.
  3. Una versión contiene la copia del directorio completo.
- Distribuidos: Este sistema descentralizado [Figura 2.4] permite que muchos desarrolladores trabajen sobre un mismo proyecto, sin necesidad de requerir una red común. Cada usuario tiene una copia local de toda la información de

un repositorio, la cual puede modificar cualquier archivo y sincronizar los aportes cuando desee, haciendo uso de una conexión a internet.

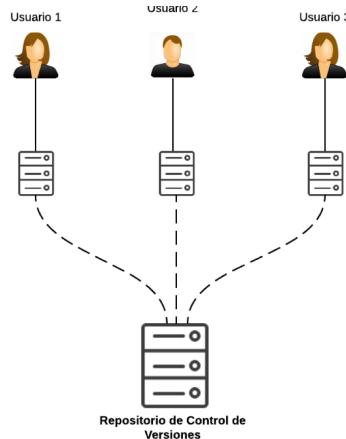


Figura 2.4: Esquema simplificado de un sistema de control de versiones distribuido

#### Principales características de los sistemas distribuidos

1. Cada desarrollador tiene su propia copia local.
2. Capacidad de regresar a una versión anterior de manera local.
3. Los repositorios distribuidos funcionan como respaldo.
4. Es posible realizar cambios sobre cualquier archivo del repositorio.
5. Unión de los cambios.

## 2.5. Marco Técnologico

# **Capítulo 3**

## **Estado del arte**

### **3.1. Sistemas, técnicas y/o métodos actuales**

La industria se queja que los actuales planes de estudios de las universidades fallan en cuestiones prácticas del desarrollo real de software, muchas empresas esperan que los graduados en ciencias de la computación sean productivos, sin necesidad de entrenamiento adicional [14, 5, 17]. Los alumnos no están siendo preparados para enfrentar el desarrollo a gran escala, donde es requerido el trabajo en equipo, habilidades de comunicación oral/escrita entre otras competencias.

Para enfrentar esta situación las infraestructuras colaborativas son utilizadas en la enseñanza de diferentes aspectos del desarrollo de software. En [9] el análisis de requerimientos se practica haciendo uso de equipos y entornos colaborativos. Varias universidades han creado entornos más realistas para la enseñanza de aspectos del desarrollo colaborativo de software [10] en [7] se analizan las herramientas y sus perspectivas de uso. La naturaleza colaborativa y los principios claves del código libre son discutidos en [13]. También algunas Universidades han enseñado la Ingeniería de software usando ambientes de aprendizaje [8] para ayudar a los alumnos en temas básicos y avanzados de las ciencias de la computación. En [11] son usadas herramientas de código libre con el fin de brindar experiencias más realistas a los alumnos. Existen comunidades tales como Teaching Open Source [2], dedicado a promover la investigación y los métodos colaborativos.

“The software Factory”: este concepto es aplicado por algunas Universidades, que a través de infraestructura y herramientas de trabajo colaborativo, proponen a sus alumnos interactuar con su entorno, solucionando problemas reales de la industria, bajo la guía de la facultad [10]. The Catholic University of America (CUA) es una pequeña institución en el corazón de Washington D.C. que utiliza el concepto de ‘software factory’ para enseñar a sus alumnos a desarrollar software. El proceso se modela en la [Figura 3.1]. A cada nivel, rol y responsabilidad le corresponde un color.

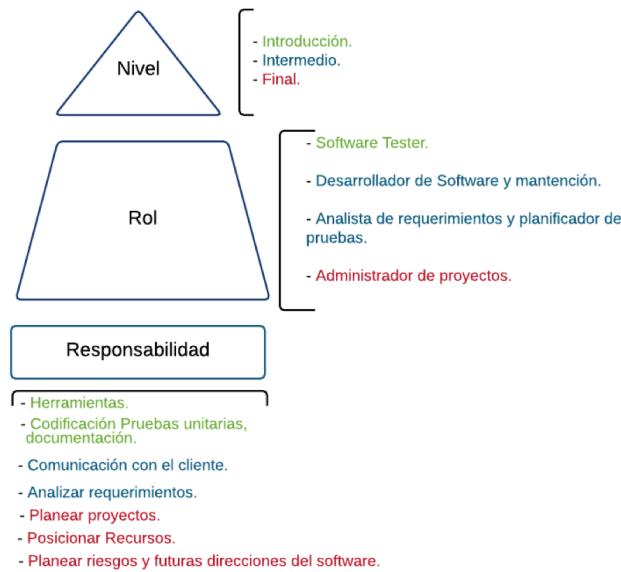


Figura 3.1: Abstracción del plan de estudio de la Universidad

- Semestre 1 - Desarrollo de software y herramientas: En esta primera etapa se introduce a los alumnos al desarrollo de software y al conjunto de herramientas disponibles.
- Semestre 2 - Software System Testing: En este curso los alumnos son puestos en el rol de Software testers, con todas las responsabilidades que esto implica.
- Semestre 3 - Desarrollo de Software y Mantención: En este curso los alumnos son puestos en el rol de desarrolladores con responsabilidades tales como escribir Software, realizar pruebas unitarias, documentar, etc.
- Semestre 4 - Desarrollo de Software y mantención: Este curso es la continuación de S3.
- Semestre 5 - Requerimientos y plan de pruebas: En este nivel los alumnos son puestos en el rol de analistas de requerimientos de sistema y test planners. Algunas de las responsabilidades aquí son: Comunicarse con el cliente, analizar los requerimientos del cliente, documentación, etc.
- Semestre 6 - Software Tesing: En este nivel los alumnos son puestos en el rol de diseñador de software.
- Semestre 7 - Administración de proyectos: En este curso los alumnos son puestos en el rol de administradores de proyectos. Las responsabilidades del alumno incluyen: Planear proyectos, posicionar recursos, estimar riesgos y planear futuras

direcciones del software.

### Análisis crítico

Con el uso explosivo uso de la internet y la permeabilidad de la tecnología en prácticamente cada aspecto de nuestras vidas, la necesidad de personal capacitado para construir software de calidad es aparente. La industria desea que las instituciones educacionales prepararen a los estudiantes en el uso de las últimas tecnologías. Pero las Universidades hacen énfasis en desarrollar habilidades de largo plazo. Ya que enfocarse en satisfacer las demandas de la industria significaría comprometer la calidad de los conocimientos del alumno. El enfoque pedagógico de las Universidades evita que al volverse obsoletas las tecnologías, ocurra lo mismo con los conocimientos del estudiante. El problema de esto, son los aspectos prácticos del desarrollo de software, usualmente no se transfiere la suficiente experiencia, que permita apreciar las implicancias de las decisiones que toman los alumnos durante el ciclo de vida del software.

# **Capítulo 4**

## **Definición del problema y solución propuesta**

### **4.1. Contexto**

Producto del rápido avance tecnológico en la industria, existe una gran demanda por especialistas preparados en el área de las ciencias de la computación. En el estudio de esta materia, el aprendizaje de la programación incluye el uso de lenguajes y sintaxis que permiten la implementación de modelos abstractos codificados (datos, control). Por lo que aprender y estudiar estas ideas, es a menudo, un desafío. Los procesos asociados al desarrollo de sistemas son una tarea compuesta de aspectos teóricos y cognitivos, en donde son necesarias tanto habilidades en programación como la aplicación de enfoques sistemáticos que guíen el desarrollo software, con el fin de asegurar que las operaciones participantes de los procesos computarizados sean usables, confiables y mantenibles. El perfil de egreso de la carrera de Ingeniería Civil en Informática de la Universidad de Valparaíso caracteriza a los futuros profesionales como Integradores de las tecnologías [9]. El desarrollo de estas competencias se adquiere conforme el alumno avanza por las etapas que componen el plan de estudio. En una primera fase, el alumno resuelve tareas específicas mediante secuencias de instrucciones o algoritmos. En una etapa intermedia, asocia y comunica estos programas con el fin de ejecutar un conjunto de coordinadas funciones, tareas o actividades que satisfagan requerimientos de pequeños proyectos. Finalmente el alumno es capaz de ejercer actividades de planificación, control y gestión de proyectos.

Los Alumnos aplican el aprendizaje acomodando los conceptos a un modelo que basa sus operaciones en un ambiente local. Este posee las herramientas suficientes para el desarrollo de cada una de las prácticas que requieran de estructura computacional para llevar a cabo sus funciones [Figura 4.1].

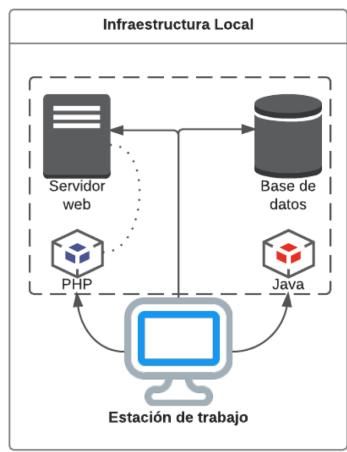


Figura 4.1: Modelo de la situación actual

Consecuencia de este esquema, el desarrollo de las actividades, es reducido al trabajo individual o de pequeños grupos, con la difícil tarea de coordinar aportes y gestionar los cambios cuando el equipo crece o se dispersa. Por otro lado, este modelo no representa adecuadamente la realidad del desarrollo de software, que existe hoy, en las organizaciones, donde los futuros Ingenieros deberán ser capaces de Integrar tecnologías en un contexto dinámico, hoy en día los productos son altamente susceptibles al cambio y es común tener a múltiples desarrolladores, en diferentes labores, accediendo concurrentemente a un mismo proyecto. Dado este escenario, es de vital importancia que los alumnos puedan aplicar los contenidos de las asignaturas usando herramientas y ambientes similares a las organizaciones, con el fin de otorgar experiencias que eventualmente puedan transferir a situaciones reales.

## 4.2. Solución propuesta

Implementación de una infraestructura [Figura 4.2]. Que proporcione recursos para el uso de tecnologías y metodologías de apoyo a los alumnos en la aplicación de las materias. Haciendo uso de herramientas estándar que faciliten la enseñanza-aprendizaje. Construyendo en el alumno, desde una etapa temprana de su formación, una visión de las actividades asociadas al ciclo de vida del software.

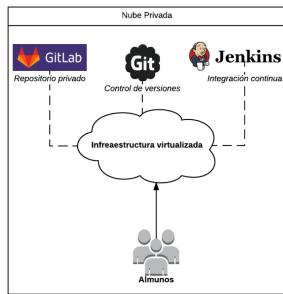


Figura 4.2: Arquitectura de alto nivel de la solución

### 4.3. Naturaleza del cambio

Actualmente la Escuela de Ingeniería Civil en Informática no posee una plataforma que fomente el trabajo en equipo en ambientes similares a los que existen en la industria, donde los flujos de trabajo, uso de las tecnologías y recursos, están definidos y estructurados acorde a metodologías modernas [15].

### 4.4. Objetivos

En esta sección se da a conocer el objetivo general del trabajo de título, junto con los objetivos específicos que permiten alcanzarlo.

#### 4.4.1. Objetivo general

- Implementar una infraestructura que permita el uso de herramientas que apoyen el desarrollo de software colaborativo en la Carrera de Ingeniería Civil en Informática de la Universidad de Valparaíso, con fines docentes, para integración de prácticas entre asignaturas.

#### 4.4.2. Objetivos específicos

- Especificar requerimientos y requisitos del sistema.
- Definir características del hardware disponible.
- Diseñar modelo de la solución.
- Definir Tecnologías adecuadas para el diseño de la infraestructura.

- Realizar diagramado acorde a la etapa de desarrollo e implementación del sistema.
- Definir políticas y flujos de trabajo.
- Determinar el funcionamiento constante y correcto de las funcionalidades del sistema.

## 4.5. Aporte

Como consecuencia de la implementación de este sistema. Se dispondrá de una Herramienta para conformar ambientes (redes, servidores, otros servicios) en donde alumnos, haciendo uso de tecnologías concretas, almacenen, administren y/o comparten distintas versiones de uno o varios proyectos haciendo uso de avances y recursos tecnológicos que hoy tienen raíces en la red.

## 4.6. Metodología de investigación y desarrollo

### 4.6.1. Investigación

Para la búsqueda de conceptos, tecnologías, trabajos y/o Investigaciones relacionadas a este tema, fueron utilizadas bibliotecas digitales [IEEE, ACM] en donde es posible descargar documentos haciendo uso de la red de la Universidad de Valparaíso.

### 4.6.2. Desarrollo

Metodología cascada, integrando el principio de trabajo en cadena, de manera de obtener resultados continuos en base a cada incremento [Figura 4.3].

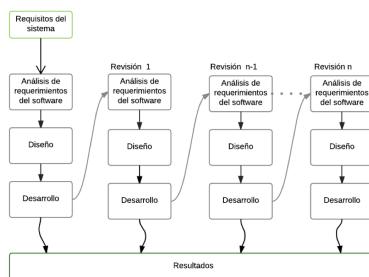


Figura 4.3: Metodología de trabajo

# **Capítulo 5**

## **Análisis**

En el siguiente capítulo se presenta el análisis realizado a través de investigaciones y reuniones en conjunto con el profesor guía del presente trabajo de título.

### **5.1. Especificación de requerimientos**

#### **5.1.1. Requerimientos funcionales**

- RF1: Servidor dedicado al ramo de programación en lenguaje C.
- RF2: El servidor en C debe tener los programas Make, gcc y vim instalados.
- RF3: Creación y configuración de la integración continua, en un servidor dedicado, haciendo uso de la herramienta Jenkins.
- RF4: Estructura operacional para programación en lenguaje Java.
- RF5: Estructura operacional para programación en lenguaje .NET.
- RF6: Estructura operacional para programación en lenguaje PHP.
- RF7: Acceso a un sistema de control de versiones.
- RF8: Servidores para el despliegue del software.

#### **5.1.2. Requerimientos no funcionales**

- RNF: El servidor debe estar disponible el 95 porciento del tiempo (Alta disponibilidad).

## 5.2. Diagrama de casos de uso (DCU)

El diagrama de casos de uso representa las operaciones que los usuarios realizan en el sistema, junto con las relaciones existentes entre cada una de las tareas [Figura 5.1].

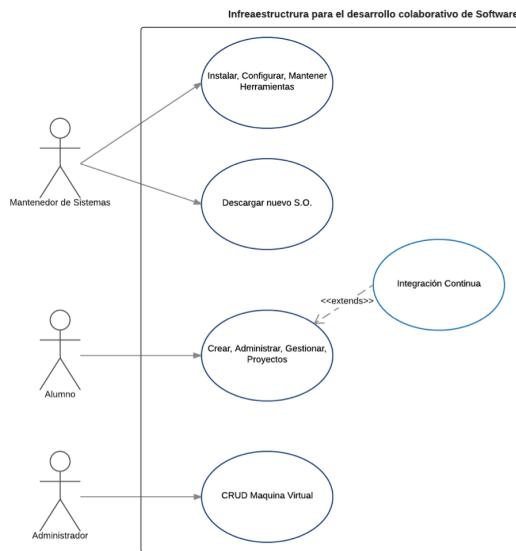


Figura 5.1: Diagrama de casos de uso

# **Capítulo 6**

## **Diseño**

En este Capítulo se presentan los niveles que comprenden el sistema a implementar:

- Nivel Físico.
- Nivel Virtual.
- Nivel Conceptual.
- Herramientas.

### **6.1. Diseño arquitectónico**

La solución tecnológica presentada en la [Figura 6.1] posee un nivel físico, que corresponde, a los recursos base de esta implementación. El nivel virtual brinda una mapeo dinámico del nivel Físico haciendo uso de un hipervisor. El nivel conceptual corresponde a la metodología que ordena el proceso de desarrollo de software, en este caso, se trata de un enfoque ágil. Finalmente se encuentra el conjunto de herramientas para facilitar el desarrollo colaborativo de software. A continuación en la [Figura 6.2] se presenta la solución general, donde los usuarios a través de herramientas podrán resolver actividades académicas en ambientes colaborativos:

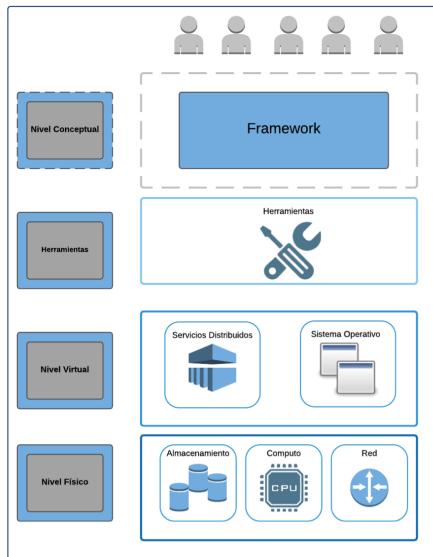


Figura 6.1: Diagrama general de la solución

## 6.2. Descomposición arquitectural

### 6.2.1. Nivel Físico

Este nivel encapsula el lote de recursos físicos disponibles para ser administrados y reorganizados según las necesidades que se presenten [Figura 6.3].

- CPU.
- Almacenamiento.
- Red.

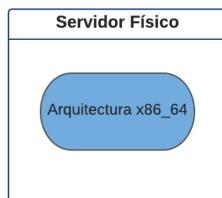


Figura 6.2: Nivel Físico

### 6.2.2. Nivel virtual

La implementación de esta capa tiene como finalidad particionar los recursos físicos, para ejecutar múltiples instancias de servidores. Los dos acercamientos típicamente utilizados son la arquitectura Hypervisor y Hosted. La arquitectura Hosted ofrece servicios de particionamiento un nivel por encima del sistema operativo estándar de esa maquina. Esto quiere decir, existe un bloque adicional entre el metal y la virtualización [Figura 6.4]. Por otro lado, la implementación B elimina este nivel, por medio de la instalación de un sistema operativo especial o hipervisor. La relación de adyacencia proporciona un acceso a los recursos de hardware más eficiente con el fin de ofrecer mayor robustez, escalabilidad y desempeño de las maquinas virtuales.

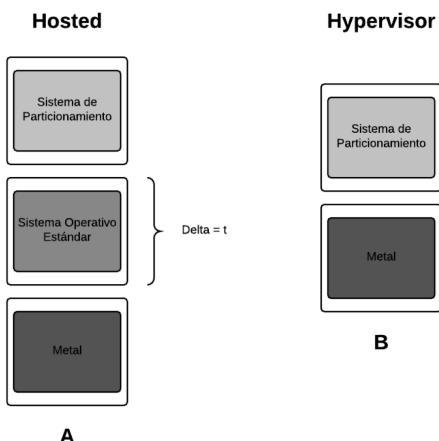


Figura 6.3: Arquitectura hosted v/s hypervisor

La figura 6.5 que se presenta a continuación ofrece mayor detalle del sistema de particionamiento a usar.

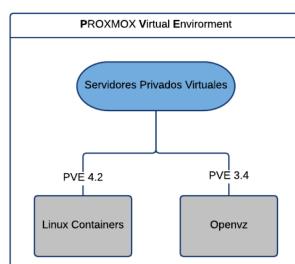


Figura 6.4: Esquema del nivel virtual

Proxmox VE esta basado en debían (GNU/Linux). Este sistema ofrece dos tipos de solución para la virtualización a nivel de sistema operativo. La tecnologia de virtualizacion cambia dependiendo de la version de Proxmox.

### Open Virtuozzo (OpenVZ)

Es una tecnología de virtualización a nivel de sistema operativo para Linux. Permite a un servidor físico ejecutar múltiples instancias de servidores. El núcleo OpenVZ provee aislamiento, administración de recursos y tolerancia a fallos. OpenVZ se encuentra disponible en la versión 3.4 del sistema operativo Proxmox Virtual Environment.

### Container-Based Virtualization (LXC)

Similar a la tecnología de virtualización OpenVZ, Linux-VServer y de otros sistemas operativos como FreeBSD jail y Solaris Containers. Linux Containers permite la priorización de recursos (CPU, Memoria, Entrada/Salida, Red, Etc) sin necesidad de iniciar la máquina virtual. Esta herramienta combina Kernel cgroups para namespaces aislados que proveen ambientes aislados y limpios para la ejecutar aplicaciones. LXC se encuentra disponible en la versión 4.2 del sistema operativo Proxmox.

## 6.3. Nivel conceptual

El objetivo principal de las metodologías ágiles es producir y entregar software funcional de forma rápida. FDD es uno de los seis acercamientos que fueron presentados con el nacimiento del manifesto ágil, este fue creado por 17 individuos en 2001. FDD es un modelo iterativo-incremental que consiste en la aplicación de cinco etapas [Figura 6.6]. Las últimas dos etapas son repetidas hasta que el producto es completado, luego, el proyecto es formalmente cerrado. Este Framework descompone las actividades asociadas al desarrollo de un sistema en partes pequeñas que puedan ser resueltas en no más de dos semanas. Estas piezas son llamadas ‘Características’, las cuales son tangibles, verificables por el usuario, aportan valor al cliente por sí solas y tienen un tiempo de desarrollo que no excede las dos semanas. Las tareas son realizadas por los integrantes del proyecto, estos desempeñan distintos roles dentro de la organización, jerarquizados, que a su vez le corresponden una conjunto de responsabilidades [Figura 6.12].

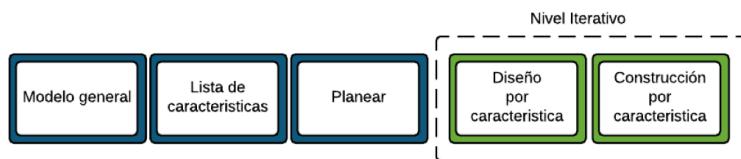


Figura 6.5: Diagrama de bloques del proceso

### 6.3.1. Modelo general

Los expertos en el dominio realizan un recorrido de alto nivel del alcance del sistema y su contexto.

#### Criterio de inicio de esta etapa

Ya fueron seleccionados los expertos de dominio, programador jefe y arquitecto adecuados para este proyecto.

#### Tareas de esta etapa

- Formar un equipo para el modelado.
- Estudio de documentación asociada al dominio.
- Entregar información general del área de dominio a modelar.
- Formar grupos de modelado mas pequeños.

#### Criterio para el cierre de esta etapa

El equipo de modelado debe producir un modelo en conformidad con el arquitecto jefe del proyecto.

#### Salidas

- Diagramas de clases.
- Diagrama de secuencia, si corresponde.

#### Diagrama de flujo de trabajo

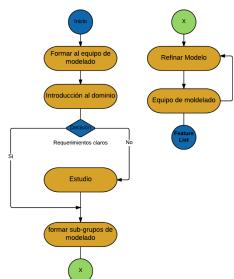


Figura 6.6: Diagrama de flujo de la primera etapa.

### 6.3.2. Lista de características

El producto a desarrollar es descompuesto en una lista de actividades, esta lista corresponde a pequeñas piezas que aportan valor al cliente y son expresadas de la forma Acción-Resultado-Objeto. Las funcionalidades que son parte del dominio son descompuestas y distribuidas en las diferentes áreas. Por lo que cada área posee un numero acotado de ‘Características’. Como resultado, se tiene un listado jerarquizado de funcionalidades.

#### Criterio para el inicio de esta etapa

- El equipo de modelamiento a completado exitosamente su trabajo.

#### Tareas para esta etapa

- Formar el equipo para este proceso.
- Construir la lista de características.

#### Roles participantes de esta etapa.

Administrador de proyecto, administrador de desarrollo.

#### Criterio para el cierre de esta etapa

Como salida se debe tener una lista de características aprobada por administrador de proyecto y desarrollo.

#### Salida

- Lista general de características por área.

#### Diagrama de flujo de trabajo



Figura 6.7: Diagrama de flujo de la segunda etapa.

### 6.3.3. Planear por característica

En esta etapa se planea el orden en que serán implementadas las características. Para esto nos basamos en las dependencias, complejidad e importancia de cada una dentro del sistema a construir.

#### Criterio para el inicio de esta etapa

- La lista de características a sido completada con éxito.

#### Tareas de esta etapa

- Formar el equipo para este proceso.
- Determinar la secuencia de desarrollo.
- Asignar un grupo de características al Programador Jefe.
- Asignar clases a los desarrolladores.

#### Roles participantes de esta etapa

Administrador de proyecto, Administrador de desarrollo, Programador jefe.

#### Criterio para el cierre de esta etapa

El proceso es cerrado en conformidad con el administrador de proyecto y desarrollo.

#### Salida

- Lista de características con sus respectivas fechas.
- Programador jefe asignado a la lista de características.
- Lista de clases para los programadores.

### Diagrama de flujo de trabajo

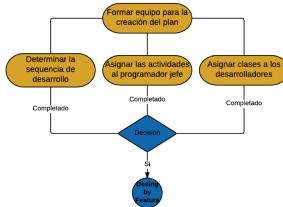


Figura 6.8: Diagrama de flujo de la tercera etapa

#### 6.3.4. Diseño por característica

Por cada característica es creado un paquete de diseño, el programador jefe selecciona un pequeño grupo de características, asignadas previamente, que luego serán desarrolladas.

##### Criterio para el inicio de esta etapa

El plan a sido completado exitosamente.

##### Tareas de esta etapa

- Formar al equipo de diseño.
- Desarrollar la secuencia de diagramas pertinentes a esta etapa.
- Refinar el modelo de objetos.
- Inspección del diseño.

##### Roles participantes de esta etapa

Programador jefe, equipo de desarrollo, experto de dominio.

##### Criterio para el fin de esta etapa

Creación exitosa y verificada del o los paquetes de diseño.

##### Salida

- Diagrama de secuencia.
- Modelo de objetos con sus respectivos métodos, atributos y clases.

### Diagrama de flujo de trabajo



Figura 6.9: Diagrama de flujo de la cuarta etapa

#### 6.3.5. Construcción de las características

Trabajando desde los paquetes de diseño, los programadores implementan los elementos necesarios para sus clases. El código desarrollado es probado e inspeccionado.

##### Criterio para el inicio de esta etapa

La etapa de diseño a sido completada con éxito. Los paquetes de diseños han sido inspeccionados correctamente.

##### Tareas de esta etapa

- Implementar clases y métodos.
- Realizar inspección del código.
- Pruebas unitarias.
- Aceptación.
- Verificación.

##### Roles participantes de esta etapa

Programador, Tester.

### Criterio para el fin de esta etapa

Desarrollo de una o mas características.

### Diagrama de flujo de trabajo

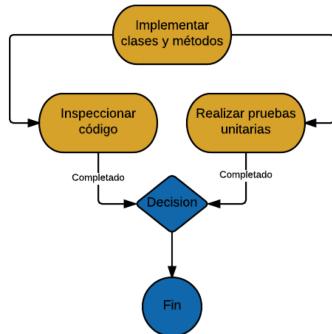


Figura 6.10: Diagrama de flujo de la quinta etapa.

## 6.4. Roles y responsabilidades

### Administrador de proyecto (AP)

Es el encargado de corroborar el progreso del desarrollo del producto, administrar recursos, presupuestos y equipamiento.

### Arquitecto jefe (AJ)

Esta persona posee habilidades técnicas y de modelamiento, es el responsable de la construcción del modelo general del producto, se encarga de coordinar sesiones en donde el equipo colabora en el diseño del sistema.

### Administrador de desarrollo (AD)

Es responsable de liderar las actividades diarias de desarrollo. Son requeridas buenas habilidades técnicas. El AD deberá resolver conflictos que el programador jefe no puede abarcar.

### Programador jefe (PJ)

Este rol corresponde a un desarrollador con experiencia, participando en el levantamiento de requerimientos de alto nivel, analizando y diseñando actividades que serán desarrolladas por pequeños grupos de trabajo.

### Administrador de entregas (AE)

Se asegura que el programador jefe entregue reportes de progreso cada semana. Estas son informadas directamente al administrador de proyecto.

### Programador (CO)

Es un miembro de un pequeño grupo de trabajo liderado por el jefe de proyecto. Esta persona diseña, codifica, prueba y documenta las características del sistema en construcción.

### Experto de dominio(ED)

Corresponde a los usuarios, auspiciadores, analistas de negocio. Los desarrolladores confían en ellos, ya que debido a su conocimiento se asegura la entrega de un producto correcto.

### Diagrama de roles

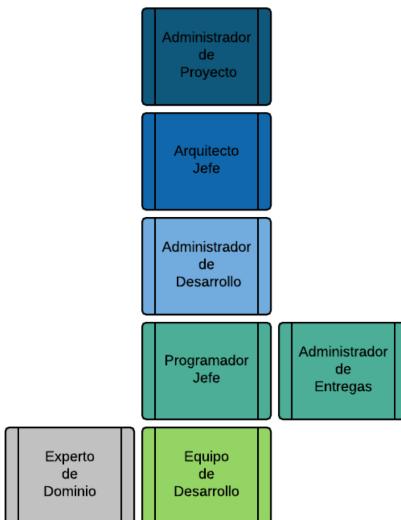


Figura 6.11: Modelo jerárquico de roles

# Capítulo 7

## Herramientas

### Herramienta para la administración de proyectos - Gitlab

- Planificación de sprint.
- Creación de proyectos.
- Distribuir tareas.

### Estrategia de bracheo

El flujo de trabajo se organiza en dos ramas principales, Master y Develop. Ademas existen tres ramas auxiliares Feature, reléase, hotfix [Figura 7.1].



Figura 7.1: Ejemplo de funcionamiento

- Master: Cualquier commit que pongamos en esta rama debe estar listo para ir a producción.
- Develop: Aquí se encuentra todo el código asociada a una versión planificada del proyecto.
- Feature: Se utilizan para desarrollar nuevas características de la aplicación que, una vez terminadas, se incorporan a la rama develop.

- Release: se utilizan para preparar el siguiente código en producción. En estas ramas se hacen los últimos ajustes y se corrigen los últimos bugs antes de pasar el código a producción incorporándolo a la rama master.
- Hotfix: Esas ramas se utilizan para corregir errores y bugs en el código en producción (no planificados).

## 7.1. Comunicación - Slack

Esta es una herramienta de mensajería instantánea para equipos, permite tener todas las comunicaciones en un mismo lugar.

### Principales características

- Separación de las comunicaciones en canales, que puede ser públicos o privados.
- Integración con GIT.
- Facilidad para compartir archivos.
- Multiplataforma.

## 7.2. Repositorio de contenidos - GIT

Siendo unas de las mas populares tecnologías de código libre para el control de versiones, Git y Mercurial son muy similares en funcionamiento. Cada archivo modificado es versionado. Para mayor eficiencia estos sistemas de control de versiones no copian archivos sin modificar, en su lugar crean una referencia que apunta a otra versión

### 7.3. Gestión del flujo de trabajo - SourceTree

Source tree es un cliente que permite el uso y seguimiento de repositorios GIT, esta herramienta permite gestionar nuestros proyectos remotos o locales, haciendo uso de una interfaz grafica, tal como podemos apreciar en la [Figura 7.2].



Figura 7.2: Ejemplo de funcionamiento

### 7.4. Integración continua - Jenkins

La integración continua es una práctica del desarrollo de software que nace con la aparición de las metodologías ágiles, esta tecnología permite a los miembros de un equipo de trabajo integrar constantemente el código fuente de uno o varios proyectos. Una build es realizada periódicamente para comprobar los cambios, al código fuente, funcionan correctamente antes de ser puestos en los ambientes finales de producción. Jenkins es un servidor de Integración Continua, gratuito y de código libre basado en tareas, donde se indican las actividades a realizar en una determinada Build. Esta herramienta se enlaza a todo el proceso de desarrollo cuando es importante inspeccionar el código y visualizar resultados de las pruebas unitarias.

### 7.5. Diagrama final de la solución

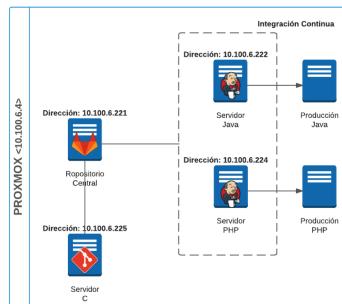


Figura 7.3: Modelo de la solución

## 7.6. Diseño de pruebas

En esta sección se muestra la elaboración del tipo de prueba que se utilizará para evaluar la infraestructura para el desarrollo colaborativo de software.

### 7.6.1. Pruebas unitarias

Las pruebas unitarias sirven para aislar módulos del sistema para demostrar su funcionamiento ante un escenario predefinido. Dado que este sistema corresponde a la integración de diferentes herramientas de código libre, las pruebas serán del tipo caja negra, se especificará de forma previa la entrada y salida de la función a testear.

Función a probar	Caso de prueba
Entrada válida	
Salida esperada	
Entrada inválida	
Salida esperada	
Resultado:	

Tabla 7.1: Modelo general de pruebas.

# Capítulo 8

## Implementación

En este capítulo se presenta la implementación de cada uno de los niveles de la infraestructura de apoyo al desarrollo colaborativo de software con fines docentes, para la integración de prácticas entre asignaturas.

### 8.1. Hardware de desarrollo

#### Hardware adicional

- **Nombre:** Macbook Pro, equipo de trabajo del alumno.
- **Sistema Operativo:** OS X El Capitan.
- **CPU:** 2,5 GHz Intel Core i5.
- **Memoria RAM:** 8000MB.
- **HDD:** SATA 2, 500GB.

#### Nivel Físico - Especificaciones del equipo

- **Nombre:** Nereo, Servidor físico proporcionado por la escuela de Ingeniería Civil en Informática.
- **Sistema Operativo:** Proxmox Virtual Environment v3.4, (Debian Wheezy 7.8).
- **CPU:** Intel(R) Xeon(R) CPU E5504, 2GHz.
- **Memoria RAM:** 4100MB.

- **HDD:** Hitachi SATA HDS721032CLA362 JPFOA39C, 320GB.
- **DVD:** DV-28E-V.

## 8.2. Herramientas para el desarrollo

### 8.2.1. Nivel virtual - Proxmox Virtual Enviorment

Para esta parte de la implementación fue usado el sistema operativo, de código libre, Proxmox Virtual Enviorment v3.4, como solución de virtualización. Proxmox es una herramienta que soporta dos tipos de virtualización, Kernel-Based Virtual Machine (KVM) y Container-Based Virtualization.

OpenVZ permite a un servidor físico ejecutar múltiples instancias aisladas de sistemas operativos, llamados contenedores. Este nucleo modificado provee virtualización, aislamiento y administración de los recursos. Cada maquina virtual posee su propio sistema de archivos, usuarios, red, dispositivos y árbol de procesos. La **administración de los recursos** funciona en cuatro niveles, estos recursos pueden ser modificados en tiempo de ejecución por lo que no es necesario reiniciar los servidores.

1. **Cuota de disco:** Cada contenedor tiene su propia cuota de disco en bloques de disco e innodes.
2. **Planificador de CPU:** Un primer nivel decide a que contenedor le asignara tiempo de CPU, esto en base a los valores de CPU de cada contenedor, en un segundo nivel, decide que proceso ejecutar en ese contenedor.
3. **Planificador de entrada y salida:** Basado en la asignación de prioridades de entrada y salida en cada contenedor. El planificador distribuye el ancho de banda disponible acorde a esas prioridades.
4. **User Beancounters:** Este nivel asegura que los recursos del sistema no sean monopolizados por un solo contenedor.

Se configuro la red de Proxmox dentro del segmento de direcciones de la Facultad de Ingeniería de la Universidad de Valparaíso [Figura 9.1].

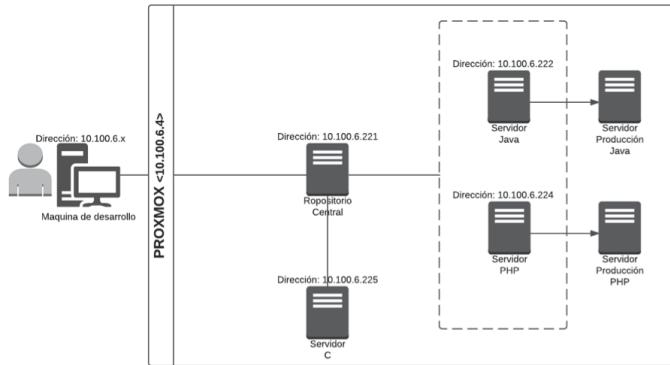


Figura 8.1: Modelo de red del sistema

### 8.2.2. Servidores virtuales - Estructura operacional

#### Repositorio central

Servidor de administración de repositorios, al cual se conectan los servidores de integración continua y el servidor de desarrollo en C.

- **Recursos asignados:** RAM @1500MB, HDD 80GB.
- **Red:** Veth device.
- **Sistema operativo:** Ubuntu 14.04 64bits.
- **Paquetes instalados:** Gitlab 8.10.

#### Servidor Java

Maquina de integración continua para el desarrollo en lenguaje Java.

- **Recursos asignados:** RAM @1024MB, HDD 25GB.
- **Red:** Veth device.
- **Dirección IP:** 10.100.6.222.
- **Sistema operativo:** Debian 7.
- **Paquetes instalados:** Jdk 1.8, Maven 3.3, Git 2.7, Jenkins 2.7.

### Servidor PHP

Maquina de integración continua para el desarrollo en lenguaje PHP.

- **Recursos asignados:** RAM @1024MB, HDD 25GB.
- **Red:** Veth device.
- **Dirección IP:** 10.100.6.224.
- **Sistema operativo:** Debian 7.
- **Paquetes instalados:** PHP 5.6, Composer 1.2, Git 2.7, Jenkins 2.7.

### Servidor C

Maquina que permite el desarrollo de programas en lenguaje C.

- **Recursos asignados:** RAM @650MB, HDD 25GB.
- **Red:** Veth device.
- **Dirección IP:** 10.100.6.225.
- **Sistema operativo:** Debian 7.
- **Paquetes instalados:** Git 2.7, Gcc 6.2, Make.

### Producción Java

Servidor de aplicación con la ultima versión del código funcional.

- **Recursos asignados:** RAM @650MB, HDD 25GB.
- **Red:** Veth device.
- **Dirección IP:** 10.100.6.226.
- **Sistema operativo:** Debian 7.
- **Paquetes instalados:** JDK 1.8.

## Producción PHP

Servidor de aplicación con la ultima versión del código funcional.

- **Recursos asignados:** RAM @650MB, HDD 25GB.
- **Red:** Veth device.
- **Dirección IP:** 10.100.6.227.
- **Sistema operativo:** Debian 7.
- **Paquetes instalados:** PHP 5.6.

### 8.2.3. Herramientas del sistema - Versión

Git 2.7: Es un sistema de control de versiones distribuido de código abierto, utilizado en el desarrollo de software y otras tareas. Como sistema distribuido apunta a la velocidad , integración de los datos y soporte al flujo de trabajo distribuido no lineal.

Gitlab 8.11: Es un administrador de repositorios de código libre en donde es posible, tener un seguimiento de las versiones de los repositorios Git. Con esta herramienta es posible el seguimiento de problemas, revisión de código y registro de actividad.

Jenkins 2.19: Es una herramienta, de código libre, para integración continua escrita en Java. Jenkins ejecuta pruebas y construye el código almacenado en el repositorio central.

PHP 5.6: Es un lenguaje de scripting utilizado para construir aplicaciones en el servidor PHP.

PHPunit 5.5.4: Framework para la ejecución de pruebas unitarias para desarrollos en php.

JUnit 4.12: Framework para la ejecución de pruebas unitarias en entornos de desarrollo Java.

Composer 1.2: Es una herramienta para la administración de dependencias en PHP, permite declarar en el archivo composer.json, las librerías que los proyectos necesitan instalar o actualizar de forma automática.

Jdk 1.8: Plataforma Java, que permite la construcción de aplicaciones en el servidor de desarrollo Java.

Maven 3.3.9: Es una herramienta para la administración de dependencias en Java, permite declarar las librerías que un proyecto necesita instalar de forma automática.

Gcc 6.2: Herramienta de código libre que posee un conjunto de compiladores para el lenguaje C.

# Capítulo 9

## Vistas

### 9.1. Proxmox Virtual Environment

Pantalla de acceso a Proxmox VE 3.4 [Figura 10.1]

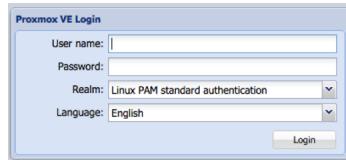


Figura 9.1: Login Proxmox VE

En la pantalla principal tenemos acceso a la lista de servidores virtuales del nodo pve y pestañas que facilitan la configuración, acceso a información del sistema y graficas del funcionamiento global [Figura 10.2].



Figura 9.2: Login Proxmox VE

Los botones create CT/VM permiten crear contenedores y maquinas virtuales almacenadas en el disco del nodo (pve). Desde el disco local podemos subir imágenes

y plantillas de otros sistemas operativos [Figura 10.3].

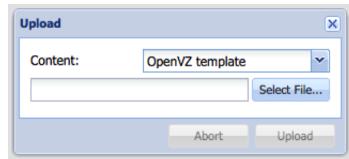


Figura 9.3: Pantalla de carga de imágenes y plantillas

## 9.2. Gitlab

Una vez registrados, podemos acceder a la pagina de inicio, en donde es posible seleccionar proyectos o crear nuevos. El menú lateral ‘Dashboard’ nos permite navegar entre las principales funciones del sistema [Figura 10.4].

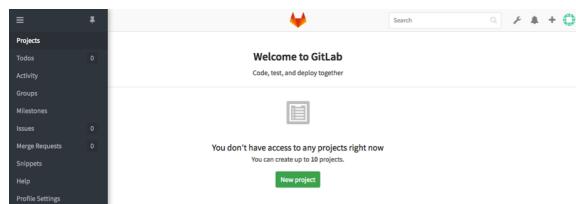


Figura 9.4: Pantalla de inicio Gitlab

Al presionar sobre el botón ‘New project’ se tiene la siguiente vista [Figura 10.5], desde aquí podemos importar repositorios o crear nuevos.

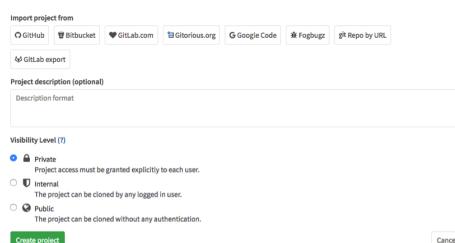


Figura 9.5: Creación nuevo proyecto

Dentro de un proyecto [Figura 10.6], es posible tener acceso a sus archivos y actividad.

The screenshot shows a GitHub repository interface. At the top, there are tabs for 'Files', 'Commits', 'Network', 'Compare', 'Branches', and 'Tags'. Below the tabs, there is a dropdown menu set to 'master' and a search bar containing 'php /'. A '+' button is also present. The main area displays a table of files with their last update times and commit messages:

Name	Last Update	Last Commit
proyecto.git	3 minutes ago	Primer clase
src	3 days ago	segundo Commit
test	2 days ago	validar
.gitignore	3 days ago	Primer Commit
composer.json	2 days ago	Primer commit
composer.lock	3 days ago	Primer Commit
readme	14 minutes ago	esto es mas apropiado

Below the table, the 'readme' file is expanded, showing its content: 'Integración'.

Figura 9.6: Creación nuevo proyecto

Desde la pestaña Network es posible acceder a la línea de tiempo del proyecto, aquí podemos volver a una versión anterior del trabajo o movernos a una más reciente [Figura 10.7].

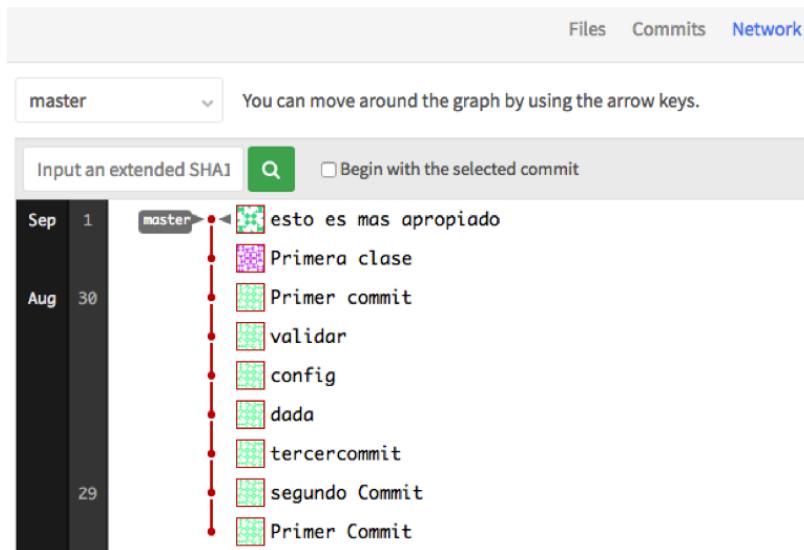


Figura 9.7: Creación nuevo proyecto

Para incorporar colaboradores a un proyecto, el creador debe agregar las llaves públicas de los miembros del equipo (idrsa.pub), en su perfil de usuario [Figura 10.8].

**SSH Keys**

SSH keys allow you to establish a secure connection between your computer and GitLab.

**Add an SSH key**

Before you can add an SSH key you need to [generate it](#).

**Key**

```
ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQCMfdjetqiz4ehlp9UPsoowJ9vXwDMWLC3AJYpZE1I0FLYKiuBHISIJYb8yg9eqk6dxkQj6xYaxikxOpnqn9NL0/
GU9u8Yg3qovRElpdwZsvwUQA3F6Tl3Y75mpuXIMRINnNNV11hWCEk9hoGyAkh7XWnm4wjNbbHz0u9KrC7caLDBMrKVcv0OjbLMd0Ge5E1ov/1or0
/q5J+YyHWxR6vANDTGaoEZejcUVlXWowS0oclqwHL/Zy73BxAJFuEFMTzpyTCbaO501sOZNGf1SYygXzVhxH3JF9u84Z1GEEFvzM6ZXy2lc8A4c4V
mWqRR6sguRZyJDIksLPT03Xuwff developer@nombredelalumno.local
```

**Title**

**Add key**

**Your SSH keys (1)**

	Nombre del alumno 32:c2:61:40:6c:b9:94:65:6c:9b:48:10:ae:fa:2b:60	created 2 minutes ago	
--	--	-----------------------	--

Figura 9.8: Creación nuevo proyecto

### 9.3. Jenkins

Para acceder a la pantalla de inicio de Jenkins debemos llenar los campos usuario y contraseña [Figura 10.9].

Usuario:

Contraseña:

Recuerdame en esta computadora

**Entrar**

Figura 9.9: Login Jenkins

La pantalla principal del sistema es como se muestra [Figura 10.10], desde aquí es posible configurar, tener acceso y crear nuevas tareas (Proyecto-php).

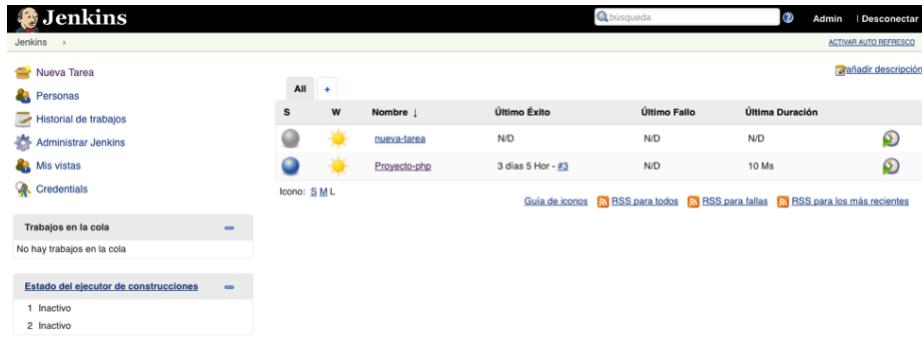


Figura 9.10: Pantalla de inicio Jenkins

Con el botón nueva tarea permite configurar la integración de nuevos proyectos [Figura 10.11], aquí es posible especificar el repositorio, triggers y pasos de construcción.

This screenshot shows the 'Configure New Item' page for a new project named 'nueva-tarea'. It includes fields for 'Project name' and 'Description', and a 'Plain text' preview area. Below these are several configuration options with checkboxes: 'Desechar ejecuciones antiguas', 'Esta ejecución debe parametrizarse', 'GitHub project', 'Throttle builds', 'Desactivar la ejecución', and 'Lanzar ejecuciones concurrentes en caso de ser necesario'. A 'Configurar el origen del código fuente' section follows, with 'Git' selected. It shows a 'Repository URL' field with an error message 'Please enter Git repository.' and a 'Guarda' button.

Figura 9.11: Crear/Configurar proyecto

Desde el menú build step podemos agregar comandos (probar, instalar, compilar) para que sean ejecutados por Jenkins [Figura 10.12].



Figura 9.12: Crear/Configurar proyecto

Una vez creado y configurado un proyecto es posible ejecutar manualmente su construcción. Se tiene una copia de los archivos de la ultima build dentro del espacio de trabajo de Jenkins, en el historial de tareas tiene el registro de las ultimas build creadas, azul indica que el proceso fue exitoso [Figura 10.13].



Figura 9.13: Construir proyecto

Al seleccionar una build del menú ‘Historia de tareas’ [Figura 10.14] podemos acceder a su salida de consola.



Figura 9.14: Salida de consola

El uso del sistema es como se muestra en la Figura, en el contexto de la utilización de la metodología ágil ‘Feature Driven Development’, se tendría el siguiente flujo de trabajo.

Primero un grupo de usuarios, trabaja en el desarrollo de una característica, con tiempo de implementación menor a dos días, estos desarrolladores hacen push (envían cambios al repositorio central) de sus últimas versiones. Luego el servidor de integración continua, Java o PHP, automáticamente ejecuta los test a las clases y construye una Build de la ultima versión del Master branch, esto a través de la configuración de un trigger, si todas las pruebas son superadas, el servidor de integración continua se encarga de mover los archivos al ambiente de producción que corresponda.

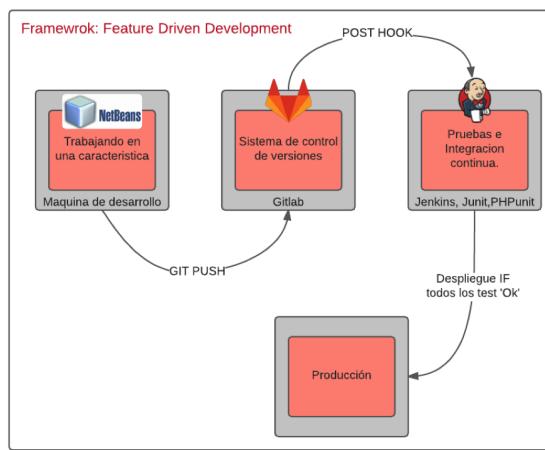


Figura 9.15: Flujo general del sistema

## 9.4. Navegabilidad del sistema

Cada uno de los sistemas navegables de esta solución son accedidos de manera independiente, a través del browser del usuario.

### 9.4.1. Proxmox

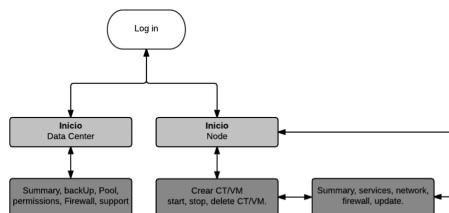


Figura 9.16: Diagrama de navegabilidad de la web GUI de Proxmox

#### 9.4.2. Gitlab

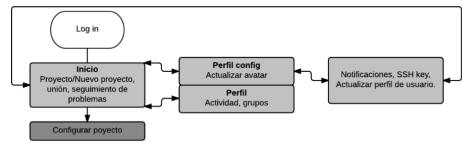


Figura 9.17: Diagrama de navegabilidad de la web GUI de Gitlab

#### 9.4.3. Jenkins

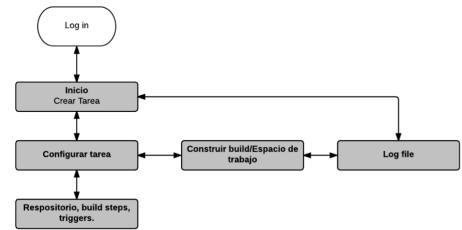


Figura 9.18: Diagrama de navegabilidad de la web GUI de Jenkins

# **Capítulo 10**

## **Pruebas**

### **Nivel de las pruebas**

- Pruebas de requerimientos: Primero se verifican los los requerimientos y luego se comprueba que el sistema los satisaga correctamente.
- Pruebas funcionales y de sistema: Usando tecnicas de pruebas del tipo caja negra, se busca determinar si la funcionalidad especificada en el requerimiento opera correctamente.
- Pruebas de validación: Mediante un ejemplo de funcionamiento se verifica el cumplimiento de las especificaciones del sistema y si este logra el propósito para el cual fue diseñado.

### **10.1. Pruebas de requerimientos**

Los resultados de la prueba de requerimientos se presentan en la Tabla 10.1, en donde se visualiza que se cumplen los item en su totalidad. La Tabla 10.1 muestra la lista de requerimientos, en donde es posible apreciar el cumplimiento de los Items.

Pregunta	SI	NO
¿El Documento se adhiere a los estandares?	X	
Claridad	X	
¿La especificación es clara y facil de entender?	X	
¿Se refleja el comportamiento del sistema?		X
¿Estan Especificados los requerimientos?	X	
¿Los requerimientos ayudan a cumplir el propposito del sistema?	X	

Tabla 10.1: Tabla de evaluación de requerimientos

Identificador	Estado
RF1	Completado
RF2	Completado
RF3	Completado
RF4	Completado
RF5	Completado
RF6	Completado
RF7	Completado
RF8	Completado
Total item:	8/8

Tabla 10.2: Lista de requerimientos

## 10.2. Pruebas funcionales

Las Pruebas fueron realizadas bajo el principio de caja negra [Figura 10.1], testers examinan en un alto nivel el diseño y los requerimientos para planear los casos de prueba, se busca asegurar funcionamiento del modulo. De esta manera se examinan las salidas del sistema, sin importar el funcionamiento interno del modulo inspeccionado.

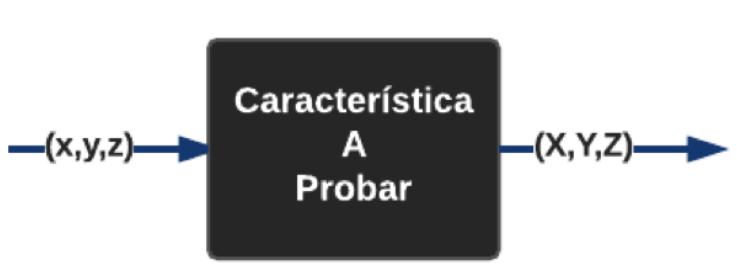


Figura 10.1: Diagrama de entrada y salida

En esta etapa de la evaluación se hara uso de una métrica [Tabla 10.2] que facilite identificar el origen de un resultado incorrecto o inesperado.

Resultado	Definición	Sigla
Error Humano	Acción Humana que produce un resultado incorrecto	M
Defecto	Paso, proceso o definición de datos incorrecto	D
Falla	la incapacidad de un componente para realizar la función requerida dentro de los requisitos de funcionamiento especificados.	F
Error	Diferencia entre el valor medido y el teorico	E
No aplica	Sin hallazgos	n/a

Tabla 10.3: Métricas de evaluación

### 10.2.1. Detalle de las pruebas funcionales

A continuación se presentan los diagramas de prueba y resultados de la evaluación del sistema.

#### Prueba funcional 01



Figura 10.2: Diagrama de entrada y salida

ID	Escenario	Caso de prueba	Entrada	Salida esperada
01.1	Iniciar sesión Gitlab	Ingreso de campos usuario y contraseña correctos	Usuario:'almunoiciuv', Password: alumnoiciuv	El Usuario Inicia Sesión

Tabla 10.4: Resumen de resultados

ID	Escenario	Caso de Prueba	Entrada	Salida Esperada
01.2	Iniciar sesión Gitlab	Ingreso del campo usuario incorrecto	Usuario:'almunouvici', Pass-word:'alumnouvici'	Error al iniciar sesión
01.2	Iniciar Sesión Gitlab	Ingreso del campo contraseña incorrecto	Usuario:'almunoiciuv', Pass-word:'alumnouvici'	Error al iniciar sesión

Tabla 10.5: Resumen de resultados

ID	Escenario	Caso de prueba	Entrada	Salida esperada
01.3	Iniciar sesión Gitlab	Campo Usuario Vacio	Password:alumnouvici	El sistema indica que el campo Usuario se encuentra vacio.
01.3	Iniciar sesión Gitlab	Campo Usuario Vacio	Usuario:alumnouvici :	El sistema indica que el campo Usuario se encuentra vacio.
01.3	Iniciar sesión Gitlab	Campo Usuario y Contraseña Vacio		El sistema indica que el campo Uuuario se encuentra vacio.

Tabla 10.6: Resumen de resultados

## Prueba funcional 02

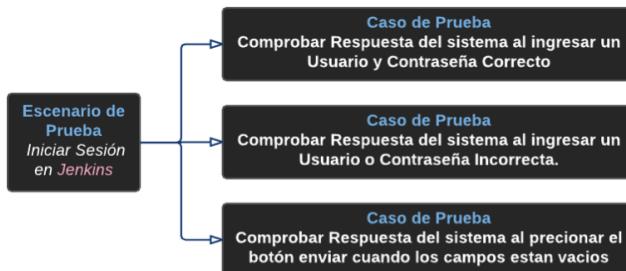


Figura 10.3: Diagrama de entrada y salida

ID	Escenario	Caso de prueba	Entrada	Salida esperada
02.1	Iniciar sesión Jenkins	Ingreso de campos usuario y contraseña correctos	Usuario:'almunoiciuv', Password: alumnoiciuv	El usuario inicia sesión

Tabla 10.7: Resumen de resultados

ID	Escenario	Caso de prueba	Entrada	Salida esperada
02.2	Iniciar sesión Jenkins	Ingreso del campo usuario incorrecto	Usuario:'almunouvici', Pass-word:'alumnouvici'	Error al iniciar sesión
02.2	Iniciar sesión Jenkins	Ingreso del campo contraseña incorrecto	Usuario:'almunoiciuv', Pass-word:'alumnouvici'	Error al iniciar sesión
02.2	Iniciar sesión Jenkins	Ingreso ambos campos incorrectos	Usuario:'almunouv', Password:'alumnouv'	Error al iniciar sesión

Tabla 10.8: Resumen de resultados

ID	Escenario	Caso de prueba	Entrada	Salida esperada
02.3	Iniciar sesión Jenkins	Campo usuario vacio	Password: alumnouvi-ci	El sistema indica que el campo usuario se encuentra vacio.
02.3	Iniciar Sesión Jenkins	Campo Usuario Vacio	Usuario: alumnouvici	El sistema indica que el campo usuario se encuentra vacio.
02.3	Iniciar sesión Jenkins	Campo usuario y contraseña Vacio	vacio	El sistema indica que el campo usuario se encuentra vacio.

Tabla 10.9: Resumen de resultados

### Prueba funcional 03

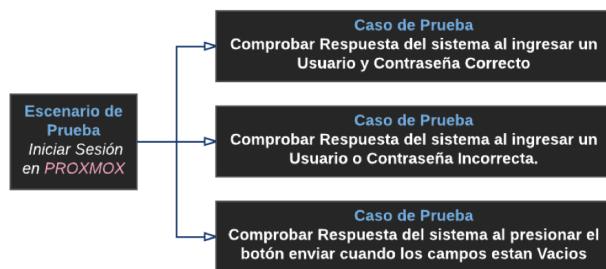


Figura 10.4: Diagrama de entrada y salida

ID	Escenario	Caso de prueba	Entrada	Salida esperada
03.1	Iniciar sesión Proxmox	Ingreso de campos usuario y contraseña correctos	Usuario:'root', Password: adminici	El usuario Inicia sesión

Tabla 10.10: Resumen de resultados

ID	Escenario	Caso de prueba	Entrada	Salida esperada
03.2	Iniciar sesión Proxmox	Ingreso del campo Usuario incorrecto	Usuario:'rot', Password:'adminici'	Error al iniciar sesión
03.2	Iniciar sesión Proxmox	Ingreso del campo contraseña incorrecto	Usuario:'root', Password:'iciadmin'	Error al iniciar sesión
03.2	Iniciar sesión Proxmox	Ambos campos incorrectos	Usuario:'root', Password:'iciadmin'	Error al iniciar sesión

Tabla 10.11: Resumen de resultados

ID	Escenario	Caso de prueba	Entrada	Salida esperada
03.3	Iniciar sesión en Proxmox	Campo Usuario vacio	Password: alumnouvici	El sistema indica que el campo usuario se encuentra vacio.
03.3	Iniciar sesión Promox	Campo usuario vacio	Usuario: alumnouvici	El sistema indica que el campo usuario se encuentra vacio.
03.3	Iniciar sesión Proxmox	Campo usuario y Contraseña Vacio	vacio	El sistema indica que el campo usuario se encuentra vacio.

Tabla 10.12: Resumen de resultados

## Prueba funcional 04

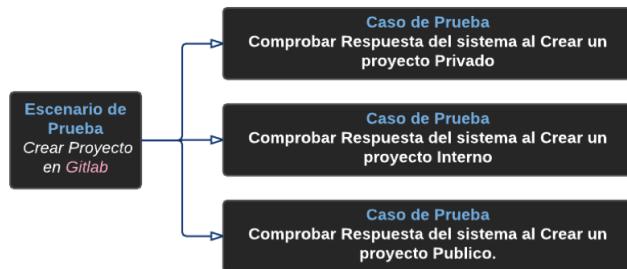


Figura 10.5: Diagrama de entrada y salida

ID	Escenario	Caso de prueba	Entrada	Salida esperada
04.1	Crear proyecto en Gitlab	Crear proyecto privado	Nombre: ProyectoPrivado	Proyecto creado

Tabla 10.13: Resumen de resultados

ID	Escenario	Caso de prueba	Entrada	Salida esperada
04.2	Crear proyecto en Gitlab	Crear proyecto interno	Nombre: ProyectoInterno	Proyecto creado

Tabla 10.14: Resumen de resultados

ID	Escenario	Caso de prueba	Entrada	Salida esperada
04.3	Crear proyecto en Gitlab	Crear proyecto público	Nombre: ProyectoPublico	Proyecto creado

Tabla 10.15: Resumen de resultados

## Prueba Funcional 05

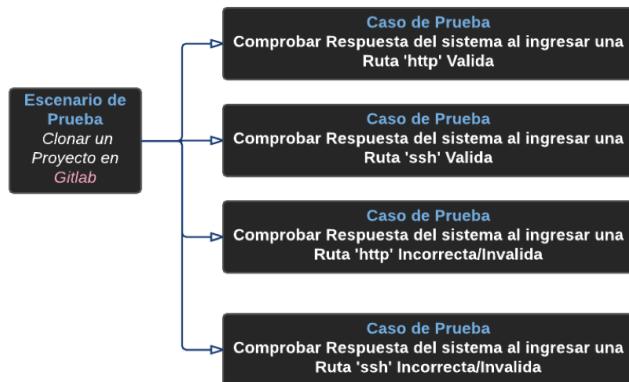


Figura 10.6: Diagrama de entrada y salida

ID	Escenario	Caso de prueba	Entrada	Salida esperada
05.1	Clonar un proyecto	Clonar usando una ruta 'http' Valida	http://RepoCentral.uvici: <del>social</del> /root/proyecto-ci.git	Los archivos del proyecto son recibidos correctamente

Tabla 10.16: Resumen de resultados

ID	Escenario	Caso de prueba	Entrada	Salida esperada
05.2	Clonar un proyecto	Clonar usando una ruta 'ssh' Valida	git@RepoCentral.uvici: <del>root</del> /proyecto-ci.git	Los archivos del proyecto son recibidos correctamente

Tabla 10.17: Resumen de resultados

ID	Escenario	Caso de prueba	Entrada	Salida esperada
05.3	Clonar un proyecto	Clonar un proyecto Usando una ruta 'http' incorrecta	http://10.100.62.21/root/ <del>proyecto</del> /proyecto-ci.git	Los archivos propiados al proyecto son recibidos correctamente.

Tabla 10.18: Resumen de resultados

ID	Escenario	Caso de prueba	Entrada	Salida esperada
05.4	Clonar un proyecto	Clonar un proyecto Usando una ruta 'http' Incorrecta	http://10.100.6.221/root/ <del>asignados</del> al proyecto proyecto-ci.git	Los archivos son recibidos correctamente.

Tabla 10.19: Resumen de resultados

**Prueba funcional 06**

Figura 10.7: Diagrama de entrada y salida

ID	Escenario	Caso de prueba	Entrada	Salida esperada
06.1	Recibir cambios a proyecto Gitlab	Push de un usuario al puente maestro de un proyecto	Consola: Push Origin Master	Archivos recibidos correctamente.

Tabla 10.20: Resumen de resultados

**Prueba funcional 07**

Figura 10.8: Diagrama de entrada y salida

ID	Escenario	Caso de prueba	Entrada	Salida esperada
06.1	Crear una tarea en jenkins	Comprobar la respuesta del sistema al crear una tarea con origen de código Fuente	git@RepoCentral.uvici:root/php-projecto-ci.git	Tarea creada
06.1	Crear una tarea en jenkins	Comprobar la Respuesta del sistema al crear una Tarea con disparadores de ejecución	CheckBox: Git push	Tarea creada

Tabla 10.21: Resumen de resultados

### Prueba funcional 08



Figura 10.9: Diagrama de entrada y salida

ID	Escenario	Caso de prueba	Entrada	Salida esperada
06.1	Integrar código fuente asociado a una tarea	Comprobar la respuesta del sistema al generar una construcción	push origin master	Los archivos asociados al proyecto son recibidos correctamente.

Tabla 10.22: Resumen de resultados

### 10.2.2. Análisis de resultados

En la ejecución de las pruebas Funcionales realizadas en la sección 10.1, se obtuvieron los siguientes resultados:

- Casos de prueba ejecutados 22.
- Casos de prueba aprobados 22.

### 10.3. Pruebas de sistema

Debido a que la evaluación se aplica sobre la implementación completa de la solución, pueden ser utilizadas distintos tipos de pruebas para examinar propiedades no funcionales del sistema. La cantidad de usuarios que se espera soporte la infraestructura es proporcional a los recursos que poseen las máquinas que ejecutan las herramientas o componentes de Software. La actual distribución de los recursos asegura un flujo, sin problemas, de a lo más 100 usuarios concurrentes.

- Pruebas de stress: Prueba conducida para evaluar un sistema o componente en sus límites de desempeño especificados o requeridos.
- Pruebas de carga: Pruebas realizadas para evaluar el cumplimiento de un sistema o un componente según los requisitos de rendimiento especificados.

#### Plan de Pruebas - Repositorio Central

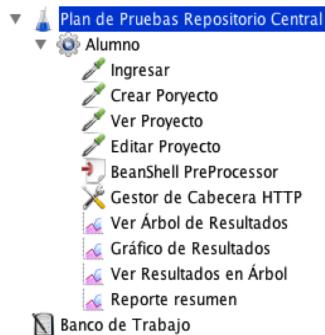


Figura 10.10: Detalle del plan de pruebas

#### Plan de Pruebas - Servidor I.C

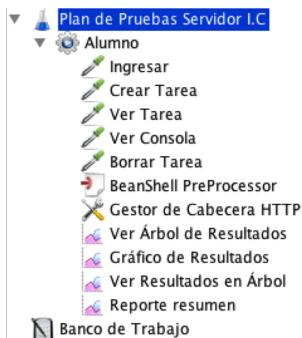


Figura 10.11: Detalle del plan de pruebas

### 10.3.1. Pruebas de carga

El plan de pruebas simula un hilo de Usuarios del tipo 'Alumno' que ejecuta una serie de peticiones. Haciendo uso de un receptor se interpreta la información del muestreador.

#### Pruebas de Carga - Jenkins

Para esta prueba se simula una carga de 100 usuarios/segundo en un periodo de 10 segundos.

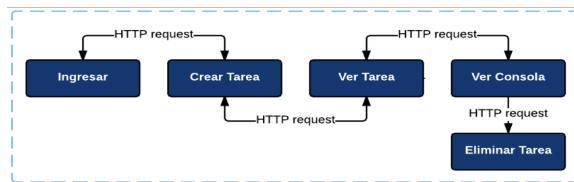


Figura 10.12: Diagrama de pruebas

El gráfico de resultados muestra el throughput (color verde) del sistema y carga de datos en función del tiempo. El servidor logra resolver el 100 por ciento de las peticiones.

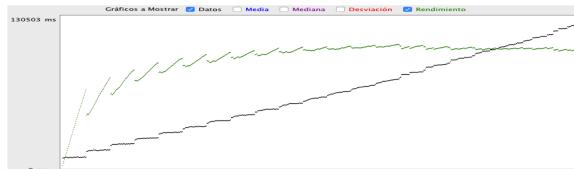


Figura 10.13: Grafico de resultados

#### Prueba de Carga - Gitlab

Para esta prueba se simula una carga de 100 usuarios/segundo en un periodo de 10 segundos.

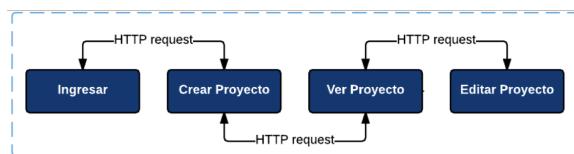


Figura 10.14: Diagrama de pruebas

El gráfico de resultados muestra el throughput del sistema y carga de datos en función del tiempo.

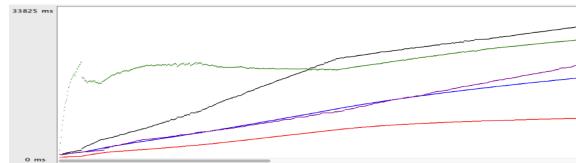


Figura 10.15: Grafico de resultados

### Prueba de Carga - Gitlab

Para esta prueba se simula una carga de 100 usuarios/segundos en períodos de 1 segundo. El servidor logra resolver el 100 por ciento de las peticiones.

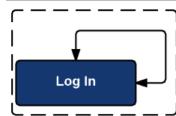


Figura 10.16: Diagrama de pruebas

El gráfico de resultados muestra el throughput del sistema y carga de datos en función del tiempo. El servidor logra resolver el 100 por ciento de las peticiones.

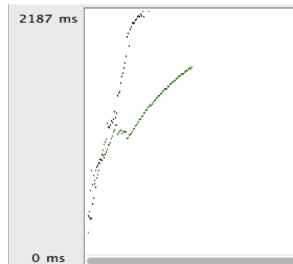


Figura 10.17: Grafico de resultados

### 10.3.2. Pruebas de estrés

#### Prueba de Estrés - Jenkins

Para esta prueba se simula una carga de 200 usuarios/segundos en períodos de 10 segundos.

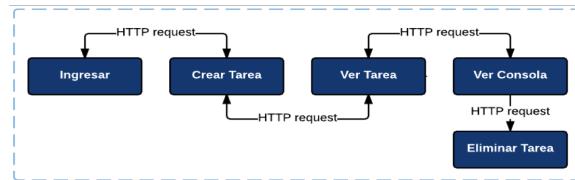


Figura 10.18: Diagrama de pruebas

El sistema no resiste la prueba, el 80 por ciento de las peticiones llega al timeout. En la Figura podemos apreciar un desempeño discontinuo.

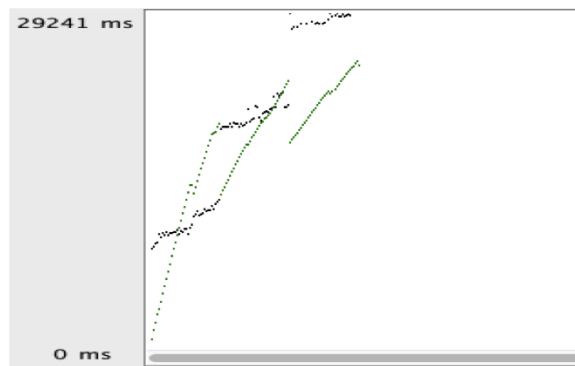


Figura 10.19: Grafico de resultados

### Prueba de Estrés - Gitlab

Para esta prueba se simula una carga de 200 usuarios en un segundo.

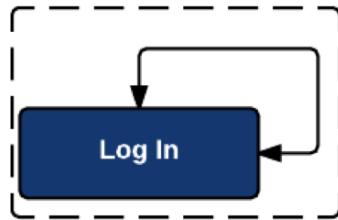


Figura 10.20: Diagrama de pruebas

El gráfico de resultados muestra el throughput del sistema y carga de datos en función del tiempo. El servidor logra resolver el 100 por ciento de las peticiones.

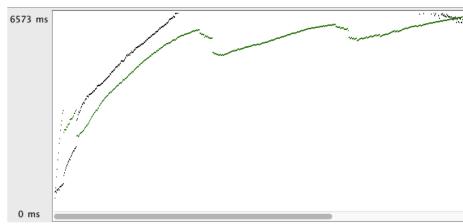


Figura 10.21: Grafico de resultados

### 10.3.3. Análisis de resultados

En la ejecución de las pruebas funcionales realizadas en la sección 10.2, se obtuvieron los siguientes resultados:

- Casos de prueba ejecutados 5.
- Casos de prueba aprobados 5.

## 10.4. Pruebas de validación

Para esta prueba se considera un grupo de 3 alumnos, cada uno con una tarea específica en la actividad.

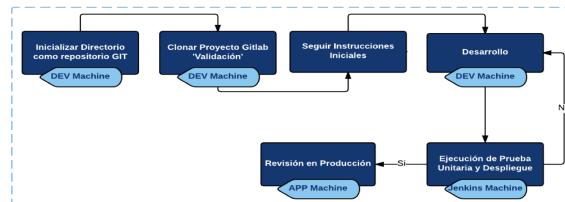


Figura 10.22: Diagrama de validación

### 10.4.1. Análisis de resultados

- Casos de prueba ejecutados 1.
- Casos de prueba aprobados 1.

A partir de los datos obtenidos se puede afirmar que el sistema se comporta de forma correcta y constante. Las mediciones de CPU y memoria de la Figura 10.23 demuestra que el sistema soporta, en condiciones normales y límites, las operaciones especificadas.



Figura 10.23: Desempeño del repositorio central

# **Capítulo 11**

## **Implantación**

A continuación se describe el proceso de implantación que se llevó a cabo en el entorno real donde funciona el sistema, cada host conectados a la infraestructura puede enviar y recibir datos del repositorio central. Para mayor detalle sobre la instalación de cada una de las herramientas y tecnologías presentes en esta implantación leer el manual de implantación [Apéndice A].

### **11.1. Características del ambiente**

La escuela de Ingeniería Civil Informática de la Universidad de Valparaíso permite el uso de un servidor con las siguientes características.

- Procesador: Intel(R) Xeon(R) CPU E5504, 2GHz.
- Memoria Ram: 4GB
- Disco Duro: 320GB
- DVD: DV-28E-V

### **11.2. Actividades previas a la implantación**

Instalación del sistema operativo Proxmox Virtual Environment 3.4 siguiendo los pasos que se muestran a continuación:

- Insertar el disco de instalación de Proxmox VE 3.4
- Particionar el disco duro que contendrá el sistema operativo usando zfe 0.

- Configurar la red del servidor con la dirección IP: [10.100.6.4] Gw: [10.100.6.254] DNS: [10.50.1.16].
- Acceder al servicio web usando <https://10.100.6.4:8006>

### Contenedores OpenVZ

Desde la interfaz grafica de Proxmox Virtual Environment 3.4 se crearon, iniciaron y actualizaron los siguientes servidores virtuales.

- Contenedor OpenVZ: Repositorio Central
- Contenedor OpenVZ: Integración PHP
- Contenedor OpenVZ: Integración Java
- Contenedor OpenVZ: Servidor C
- Contenedor OpenVZ: Producción PHP
- Contenedor OpenVZ: Producción Java

### 11.3. Actividades propias de la implantación

La implantación del sistema se realizo siguiendo la secuencia de actividades descritas a continuación:

- Primero se debe instalar el JDK development environment 1.8 y la herramienta Gitlab 8.10 en el Repositorio Central, para comprobar el funcionamiento, acceder a Gitlab desde un navegador con la siguiente dirección <http://10.100.6.221>
- Llevar a cabo la instalación de la herramienta Jenkins, PHP 5.6, composer y JDK 1.8 en el servidor de integración continua PHP, para comprobar su funcionamiento acceder a Jenkins desde un navegador con la siguiente dirección <http://10.100.6.226:8080>
- Llevar a cabo la instalación de la herramienta Jenkins y JDK 1.8 en el servidor de integración continua para el lenguaje PHP, para comprobar su funcionamiento acceder a Jenkins desde un navegador con la siguiente dirección <http://10.100.6.227:8080>.
- En el Servidor C se instaló la herramienta gcc 1.7 y el editor de texto Vim.
- Instalación de PHP 5.6 y apache 2 en el servidor de despliegue PHP.
- Instalación de JDK 1.8 en el servidor de despliegue Java.

# **Capítulo 12**

## **Conclusión**

Como resultado del trabajo realizado con respecto al desarrollo de la infraestructura, se pudo determinar que las herramientas y tecnologías permitieron el desarrollo y cumplimiento de requerimientos de este trabajo de título. Además, el uso conjunto de estas tecnologías ofrece una alternativa al proceso de desarrollo de software a la hora de ejecutar actividades asociadas al ciclo de vida del software. Facilitando el trabajo entre los miembros de un equipo, que constantemente realiza cambios y pruebas a uno o varios proyectos. Por otro lado esta solución integra herramientas y métodos de desarrollo que agilizan el proceso de desarrollo de software, asegurando experiencias más realistas, traducibles en profesionales más preparados, competentes y seguros.

# Apéndice A

## Manual de implantación

A continuación se especifican los detalles correspondientes a la instalación de tecnologías y herramientas del sistema.

### A.1. Instalar Proxmox VE 3.4

- Paso 1: Insertar la imagen de Proxmox en la unidad de DVD del servidor
- Paso 2: Seleccionar idioma y teclado, tal como se indica en la [Figura A.1].

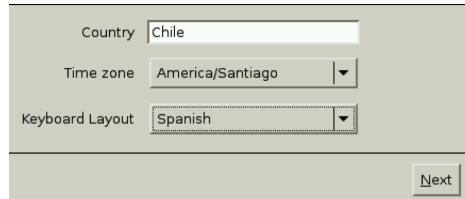


Figura A.1: Selector de lenguaje

- Paso 3: Formatear disco usando ext 3 [Figura A.2].

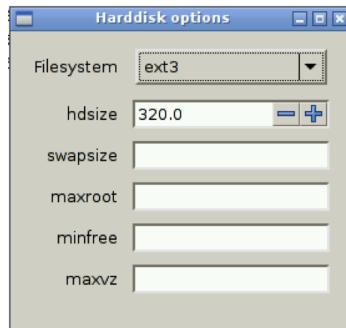


Figura A.2: Formatear disco duro

- Paso 4: Configurar los parametros de conexión, tal como se indica [Figura A.3].

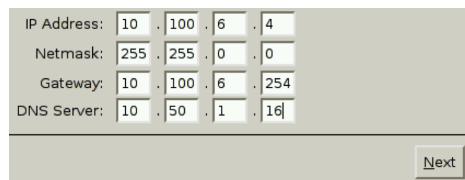


Figura A.3: Configurar red

## A.2. Instalar JDK Development Environment 1.8

- Agregar el repositorio webupd8team Java PPA, usando: add-apt-repository ppa:webupd8team/java
- Actualizar la source list con: apt-get update
- Instalar: apt-get install oracle-java8-installer

### A.2.1. Instalar PHP 5.6

- Agregar los repositorios a la source list usando: add-apt-repository ppa:ondrej/php5-5.6
- Actualizar: apt-get update
- Instalar: sudo apt-get install php5

### A.2.2. Instalar Gitlab 8.10

- Instalar y configurar las dependencias necesarias, usando: sudo apt-get install curl openssh-server ca-certificates postfix
- Agregar los paquetes de gitlab 8.10 al servidor: curl -sS https://packages.gitlab.com/install/repositories/ce/script.deb.sh — sudo bash
- instalar Gitlab: apt-get install gitlab-ce
- Configurar e iniciar gitlab: gitlab-ctl reconfigure

### A.3. Instalar Jenkins

- Agregar repositorios a la sourcelist:sh -c echo deb http://pkg.jenkins.io/debianstable binary/ /etc/apt/sources.list.d/jenkins.list
- Actualizar: apt-get update
- instalar: apt-get install jenkins.

### A.4. Administrador Proxmox

Este tipo de usuario puede acceder a la web GUI de Proxmox que facilita la creación, modificación y eliminación de contenedores [?]. Ingresar con la credencial que se muestra a continuación.

- URL: https://10.100.6.4:8006
- Usuario: root
- Password: proxuvici

### A.5. Administrador Jenkins

Este usuario puede crear tareas de integración [?] en la herramienta Jenkins para consumir datos del repositorio central. Ingresar con la credencial que se muestra a continuación.

- URL: https://10.100.6.226:8080 [PHP]
- Usuario: root

- Password: gitlabuvici
- URL: <https://10.100.6.227:8080> [Java]
- Usuario: root
- Password: gitlabuvici

## A.6. Desarrollador Gitlab

Para el uso de la herramienta Gitlab no hay restricciones de tipo de usuario, solo es requerido el registro de una cuenta para hacer uso de la herramienta [?], acceder desde:

- URL: <http://10.100.6.221:80>

## A.7. Flujo de trabajo

El uso del sistema es como se muestra en la Figura B.1, en el contexto de la utilización de la metodología ágil [Feature Driven Development], se tendría el siguiente flujo de trabajo.

Primero un grupo de usuarios, trabaja en el desarrollo de una característica, con tiempo de implementación menor a dos días, estos desarrolladores hacen push (envían cambios al repositorio central) de sus últimas versiones. Luego el servidor de integración continua, Java o PHP, automáticamente ejecuta los test a las clases y construye una Build de la ultima versión del Master branch, esto a través de la configuración de un trigger, si todas las pruebas son exitosas, el servidor de integración continua se encarga de mover los archivos a los ambientes de producción correspondiente.

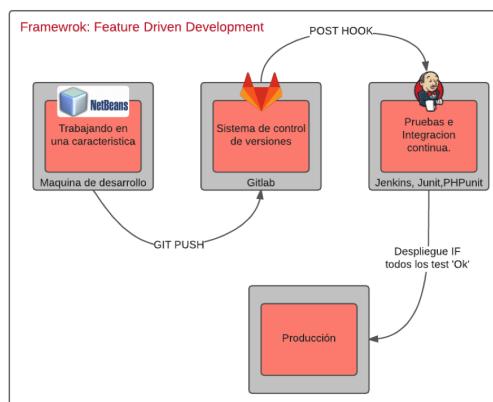


Figura A.4: Flujo general del sistema

# Bibliografía

- [1] Computer science curriculum . *ACM,IEEE*, 2008.
- [2] Online community for open source software education. [Online]: <http://www.teachingopensource.org>, 2016.
- [3] Oyenuga Augustine Yomi Akinola Kayode-Emmanuel, Olanrewaju Grace-Oluwadamilare. Improvement strategies for computer science students' academic performance in programming skill. *ACM,IEEE*, 2008.
- [4] Patrice Frison Daniel Deveaux, Regis Fleurquin. Software engineering teaching: a “docware” approach.
- [5] N.E. Gibbs. The sei education program: the challenge of teaching future software engineers.
- [6] Clinton J. Tight spiral projects for communicating software engineering concepts.
- [7] F.P. Deek J. DeFranco-Tommarello. Collaborative software development: A discussion of problem solving models and groupware technologies. *IEEE*, 2002.
- [8] F. Pena-Mora J. Favela. An experience in collaborative software engineering education.
- [9] J. Garcia-Fanjul J. Tuya. Teaching requirements analysis by means of student collaboration.
- [10] Kevin A. Gary John D. Tvedt, Roseanne Tesoriero. The software factory: Combining undergraduate computer science and software engineering education. *IEEE Software*.
- [11] Ju Long. Experience teaching a graduate course in open source software engineering.
- [12] DeClue Lu, B. Teaching agile methodology in a software engineering capstone course.

- [13] O'Reilly Media. The cathedral and the bazaar.
- [14] R. S. Kerridge R. J. Dawson, R. W. Newsham. Introducing new software engineering graduates to the 'real world' at the gpt company.
- [15] D. Boskovic S. Kalem, D. Donko. Agile methods for cloud computing.
- [16] Stefan Brandle Tom Nurkkala. Software studio: teaching professional software engineering.
- [17] Anthony I. Wasserman. Wasserman a.i. toward a discipline of software engineering.