



Facultad de Ingeniería  
Escuela de Ingeniería Civil en Informática

# **DESARROLLO DE UNA PLATAFORMA PARA LA SOLICITUD Y GESTIÓN DE REQUERIMIENTOS Y SCM**

Por

**Alejandro Alvarez Ahumada**

Trabajo realizado para optar al Título de  
**INGENIERO CIVIL EN INFORMÁTICA**  
Prof. Guía: Carlos Becerra Castro  
Prof. Co-Referente: Nombre Profesor Correferente  
Septiembre 2012

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.

---

Carlos Becerra Castro    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.

---

Nombre Profesor Correferente    Profesor Co-Referente

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.

---

Nombre Profesor Informante 1    Profesor Informante

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

# Resumen

La Dirección de Servicios de Información y Computación (DISICO) de la Universidad de Valparaíso durante los últimos años ha estado en constante crecimiento y en busca de mejoras que le permitan brindar un mejor servicio. Aunque en este poco tiempo son muchas las mejoras que se han hecho, aún quedan aspectos por mejorar, algunos de estos son los procesos relacionados a las solicitudes de requerimientos y solicitudes de cambios, para las cuales ya se han diseñado procedimientos y metodologías, sin embargo se carece de una herramienta que permita la automatización de estas. El propósito de este trabajo de título es dar solución a dicho problema mediante el desarrollo de una plataforma que permita automatizar los procedimientos actuales de solicitud de requerimientos y SCM. Los principales resultados que se esperan son disminuir el tiempo y esfuerzo invertido en la aplicación de las metodologías que existen actualmente.

# Agradecimientos

Aquí pueden colocar sus agradecimientos. Si han estudiado con becas es recomendable colocar los agradecimientos a las instituciones que les otorgaron las becas.

# Índice general

<b>Resumen</b>	<b>III</b>
<b>Agradecimientos</b>	<b>IV</b>
<b>1. Pruebas</b>	<b>1</b>
1.1. Pruebas Unitarias . . . . .	1
1.1.1. Análisis de Resultados . . . . .	7
1.1.2. Problemas detectados . . . . .	7
1.2. Pruebas Integración . . . . .	8
1.2.1. Análisis de Resultados . . . . .	9

# Índice de tablas

1.1. Test unitarios . . . . .	7
1.2. Test Integración . . . . .	9

# Índice de figuras

# Capítulo 1

## Pruebas

En este capítulo se detallan las pruebas realizadas, junto con los resultados obtenidos durante la realización de estas. Las pruebas realizadas se dividen en:

- Pruebas Unitarias.
- Pruebas de Integración.
- Pruebas de Rendimiento.
- Pruebas de Aceptación.
- Pruebas Beta.

El principal enfoque de las pruebas es la detección de errores.

### 1.1. Pruebas Unitarias

Para llevar a cabo las pruebas unitarias se diseñó un set de pruebas automatizadas, las cuales fueron implementadas haciendo uso del framework de pruebas JUnit y de Glassfish Embedded (dentro del cual se despliegan los EJB para ser utilizados durante la ejecución de las pruebas).

Hacer pruebas JUnit en clases java corrientes, es bastante simple y no presenta mayor dificultad, sin embargo cuando se desea realizar pruebas, ya sea unitarias o de integración, en clases java que son Enterprise Java Beans (EJBs), se presentan algunas dificultades las cuales radican principalmente en el hecho que los EJB son componentes gestionados por un servidor de aplicaciones y por ende no pueden funcionar fuera de este, esto significa que para poder probar los EJB debemos desplegar los componentes dentro de un servidor de aplicaciones, junto con las configuraciones de persistencia y datasources correspondientes,



para solucionar este inconveniente es que se utiliza Glassfish Embedded, que actúa como contenedor embebido el cual es mas rápido y ligero.

El procedimiento utilizado para ejecutar los test dentro Glassfish Embedded es el siguiente:

- Copiar los archivos del directorio *build/classes* a un directorio temporal *build/embedded* para el despliegue de los EJB.
- Además se copia el archivo *test-persistence.xml* (que contiene una configuración diferente para la ejecución de las pruebas en una base de datos diferente a la de producción) dentro del directorio *build/embedded/META-INF/* y se renombra como *persistence.xml*.
- Antes de levantar Glassfish Embedded se crea un *properties* de configuración que le indica a Glassfish donde se encuentran los módulos a desplegar y el archivo de configuración *domain.xml* que debe utilizar (en este caso es un archivo aparte que contiene la configuración necesaria para la ejecución del test).
- Luego se inicia el Glassfish Embedded.
- Se ejecutan los test necesarios.
- Por ultimo se cierra Glassfish Embedded y se borra el directorio temporal *build/embedded*.

Para esto gestionar todo ese proceso se implemento una nueva clase denominada *BaseTestEJB* de la cual extienden todos los test que hacen uso de los EJB.

De acuerdo a lo que se especifico en la fase de diseño de pruebas, se opto por no realizar una documentación extensiva de los test unitarios ya que estos deben ser auto-explicativos y de manera tal que el mismo código documenta la prueba, sus entradas y salidas esperadas.

Sin embargo a continuación en la Tabla 1.1 se presenta un resumen las pruebas unitarias diseñadas y ejecutadas, con su nombre y propósito.

Clase	Test	Propósito
Resources	testGetValue	Verificar que el método getValue es capaz de recuperar la cadena "ABCD" desde un archivo de propiedades.
	testGetValueCon Espacios	Verificar que el método getValue es capaz de recuperar la cadena "A B C D" desde un archivo de propiedades sin verse afectado por la cantidad de espacios entre los caracteres.
	testGetValueShort	Verificar que el método getValueShort es capaz de recuperar cadena desde un archivo de propiedades y convertirla a Short siempre que cumpla con el formato de este.
	testGetValueShort Negativo	Verificar que el método getValueShort es capaz de recuperar cadena desde un archivo de propiedades y convertirla a Short aunque este sea negativo.
	testGetValueShort ErrorEnString	Verificar que el método getValueShort dispara la excepción NumberFormatException al leer un String desde el archivo de propiedades.
	testGetValueShort ErrorValorMayor AShort	Verificar que el método getValueShort dispara la excepción NumberFormatException al leer un número entero que excede el valor máximo de un Short.
	testGetValueShort ErrorValorDecima	Verificar que el método getValueShort dispara la excepción NumberFormatException al leer un valor con decimales desde el archivo de propiedades.
	testGetValueInteger	Verificar que el método getValueInteger es capaz de recuperar cadena desde un archivo de propiedades y convertirla a Integer siempre que cumpla con el formato de este.
	testGetValueInteger Negativo	Verificar que el método getValueInteger es capaz de recuperar cadena desde un archivo de propiedades y convertirla a Integer aunque este sea negativo.
	testGetValueInteger ErrorValorMayorAInteger	Verificar que el método getValueInteger dispara la excepción NumberFormatException al leer un número entero que excede el valor máximo de un Integer.

Clase	Test	Propósito
Resources	testGetValueInteger ErrorValorMayorInteger	Verificar que el método getValueInteger dispara la excepción NumberFormatException al leer un numero entero que excede el valor máximo de un Integer.
	testGetValueInteger ErrorValorDecimal	Verificar que el método getValueInteger dispara la excepción NumberFormatException al leer un valor con decimales desde el archivo de propiedades.
	testGetValueLong	Verificar que el método getValueLong es capaz de recuperar cadena desde un archivo de propiedades y convertirla a Long siempre que cumpla con el formato de este.
	testGetValueLong Negativo	Verificar que el método getValueLong es capaz de recuperar cadena desde un archivo de propiedades y convertirla a Long aunque este sea negativo.
	testGetValueLong ErrorValorMayorALong	Verificar que el método getValueLong dispara la excepción NumberFormatException al leer un numero entero que excede el valor máximo de un Long.
	testGetValueLong ErrorValorDecimal	Verificar que el método getValueLong dispara la excepción NumberFormatException al leer un valor con decimales desde el archivo de propiedades.
	testGetPropertiesPath	Verifica que el metodo getPropertiesPath obtenga la ruta de un archivo properties correctamente.
	testGetPropertiesPath NotFound	Verifica que el metodo getPropertiesPath dispare la excepcion MissingResourceException al intentar obtener la ruta de un propertie que no existe.
	testGetPageList	Verifica que el metodo getPageList obtenga una lista con todas las propiedades contenidas en el archivo propertie especificado.
	testGetMapPageList	Verifica que el metodo getMapPageList obtenga un Map con todas las propiedades contenidas en el archivo propertie especificado.

Clase	Test	Propósito
MathUtil	testCalcularPorcentajeRedondeado	Verifica que el calculo del porcentaje entero sin decimal.
	testCalcularPorcentajeRedondeadoParaArriba	Verifica que el porcentaje sea redondeado hacia arriba cuando el decimal es mayor o igual a 5.
	testCalcularPorcentajeRedondeadoParaAbajo	Verifica que el porcentaje sea redondeado hacia abajo cuando el decimal es menor a 5.
	testCalcularReglaDeTres	Verifica el calculo de una regla de tres.
	testCalcularReglaDeTresParaDenominadorCero	Verifica que se devuelva un 0 en caso que el denominador sea 0.
	testCalcularReglaDeTresParaNumeradorCero	Verifica que se devuelva un 0 en caso que el numerador sea 0.
	testCalcularReglaDeTresParaCien	Verifica el caso en que se debe devolver 100.
	testCalcularReglaDeTresParaPorcentajeMayorACien	Verifica que funcione cuando el numerador es mayor a cien.
	testRedondearCero	Verifica la funcion redondear cuando el numero es cero.
	testRedondearParaArribaSinDecimal	Verifica que el numero sea redondeado hacia arriba cuando el decimal es mayor o igual a 5 y el resultado no debe tener ningún decimal.
	testRedondearParaAbajoSinDecimal	Verifica que el numero sea redondeado hacia abajo cuando el decimal es menor a 5 y el resultado no debe tener ningún decimal.
	testRedondearParaArribaConUnDecimal	Verifica que el numero sea redondeado hacia arriba cuando el decimal es mayor o igual a 5 y el resultado no debe tener un decimal.
	testRedondearParaAbajoConUnDecimal	Verifica que el numero sea redondeado hacia abajo cuando el decimal es menor a 5 y el resultado no debe tener un decimal.
	testRedondearParaArribaConDecimal5	Verifica el caso en que el decimal es 5.
	testRedondearFloatCero	Verifica la función redondear cuando el numero es cero. Cuando el resultado es convertido de BigDecimal a Float.

Clase	Test	Propósito
MathUtils	testRedondearFloat ParaArribaSinDecimal	Verifica que el numero sea redondeado hacia arriba cuando el decimal es mayor o igual a 5 y el resultado no debe tener ningún decimal. Cuando el resultado es convertido de BigDecimal a Float.
	testRedondearFloat ParaAbajoSinDecimal	Verifica que el numero sea redondeado hacia abajo cuando el decimal es menor a 5 y el resultado no debe tener ningún decimal. Cuando el resultado es convertido de BigDecimal a Float.
	testRedondearFloat ParaArribaConUnDecimal	Verifica que el numero sea redondeado hacia arriba cuando el decimal es mayor o igual a 5 y el resultado no debe tener un decimal. Cuando el resultado es convertido de BigDecimal a Float.
	testRedondearFloat ParaAbajoConUnDecimal	Verifica que el numero sea redondeado hacia abajo cuando el decimal es menor a 5 y el resultado no debe tener un decimal. Cuando el resultado es convertido de BigDecimal a Float.
	testRedondearFloat ParaArribaConDecimal5	Verifica el caso en que el decimal es 5. Cuando el resultado es convertido de BigDecimal a Float.
TimerSolicitud Requerimientos UpdateTest	testBuscarSolicitudes VencidasVerificar Cambio	Verifica que el método buscarSolicitudesVencidas haya cambiado el estado de una solicitud enviada cuya fecha de vencimiento ya fue superada.
	testBuscarSolicitudes VencidasVerificar NoCambioVencida	Verifica que el método buscarSolicitudesVencidas no haya cambiado el estado de solicitudes que ya se encontraban en estado vencida.
	testBuscarSolicitudes VencidasVerificar NoCambioCerrada	Verifica que el método buscarSolicitudesVencidas no haya cambiado el estado de solicitudes que se Cerraron después de la fecha de vencimiento.
	testBuscarSolicitudes VencidasVerificar NoCambioFinalizada SinRespuesta	Verifica que el método buscarSolicitudesVencidas no haya cambiado el estado de solicitudes que se Finalizaron sin respuesta después de la fecha de vencimiento.
	testBuscarSolicitudes VencidasVerificar NoCambioSolicitud SinFechaVencimiento	Verifica que el método buscarSolicitudesVencidas no haya cambiado el estado de solicitudes que no poseen fecha de vencimiento

Clase	Test	Propósito
SolicitudRequerimientoEJB	testGenerarCodigoNumCero	Verifica que el código es vacío cuando el número es cero.
	testGenerarCodigoNumNegativo	Verifica que el código es vacío cuando el número es negativo.
	testGenerarCodigoNumUno	Verifica que el código es "q" cuando el número es 1.
	testGenerarCodigoMaxLong	Verifica que el código es "2teCogGBXee" cuando el número es el máximo valor de un Long.
	testGenerarCodigoConsultaNoDuplicados	Verifica que el código no se repite aunque sea generado en instantes de tiempo casi iguales y con ruts similares.
	testValidarCodigoConsultaExistente	Verifica que un código de consulta sea invalido cuando ya existe.
	testValidarCodigoConsultaInexistente	Verifica que un código de consulta es invalido cuando no existe.

Tabla 1.1: Test unitarios

### 1.1.1. Análisis de Resultados

Todas las pruebas han sido implementadas y ejecutadas satisfactoriamente, y todas han sido superadas, es decir luz verde. Todos aquellos que no fueron superados, fueron corregidos y solucionados a la brevedad. Y ante cualquier modificación del código fuente en el futuro, debe volver a ser probadas con este mismo set de pruebas y corregido hasta volver a lograr un 100 % de aprobación de los tests.

### 1.1.2. Problemas detectados

Durante el desarrollo de los tests, los principales problemas que existieron fueron principalmente debido a la incorrecta configuración de los frameworks de pruebas, pero una vez superado aquello, no se encontraron demasiados problemas. Los principales problemas que se detectaron y corrigieron.

- Problemas en el calculo de porcentajes, cuando existía un denominador 0.
- Duplicación de códigos de consulta en lapsos de tiempos muy cercanos, al no existir variación en la semilla del generador de números aleatorios utilizado en una parte del algoritmo.

## 1.2. Pruebas Integración

Una vez completada la fase de pruebas unitarias, se realizaron las pruebas de integración, las cuales a diferencia de los especificado en la fase de diseño, se opto por no utilizar Arquillian ya que solo era necesario para utilizar inyección de dependencia en los test. Y presenta algunos problemas en proyectos que no gestionan sus dependencias con maven.

Las pruebas realizadas en esta fase se separaron en 2 tipos, las que prueban directamente la capa de persistencia (ya que probarlas requiere modificar el estado de la base de datos y normalmente de a lo menos las funciones crear, buscar y eliminar, para poder agrega nuevos datos al inicio del test, buscar para verificar la correcta creación o modificación de algo y la eliminación para restaurar el estado de la bd, a su estado original). Y ademas un conjunto de pruebas acorde al esquema de integración definido. En la Tabla 1.2 se presentan las pruebas de integración realizadas.

Nº	Objetivo	Prueba
1	Comprobar la integración entre el Portal de Login institucional y el sistema interno de Autenticación.	Ingresar con una cuenta registrada (rut 11111111), a través del portal de Login institucional y verificar que al autenticarse el usuario es redireccionado a la aplicación.
2	Comprobar la integración entre el modulo de Autenticación y el de Envío de Solicitudes.	Ingresar al sistema a través del portal de Login Institucional con el usuario 11111111, y corroborar que los datos personales del usuario mostrados en la aplicación son los mismo proporcionados por LDAP.
3	Comprobar la integración entre el modulo de Envío de Solicitudes y el modulo de Mensajería.	Crear una solicitud de requerimiento y verificar que esta aya sido creada, que un correo electrónico aya sido enviada al mail del solicitante y que el código enviado sea el mismo de la solicitud.
4	Comprobar la integración entre el modulo de Envío de Solicitudes y el modulo de Atención de solicitudes.	Crear una nueva solicitud de requerimiento para el Área de Desarrollo y posteriormente ingresar con la cuenta correspondiente al jefe del área de desarrollo y corroborar que la solicitud creada aparezca como una nueva solicitud entre la lista de solicitudes del área.
5	Comprobar la integración entre el modulo de Atención de solicitudes y el modulo de Gestión de Proyectos	Seleccionar una solicitud previamente creada y escoger la opción convertir en proyecto. Verificar que al seleccionar esta opción la solicitud quede cerrada y se haya redireccionado al usuario al formulario de creación de proyecto.
6	Comprobar la integración entre el modulo de Gestión de Proyectos y el modulo de SCM	Crear un nuevo Proyecto, luego ingresar al modulo de SCM y verificar que el proyecto aparece entre la lista de proyectos del área y se puedan definir tareas de SCM para este.

Tabla 1.2: Test Integración

### 1.2.1. Análisis de Resultados

Esta pruebas fueron realizadas y superadas en su totalidad. Verificando así que existe una correcta integración de los módulos diseñados. Estas pruebas fueron ejecutadas manualmente dado la dificultad de automatizarlas.