Universidad de Valladolid

E.T.S Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática de Sistemas

Estudio comparativo de frameworks de testing sobre arquitectura J-everis

Autor:

Fernando Marín Sánchez

Tutor:

Valentín Cardeñoso Payo

Agradecimientos

Como siempre, mi más sincero agradecimiento a mis padres, hermano y abuelos, que sin su ayuda y motivación a lo largo de toda mi vida ahora no estaría aquí.

Agradecimientos también a mis compañeros del Equipo Rojo porque lo que he aprendido con ellos no se puede encontrar en los libros, a mis amigos y familiares que me han retado y motivado a seguir adelante, en especial a Mercre y Javi; a todos los miembros de la oficina de Everis Valladolid ya que de no ser por ellos este trabajo no habría visto la luz; a RITSI, AU y la delegación de alumnos de la ETSII que me han hecho formarme como persona y aprender de todos ellos; y a Clara, por aguantarme tanto en los días buenos como en los malos.

Finalmente también quería dedicar este agradecimiento también a Valentín, mi tutor, y a Marcelino, antiguo profesor del instituto, que sin su paciencia y aguante todo hubiera sido más difícil.

A todos vosotros muchas gracias por estar ahí.

Índice

Índice de tablas

1. Abstract

Este trabajo presenta un estudio sobre los diferentes frameworks[1] del lenguaje de programación Java orientados a la producción de pruebas[2], con el objetivo de descubrir el framework que cubra más cualidades de la tecnología utilizada en Everis llamada J-everis. En este sentido se buscan las capacidades de dichos frameworks que puedan ser más útiles, es decir, que faciliten el desarrollo y mejoren la utilidad de los tests en entornos empresariales. El estudio realizado es suficientemente amplio en número de frameworks evaluados y en criterios de evaluación como para proporcionar una aproximación realista a lo que podría ser un estudio profesional de mercado para los frameworks. Al final se muestra el framework seleccionado después de haber encontrado numerosos problemas a la hora de la selección de los criterios.

[1] También llamado ***infraestructura digital***, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de *software* concretos. Fuente [53]

[2] En este documento se han utilizado indistintamente los términos prueba y test.

2. Introducción

Durante años el desarrollo *software* ha planteado retos, retos que se han intentado solucionar con diferentes metodologías de trabajo para los equipos y las empresas. En este ámbito se han ido nuevas formas para intentar acortar los tiempos de desarrollo de trabajo y mejorar la productividad en los ciclos de vida, sobre todo en las etapas de mantenimiento, las cuales son cada vez más amplias y costosas tanto para las empresas como para los clientes.

A través de los modelos ágiles se intentan subsanar todos los problemas que pueden surgir en estos tiempos donde los requisitos cada vez son más ligeros, o dicho de otra manera, los cambios en los requisitos son a menudo algo normal en los contratos de desarrollos de *software*. La predisposición de los equipos por estar abiertos a los cambios de comportamiento del *software* hace que se necesite dar un enfoque diferente a las metodologías más "clásicas" donde los requisitos estaban prestablecidos y acordados al principio y no se volvían a modificar.

En este marco de estudio, el *Test Driven Development* (a partir de ahora TDD) gana importancia al ser una metodología práctica orientada al desarrollo y diseño de *software* orientado a objetos. Kent Beck, creador del TDD, explica [1] que el TDD es una implementación del *Extreme Programming* (XP) en el que se le da peso a las pruebas que validarán el código antes de crear el propio código, de esta forma no se condicionan las pruebas al código desarrollado y exclusivamente se centran en cumplir los requisitos y no se añaden funcionalidades. Tal y como dice él mismo, *"Nunca escribas una funcionalidad sin escribir primero una prueba que falle"*. Esto implica que se debe pensar antes la prueba o de otra manera "pensar" bien los requisitos que deben validar las funcionalidades.

El TDD no solo se basa en hacer pruebas unitarias durante todo el desarrollo, sino que también consiste en mantener todo el código probado mediante pruebas funcionales, pruebas de aceptación (ATDD) y pruebas de comportamiento (BDD). Todo esto se explicará más adelante en el caso de estudio.

Para adaptarse a estos nuevos tiempos con las metodologías que implican un rápido desarrollo y versatilidad, se necesita un entorno con unas herramientas que mejoren el alcance de los tests del código, faciliten su desarrollo y acorten los tiempos invertidos en esta etapa. Para ello, las empresas de *software* deben hacer hincapié en investigar y especializarse en el uso de herramientas que acorten los desarrollos y aseguren una calidad aún mayor del producto entregado. Es ahí donde entran los *frameworks* de *testing*, una serie de herramientas sobre determinados lenguajes de programación que agregan funcionalidades y simplifican el desarrollo de las pruebas en los entornos de *testing* tradicionales.

3. Estado del arte

El área de pruebas del desarrollo de *software* sobre Java ahora mismo está dominado por JUnit, pero hay *frameworks* que se están imponiendo junto a éste para ampliar en funcionalidad las carencias que tiene. *Frameworks* como son Cucumber o Mockito complementan especialmente bien a este otro.

En el mercado hay infinidad de *frameworks* de *testing* implementados sobre JUnit que pueden facilitar la ejecución de las pruebas sobre un proyecto, pero hay tal cantidad de ellos que es necesario establecer criterios de selección para que los más potentes y versátiles *frameworks* no se diluyan en el gran abanico de oferta. Hay que ver cual destaca más para generar una diferencia comparativa con respecto al resto de la competencia.

En este sentido J-everis destaca por ser un complemento en el entorno empresarial de Everis que facilita los desarrollos sobre Java y la arquitectura Java EE, por ello necesita ser puntero en todos los aspectos. Es aquí donde interviene este estudio que trata de detectar qué *framework* se adapta mejor a la manera de trabajar de la herramienta J-everis y pretende ser un punto de partida para la futura integración del ganador.

3.1. J-everis

En esta sección se va a ofrecer una idea de lo que es J-everis, *framework* sobre el que se va a hacer el estudio comparativo de los *frameworks* de *testing*.

La descripción de la arquitectura y todo el conjunto de imágenes usados en este apartado han sido extraídos de la documentación que se encuentra en el portal interno de Confluence de Everis sólo accesible por los trabajadores de la empresa [2]. Esta descripción de la arquitectura se ha realizado entre todos los miembros de Everis que van a presentar estudios referentes a J-everis y para la Universidad de Valladolid, en concreto se ha realizado junto a Israel González Aguayo y David Tejedor Zurdo.

3.1.1. Introducción a J-everis

J-everis es una arquitectura Java Empresarial creada, gestionada y mantenida por Everis, que proporciona cobertura a las actividades relacionadas con el diseño, construcción, implantación y mantenimiento de sistemas y aplicaciones.

Esta arquitectura lleva en desarrollo desde el 2011 cuando sufrió una restructuración por completo de la arquitectura empresarial anterior. El proyecto es desarrollado la Oficina Técnica de Everis.

Los principales objetivos que persigue el uso de este *framework* son:

- Ofrecer una arquitectura común de construcción de aplicaciones JavaEE.
- Proporcionar un entorno de trabajo, documentación, soporte y mantenimiento de sus componentes.
- Simplificar la complejidad inherente a JavaEE, ofreciendo un marco de referencia de trabajo.
- Proponer una solución alineada con los estándares y soluciones más utilizadas para la comunidad *Open Source*.
- Presentar una solución abierta que permita agregar e intercambiar cualquier pieza

con un coste reducido.

La arquitectura de J-everis se basa en la arquitectura MVC (Modelo Vista Controlador), donde existe un proceso de abstracción que permite dividir una aplicación en componentes lógicos con responsabilidades diferenciadas y que pueden ser desarrolladas incluso por diferentes roles de equipo.

3.1.2. Arquitectura de ejecución

La arquitectura de ejecución de J-everis es una arquitectura multicapa dividida de la siguiente forma:

- **Capa de presentación**: Responsable de la interacción con el usuario a través de una interfaz de usuario. Ésta, además, ofrece acceso a servicios de negocio que se encuentran en la capa de servicios. Sigue el patrón MVC, con el fin de desacoplar el diseño de la interfaz (vistas, flujos, etc.), del resto de funcionalidades.
- **Capa de negocio**: Contiene el modelo de la aplicación y todos sus servicios de negocio.
- **Capa de integración**: Ésta capa contiene el conjunto de funcionalidades que permiten a la aplicación integrarse con otras aplicaciones.
- **Capa de datos**: Contiene la interfaz para conectar la capa de negocios con la base de datos.
- **Funcionalidades transversales**: Ofrece un conjunto de funcionalidades al resto de capas, así como la gestión de excepciones, trazas y *logging*, seguridad, etc.

Una de las características más importantes de J-everis es su modularidad, es decir, el *framework* cuenta con un conjunto de módulos que se pueden agregar o eliminar a un proyecto J-everis, y cada uno de estos cuenta con un conjunto de funcionalidades que se adaptan perfectamente al proyecto ya existente pudiendo ampliar el proyecto de forma fácil y sin afectar al resto de funcionalidades.

Los módulos que se muestran en la Figura 1 son los módulos de la versión 3.2 de J-everis, aunque desde la Oficina Técnica de J-everis en Everis, se intenta ampliar el *framework* con las necesidades que van demandando a aquellos proyectos (iniciados o no) que usan o van a utilizar J-everis.

Para poder empezar a describir los módulos, es necesario dar a conocer que hay dos tipos de proyectos J-everis: J-everis Web y J-everis *Business*. El primero contiene la capa web, mientras que el segundo contiene toda la lógica de negocio y el acceso a base de datos.

También puede crearse únicamente una capa *business* que despliega *Web services*, para ser consumidos por clientes externos.

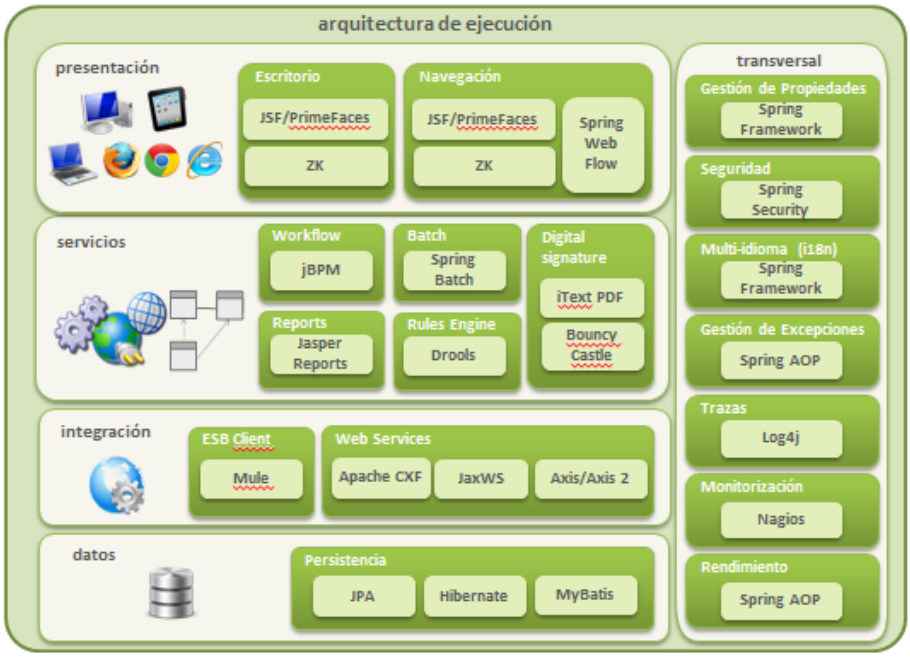

Figura 1 Esquema de los módulos de J-everis

3.1.2.1. Capa de presentación

La capa de presentación, tal y como se ha comentado anteriormente, es la responsable de la interacción con el usuario. Ésta, se constituye por dos partes, el escritorio y la navegación como se muestra en la Figura 2.

Figura 2 Esquema de los módulos de la capa de presentación de J-everis

- **Escritorio**: El término abstracto "Escritorio" es dado a la parte visual de la aplicación, es decir, aquello que va a ser mostrador al usuario. Las funcionalidades que maneja son: gestión de vistas, inicio de los flujos de navegación y hace uso de servicios como seguridad y i18n (internacionalización).
- **Navegación**: La parte de navegación es aquella que gestiona qué va a mostrarse al usuario después de que este realice una acción. Para ello, la tecnología que se utiliza en el *framework* es Spring Web Flow,. la cual permite definir flujos de navegación que determinan el conjunto de acciones que se realizarán y en qué orden.

3.1.2.2. Capa de negocio

La capa de negocio es la parte que contiene los servicios de la aplicación y el modelo.

En este caso, se delimitan cinco módulos diferentes e independientes, que podrán ser agregados a cualquier aplicación desarrollada con J-everis. Se pueden observar los componentes en la Figura 3.

Figura 3 Esquema de los módulos de la capa de servicios de J-everis

3.1.2.2.1. Módulo *batch*

El módulo de *batch* permite a la arquitectura disponer de una herramienta para crear procesos *batch*, es decir, aquellos que se ejecutan según programación de forma repetitiva o puntual en el momento de tiempo que se le defina. Este módulo está basado en el *framework* Spring Batch, que ofrece una manera más sencilla de lidiar con los procesos periódicos, proporcionando una interfaz que ofrece una mejor gestión del mantenimiento.

Además de ejecutar trabajos y tareas, es capaz de guardar la contabilidad de todo lo que ha pasado con estos trabajos y tareas en la base de datos, por lo que se tiene *logging/tracing* de todo lo que ha ido sucediendo con los trabajos y tareas en la base de datos.

3.1.2.2.2. Módulo de reportes

Mediante el módulo de reportes se añade al proyecto la capacidad de generar informes, con la característica de ser agnóstico de la tecnología de informes que se utilice. El desarrollador sólo debe encargarse del diseño de los informes con la tecnología que se haya seleccionado.

El *framework* J-everis ofrece una implementación con Jasper Reports, que da la posibilidad de una fácil creación de informes con un amplio abanico de formatos.

3.1.2.2.3. Módulo de la firma digital

El módulo de firma digital añade a una aplicación J-everis las funcionalidades necesarias para trabajar con la propia firma digital. Éstas son:

- **Firma en cliente:** Permite firmar cualquier texto o formulario web completo.
- **Acuse de recibo de la firma en cliente.**
- **Validación de *TimeStamp*:** El *TimeStamp* es un mecanismo que permite demostrar que una serie de datos han existido y no han sido alterados desde un instante específico de tiempo.
- **Validación de firma y extracción de información del certificado firmante.**
- **Validación de certificados.**

- **Firma de documentos en el servidor:** Mediante esta funcionalidad se pueden sellar archivos PDF en el servidor.
- **Sellado de tiempo.**

3.1.2.2.4. Módulo de *workflow*

El propósito del módulo de *workflow* es ofrecer una Interfaz de Programación de Aplicaciones [3] (a partir de ahora API, del inglés *Application Programming Interface*) genérica desacoplada de cualquier motor de *workflow*. Esta API ofrece los servicios básicos y comunes para los principales motores de *workflow* del mercado. Su utilización da total libertad para cambiar el proveedor de *workflow* sin apenas tener que hacer cambios en el código.

El objetivo del módulo de *workflow* contempla integrar cualquier motor BPM dentro de la arquitectura J-everis para la interactuación de los procesos de negocio dentro de las aplicaciones desarrolladas con esta arquitectura.

Este módulo tiene las siguientes propiedades:

- **Genérico:** Engloba acciones comunes a todos los motores de BPM.
- **Independiente:** No está sujeto a ninguna especificación propia de ningún motor de procesos BPM.
- **Alcance del ciclo de vida de las tareas humanas:** Los procesos BPM se caracterizan por tener tareas humanas o automáticas. Se abarca el proceso de ejecución de las tareas humanas.
- **Arranque de las instancias de procesos:** Puede iniciar instancias de procesos BPM de forma que un actor pueda a través de la capa de presentación realizar el arranque de un proceso BPM.

En el momento en el que se realiza este proyecto, la única implementación que se da del BPM es con jBPM, un motor *open source* que está perfectamente adaptado a la arquitectura de j- everis.

3.1.2.2.5. Módulo de motor de reglas

El propósito del módulo de motor de reglas es ofrecer una API genérica desacoplada de cualquier motor de reglas. Esta API ofrece los servicios básicos y comunes para los principales motores de reglas del mercado. Su utilización da total libertad para cambiar el proveedor sin apenas tener que hacer cambios en el código. Además se contempla la integración de cualquier motor dentro de la arquitectura J-everis para la ejecución de reglas de negocio.

El módulo cuenta con las siguientes propiedades:

- **Genérico:** Engloba acciones comunes a todos los motores de reglas.
- **Independiente:** No está sujeto a ninguna especificación propia de ningún motor de reglas.

En el momento en el que se realiza este proyecto, la única implementación que se da del motor de reglas es con *Drools*, totalmente compatible con la implementación del BPM. Hay que tener en cuenta que el uso de ambos módulos viene muchas veces ligado, pues dentro de un proceso de *workflow* pueden incluirse reglas definidas en el motor de reglas.

3.1.2.3. Capa de integración

El objetivo de las funcionalidades de la capa de integración es ofrecer servicios para conectar la aplicación que se desarrolle con otras aplicaciones ya existentes, bien sea ofreciendo algún tipo de servicio o bien consumiendo servicios ofrecidos desde otras aplicaciones. Se puede ver el conjunto de los módulos que ofrece en la Figura 4.

Figura 4 Esquema de los módulos de la capa de integración de J-everis

3.1.2.3.1. Módulo de servicios web

Los módulos de *Web services* permiten a la arquitectura publicar o consumir servicios web. En la arquitectura J-everis, existen dos módulos de *Web services*: publicación y consumo.

- **Publicación de *Web services***: Este módulo permite publicar servicios web a partir de clases anotadas o mediante la configuración de los ficheros de Spring. Con este fin se utiliza Apache CXF, un *framework Open source* Se ha escogido éste por su soporte a muchos estándares de *Web services* como SOAP, WS-Addressing, WS-Policy, WS-Security, entre otros.
- **Consumo de *Web services***: Mediante el módulo de consumo de *Web services* se añade a la arquitectura la posibilidad de utilizar (consumir) servicios web ya existentes y publicados por otras aplicaciones. Gracias a las funcionalidades ya ofrecidas por el módulo, el desarrollador no deberá trabajar en cuestiones complejas como pueden ser el establecimiento de conexiones seguras (SSL) o la conexión a través de un servidor Proxy.

En el momento de realizar este proyecto, J-everis ofrece implementaciones de consumo de servicios web con las tecnologías *JAX-WS (RI), Apache CXF, AXIS* y *AXIS2,* y es gracias a estas implementaciones que el consumo de servicios web pasa a ser una tarea sencilla, pues el desarrollador apenas es consciente de la tecnología que está utilizando.

3.1.2.3.2. Módulo de integración ESB

El módulo de integración de Bus de Servicios de Empresa (a partir de ahora ESB, del inglés *Entreprise Service Bus*) consiste en un combinado de arquitectura *software* que proporciona servicios fundamentales para arquitecturas complejas a través de un sistema de mensajes, que forman el bus, basado en las normas y que responde a eventos. Así pues se puede entender que un ESB proporciona una capa de abstracción construida sobre una implementación de un sistema de mensajes de empresa que permita a los expertos en integración explotar el valor del envío de mensajes sin tener que escribir código. Se puede observar una descripción del funcionamiento en la Figura 5.

El propósito del módulo de ESB de J-everis es ofrecer una API genérica desacoplada de cualquier bus de datos. Esta API ofrece los servicios básicos y comunes de comunicación con buses. Su utilización da libertad para cambiar de tecnología sin apenas tener que hacer cambios en el código. De esta forma, el módulo permite:

- Enviar mensajes desde la aplicación a través del bus de manera síncrona y asíncrona.
- Solicitar mensajes a través del bus.

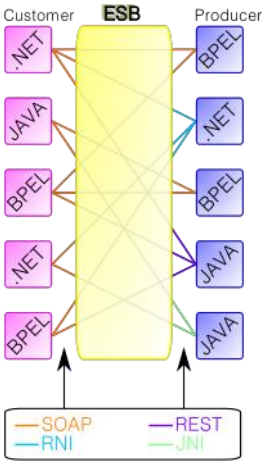

Figura 5 Esquema ESB

-

3.1.2.4. Capa de datos

La capa de datos añade al *framework* las funcionalidades necesarias para que haya interacción entre la aplicación desarrollada y la base de datos que se vaya a utilizar. Para ello disponemos del módulo de persistencia. La Figura 6 muestra los componentes disponibles para el módulo de persistencia.

Figura 6 Esquema de los módulos de la capa de datos de J-everis

3.1.2.4.1. Módulo de persistencia

El módulo de persistencia añade a la aplicación el conjunto de funcionalidades para que se puedan hacer consultas a la base de datos. Este módulo consta de distintas capas para facilitar la modularización, flexibilidad e incorporación de cualquier tipo de tecnología dependiendo de las necesidades de la aplicación.

El primero de los componentes es el que recibe el nombre de *Core*, que contiene una interfaz base con las operaciones básicas de cualquier servicio de persistencia además de servicios transversales comunes asociados al servicio de persistencia. Este módulo posee las siguientes características:

- **Interfaz base de persistencia:** proporciona las operaciones básicas comunes asociadas al servicio de persistencia.
- **Configuración de transaccionalidad:** configura la transaccionalidad mediante pointcups de Spring AOP para las operaciones definidas en el repositorio genérico. Se puede definir el tipo de transaccionalidad a aplicar para cada tipo de operación.
- **Traducción de excepciones:** Mediante Spring AOP se capturan las excepciones que se puedan producir al ejecutar los métodos del repositorio genérico y se traducen a excepciones J-everis internacionalizadas para su mejor comprensión.
- **Auditoria SQL:** Como componente añadido se ha desarrollado una funcionalidad para poder auditar y sacar estadísticas de las consultas a la base de datos.

El *framework* J-everis consta de implementaciones de persistencia con las tecnologías *JPA, Hibernate* y *MyBatis*, pero gracias a la arquitectura existente, añadir una nueva tecnología de persistencia pasa a ser un proceso fácil. El árbol de implementaciones se observa en la Figura 7.

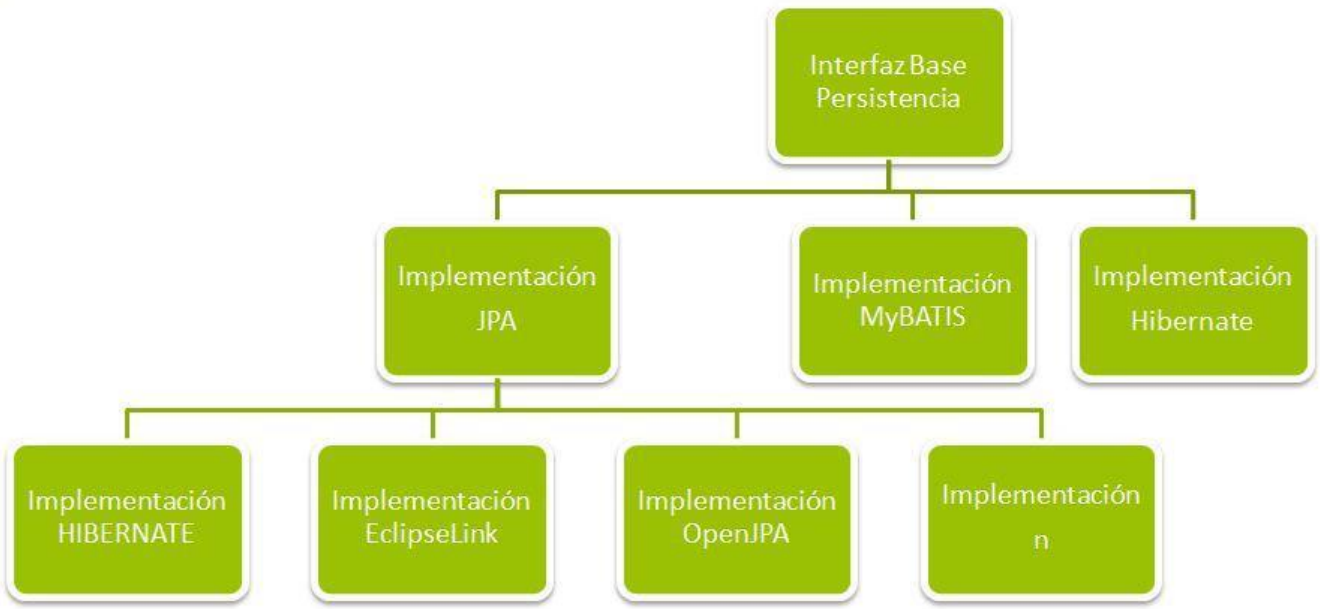

Figura 7 Esquema de la estructura del módulo de persistencia de J-everis

3.1.2.5. Funcionalidades transversales

Las funcionalidades transversales de J-everis son aquellas que están presentes en cualquier proyecto creado con el *framework*, de forma que se pueden utilizar siempre y en todos los módulos que se vayan agregando a la aplicación. Los módulos disponibles aparecen en la Figura 8.

Figura 8 Módulos de la capa transversal de J-everis

3.1.2.5.1. Módulo de seguridad

El módulo de seguridad tiene como propósito principal gestionar la autenticación y autorización de los usuarios de las aplicaciones desarrolladas con el *framework* J-everis. La autenticación permite comprobar que el usuario es quien dice ser, mientras que con la autorización se comprueba que el usuario tenga permiso para acceder al recurso que solicita.

Para el desarrollo de este módulo se ha utilizado Spring Security, el cual nos permite definir de una forma sencilla roles de usuario, a los que se les puede asignar una serie de permisos de acceso a los diferentes recursos de la aplicación. Además, puesto que pertenece al *framework* de Spring, es compatible con el resto de componentes.

3.1.2.5.2. Módulo de monitorización

El módulo de monitorización dota a las aplicaciones de la capacidad de ser monitorizadas a través de un sistema externo mediante JMX (*Java Management Extensions*), tecnología que aporta herramientas para manejar y monitorizar aplicaciones. El módulo desarrollado para J-everis es independiente del sistema de monitorización que se quiera utilizar, aunque será necesario agregar la implementación para el sistema escogido por parte de las aplicaciones.

El *framework* J-everis, consta de una implementación para un sistema de monitorización llamado *Nagios*. Esta implementación cubre los siguientes aspectos:

- Monitorización de arquitectura
  - Estado y tiempo de respuesta de los diferentes backends a los que se tiene acceso desde la aplicación. Ésta misma realiza la monitorización de su conectividad.
  - Listado de EJBs visibles desde la aplicación.

- Monitorización de la aplicación: Hace referencia a las conexiones externas a la aplicación, así como la conexión a base de datos.
    - Número de sesiones en la aplicación.
    - Número de usuarios autenticados conectados a la aplicación.
    - Número de excepciones producidas en la aplicación.

3.1.2.5.3. Módulo de rendimiento

El objetivo principal del módulo de rendimiento es servir de ayuda en el desarrollo de aplicaciones, permitiendo identificar los métodos más costosos en cuanto a tiempo de respuesta. De esta forma, la aplicación va guardando los tiempos de retardo que cuesta la realización de una función y el tiempo de respuesta de la base de datos, de forma que estos tiempos pueden ser listados.

3.1.3. Arquitectura de desarrollo

La arquitectura de desarrollo de J-everis pretende ayudar al desarrollador a crear y desarrollar aplicaciones basadas en el *framework* J-everis. Para ello, el entorno de trabajo que se sugiere consta del uso de las siguientes herramientas:

- Eclipse Indigo
- Maven
- Tomcat

En la web de J-everis se ofrece un entorno ya configurado con las herramientas mencionadas, de forma que el desarrollador solo tiene que descargarlo, instalarlo y empezar a programar.

Aunque se aconseja el uso del servidor de aplicaciones Tomcat, el *framework* J-everis es compatible con JBoss, Weblogic, WebSphere y Glassfish.

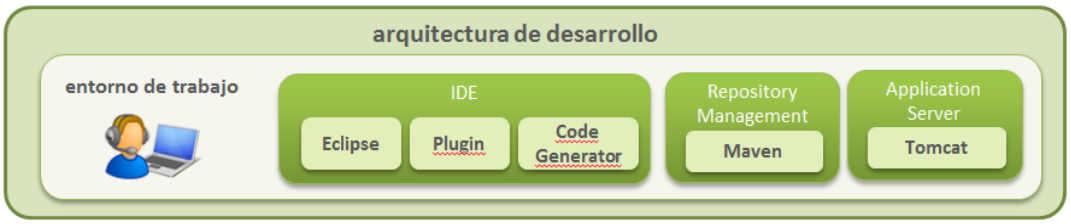

Figura 9 Esquema de la arquitectura de desarrollo de J-everis

3.1.3.1. *Plug-in* para Eclipse

El uso del IDE Eclipse, viene ligado al uso del plug-in de J-everis desarrollado para Eclipse. Mediante este plug-in, se pueden crear proyectos J-everis, agregar o eliminar módulos y filtros a un proyecto J-everis, y utilizar el generador de código que se verá más adelante. La ventaja del uso de este módulo es que para cada una de sus funcionalidades da una configuración válida, de forma que si se agrega un módulo, el proyecto quedará configurado y seguirá siendo usable aunque no se dé una configuración adaptada a la aplicación. De forma similar, al eliminar un módulo el proyecto quedará consistente y se eliminaran tanto las dependencias como los archivos de configuración que estaban únicamente relacionados con el módulo eliminado.

3.1.3.2. Generador de código

El objetivo de la generación de código es simplificar los desarrollos y aumentar la productividad de los proyectos, permitiendo generar automáticamente el código correspondiente al modelo de dominio definido en la aplicación. La herramienta ha sido desarrollada con el *framework Xtext*.

Así pues, para realizar un uso correcto de ésta herramienta, se deben realizar dos acciones:

- Definir el modelo de dominio de la aplicación: Representar las entidades de la aplicación y las relaciones entre ellas.
- Generar el código: A partir de la definición del modelo que se haya realizado se genera el código necesario para la aplicación, aunque éste puede que necesite ser editado. El código que generado consta de:
    - Clases base de las entidades y las interfaces de cada una de ellas.
    - Repositorios y servicios para cada una de las entidades.
    - Propiedades de internacionalización de la aplicación.
    - Vistas de la aplicación.
    - Flujos de la aplicación.

4. Estudio comparativo

En este estudio se ha intentado encontrar todos los *frameworks* de *testing* sobre Java que hay en el mercado, seguramente alguno se haya pasado por el hecho de que es muy difícil encontrar todos los que hay actualmente en el mercado pero el número de *frameworks* a evaluar es un conjunto suficientemente amplio como para dar un resultado válido al estudio realizado.

El objetivo de este estudio ha sido buscar de entre todos los *frameworks* que hay y ha habido, al que mejor se adapte a las cualidades que necesita J-everis, así como el que mejor cubre las mayores necesidades que suelen tener las aplicaciones empresariales creadas sobre Java y la arquitectura Java EE.

4.1. Listado, descripción y primera evaluación de los *frameworks*

En este apartado se listan los *frameworks* encontrados, así como se describen y puntúan en función a unos criterios establecidos. Estos criterios son los que aparecen a continuación.

Se ha partido de una visión de la arquitectura de una aplicación de tres capas para generar criterios de selección. Así como también se han establecido los criterios con los típicos usos que puede tener un *framework* de *testing* en su uso con TDD.

A esta altura del estudio solo se va a utilizar en un principio "Si", "No" o "-" (si no aplica) a cada ámbito evaluado de los *frameworks*. Posteriormente se contará sobre cada *frameworks* el número de apartados a los que se responde favorablemente al contenido.

Por ejemplo:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
Ejemplo	Si	Si	Si	No	Si	No	Si	Si	6

Tabla 1Ejemplo de primer análisis de características.

En función de los apartados en los que tenga un "Si", se hace una suma de ellos y se establece la primera nota de la evaluación. En este caso al tener 6 "Si" se establece una nota de 6 en la puntuación del primer filtro que se ha realizado sobre los *frameworks*.

Los criterios de estudio en este caso son 8 que se pueden dividir en 3 y 5 tal y como se ha descrito anteriormente que iban. Éstos son:

- **Arquitectura**:
  - **Capa Visual**: se contempla si el *framework* tiene funcionalidades para ejecutar tests sobre los componentes de la capa tales como ficheros HTML, que contengan web drivers o que tengan
  - **Capa Negocio**: los tests más básicos se suelen realizar sobre esta capa, donde se encuentra toda la lógica de negocio, no se necesitan muchas funcionalidades en los *frameworks* para poder realizar pruebas en ésta.
  - **Capa Datos**: se evalúa si el *framework* es capaz de hacer pruebas sobre la capa de acceso a datos con integración con diferentes drivers para gestores

de bases de datos.

- **Tipo de uso en TDD**:
  - **BDD** [4]: *behavior driven development* (BDD en adelante) o desarrollo guiado por el comportamiento un proceso que amplía las ideas de TDD. Aquí no se prueban clases, se prueban escenarios y el comportamiento de las clases a la hora de cumplir dichos escenarios, los cuales pueden estar compuestos de varias clases. De manera que los escenarios son programados por metalenguajes donde se mapea las pruebas a hacer desde los descriptores.
  - **ATDD** [5]: *acceptance test driven development* (ATDD en adelante) o desarrollo de pruebas de aceptación. Es la funcionalidad destinada a aquellos tests que determinan si los requisitos de cierta funcionalidad han sido cumplidos, como usuarios finales, no tienen por qué preocuparse por los detalles de la implementación, sino que deberían poder centrarse más en que la parte funcional cumpla los requisitos (en función de datos de entrada se prueba si el resultado es el esperado).
  - **Mock** [6]: la traducción textual es Objeto Simulado, *frameworks* de mocking sirven para simular funcionalidades que no están implementadas o no se necesitan integrar, centrándose así las pruebas en los objetos que realmente se quieren probar.
  - **Tests unitarios** [7]: son pruebas que se ejecutan sobre objetos para comprobar el correcto funcionamiento de cada método de éste. Las pruebas unitarias deben ser automatizables, completas (que cubran todo el código a probar), repetibles, independientes y que cuenten con la misma documentación que el código del proyecto.
  - **Tests funcionales**: las pruebas funcionales se basan en comprobar el funcionamiento de la aplicación en cada momento y que todo sistema funcione conforme a los requisitos en cada iteración, agregando funcionalidades nuevas el sistema tiene que probar todo el conjunto de pruebas anterior y verificar que nada falle.

A continuación se enumeran y describen brevemente los *frameworks* que han sido puestos a estudio de este trabajo así como todos los criterios y sus calificaciones explicados anteriormente:

4.1.1. Arquillian

Arquillian [8] presume de ser uno de los mejores *frameworks* de *testing* del momento, ya que es muy usado y tiene una gran cantidad de *plug-ins*. Tal y como dice en su página principal:

"Arquillian is an innovative and highly extensible testing platform for the JVM that enables developers to easily create automated integration, functional and acceptance tests for Java middleware." [9]

Lo que quiere decir que está diseñado para ser reutilizado y extendido para su uso además de la generación de más herramientas derivadas que puedan contribuir a que este *framework* sea muy poderoso tanto en tests de integración, como funcionales o de aceptación.

Las principales características con las que cuenta este *framework* son:

- Escribir tests reales: lo que para ellos significa no tener que hacer uso de mocks debido a que proporcionan la capacidad de inyección de dependencias.
- Amigable con el IDE: se integra perfectamente con las herramientas preexistentes de JUnit para los IDEs.
- Enriquecimiento de los tests: gracias al número de anotaciones, se pueden crear tests con más funcionalidades.
- Controlar el navegador: mediante la integración con diferentes web-drivers.
- Independiente del contenedor

Este *framework* cuenta además con una considerable cantidad de *plug-ins* añadidos que pueden subsanar algunas carencias que pueda llegar a tener.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Tests Unitarios	Tests Funcionales	Nota
Arquillian	Si	Si	Si	Si	Si	Si	Si	Si	8

Tabla 2 Arquillian

4.1.2. BeanSpec

BeanSpec [10] está orientado a la programación mediante la metodología BDD y de una manera muy básica, permite hacer comprobaciones del funcionamiento de una aplicación de una manera más "narrativa" al interpretar diferentes frases en forma de comportamiento de un objeto o clase.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
BeanSpec	No	No	No	Si	No	No	No	No	1

Tabla 3 BeanSpec

4.1.3. Cactus

Cactus [11] es un *framework* perteneciente al proyecto Jakarta de Apache que intenta simplificar el desarrollo de los tests del lado del servidor haciendo uso de JUnit y extendiéndolo.

El ecosistema que aporta Cactus contiene muchas herramientas diferentes que se separan en:

- El core del *framework*: la API que aporta todo lo esencial al *framework*.
- Los módulos de integración: *frameworks* que a su vez hacen más sencillo al propio Cactus (*Scripts* de Ant, *plug-in* de Eclipse…)

- Ejemplos del uso de Cactus.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
Cactus	Si	Si	No	No	Si	Si	Si	Si	6

Tabla 4 Cactus

4.1.4. Concordion

Concordion [12] se describe como una herramienta para escribir tests de aceptación automatizados en Java, lo que quiere decir que se generan los tests a partir de la descripción de los casos de uso.

Las características especiales de este *framework* son:

- Generas los tests en HTML para que puedan ser utilizados como posterior documentación y favorecer la lectura de los programas.
- Facilita la ayuda para la separación entre el "cómo" del "qué" de un programa. Es decir, ayuda a poder diferenciar cómo deben funcionar los tests de cuál es su funcionamiento concreto en código.
- Posee un pequeño conjunto de comandos sencillamente para que la curva de aprendizaje sea pequeña.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
Concordion	No	Si	No	No	Si	No	Si	No	3

Tabla 5 Concordion

4.1.5. Concutest

Concutest [13] es un *framework* de desarrollo para generar tests para programas concurrentes.

Este tipo de *frameworks* vienen motivados por el hecho de que las típicas herramientas de test no sirven para el multihilo ni la ejecución programada.

Este *framework* se basa en la unión de tres herramientas que se complementan entre sí:

- ConcJUnit: una extensión de JUnit para el soporte del aislamiento de errores entre tests de múltiples hilos.
- Thread Checker: un lenguaje de anotaciones sobre java para especificar los invariantes de las clases de los hilos.

- Schedule-Based Execution: un *framework* de *testing* para planificar la ejecución de los tests.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
Concutest	No	Si	No	No	No	No	Si	Si	3

Tabla 6 Concutest

4.1.6. Cucumber JVM

Cucumber JVM [14] es una implementación para la Máquina Virtual Java (JVM) que sirve para hacer BDD y escribir tests como si se escribieran los casos de uso.

El método para usar este *framework* es: describir el comportamiento en texto plano (con una nomenclatura determinada), escribir la definición de cómo se va a interpretar en cada comportamiento y luego escribir el código que realice las definiciones anteriormente escritas.

La implementación para la JVM cuenta con soporte para multitud de lenguajes como Scala, Groovy,…además de incorporar las funcionalidades de los *frameworks* de *testing* Needle y Arquillian. Esto hace a Cucumber más completo en los apartados donde no era significativo el uso del BDD.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
Cucumber JVM	Si	Si	Si	Si	Si	Si	Si	Si	8

Tabla 7 Cucumber JVM

4.1.7. DbUnit

DbUnit [15] es una extensión de JUnit enfocado a los proyectos que dependen de bases de datos. Este *framework* evita que éstas se corrompan por ejecutar un test que funcione mal.

Su objetivo es dejar la base de datos en estados conocidos antes y después de lanzar los tests. También tiene la capacidad de exportar e importar conjuntos de datos de XML y verificar si los datos de entrada serán válidos para la base de datos.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
DbUnit	No	Si	Si	No	Si	No	Si	No	4

Tabla 8 DbUnit

4.1.8. EasyMock

EasyMock [16] es un *framework* de mocking cuya principal característica es que puede generar los mocks de forma dinámica mediante patrón proxy que provee Java.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
EasyMock	-	-	-	No	No	Si	No	No	1

Tabla 9 EasyMock

4.1.9. FitNesse

FitNesse [17] es un *framework* de generación de tests de aceptación. La generación de estos tests y los casos de prueba se realizan mediante una wiki. La wiki en determinados casos hace de IDE.

Se pueden realizar todo tipo de casos de prueba como pueden ser comprobación de entradas o también comprobar el comportamiento con pruebas secuenciales dependientes.

Mediante el motor que tiene este *framework*, las páginas wiki existentes se transforman en los casos de prueba a través de llamadas a la API.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
FitNesse	Si	Si	Si	No	Si	No	Si	Si	6

Tabla 10 FitNesse

4.1.10. GrandTestAuto

GTA [18] es un *framework* que proporciona a JUnit la funcionalidades a mayores como la de comprobar que todos los métodos de un proyecto están cubiertos con clases de

prueba o comprobar que los tests de otros sistemas se han lanzado. También sirve para lanzar pruebas de tests automatizadas.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
GrandTestAuto	No	Si	No	No	No	No	Si	No	2

Tabla 11 GrandTestAuto

4.1.11. GroboUtils

GroboUtils [19] es un conjunto de utilidades dedicadas a mejorar la experiencia de JUnit. Este *framework* contiene un conjunto de proyectos entre los que destacan:

- Proyecto de uso de múltiples hilos en paralelo.
- Proyecto de comprobación de la cobertura de código.

También hay muchos más sobre tests de proyectos de generación de XML o interfaces.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
GroboUtils	No	Si	No	No	No	No	Si	No	2

Tabla 12 GroboUtils

4.1.12. HavaRunner

HavaRunner [20] es un *framework* de *testing* se focaliza en soportar concurrencia y tener mucha sencillez en la configuración.

La arquitectura de la API de tests concurrentes está creada con Scala, aportándole más rapidez a la hora de correr los tests.

Las principales capacidades de HavaRunner son:

- Velocidad de ejecución de los tests.
- Agrupación de los tests en Suites mediante anotaciones.
- Generación de tests con atributo final.
- No hay bloqueo debido a la ejecución asíncrona.
- Soporte de JUnit.

El objetivo de este *framework* por lo tanto es probar proyectos de una manera rápida y de forma muy organizada.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
HavaRunner	No	Si	No	No	No	No	Si	No	2

Tabla 13 HavaRunner

4.1.13. Instinct

Instinct [21] *framework* está orientado a la programación mediante la metodología BDD. A diferencia de otros, crea los escenarios con los comportamientos directamente desde las clases de código Java.

Este *framework* provee de anotaciones para contextos, especificaciones, actores y roles. Tiene total integración con JUnit y además cuenta con un *plug-in* para InteliJ IDEA.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
Instinct	No	Si	No	Si	No	No	Si	No	3

Tabla 14 Instinct

4.1.14. JBehave

JBehave [22] es un *framework* destinado a la metodología BDD.

El ciclo de uso de este *framework* está orientado a ser completamente fiel a la programación con BDD.

Primero se escribe la historieta en código plano, después se mapea la historieta a clases de Java, luego se configura como se va a ejecutar la historieta y finalmente se ejecuta la historieta para generar archivos de reporte en HTML.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
JBehave	No	Si	No	Si	No	No	Si	No	3

Tabla 15 JBehave

4.1.15. JDave

JDave [23] es otro *framework* para el uso de la metodología BDD. Este es un *framework* que aglutina diferentes tecnologías para ser lo más completo posible siguiendo la mencionada metodología.

Los componentes que integran JDave son JMock 2 como *framework* de mocking y Hamcrest como biblioteca *matching* (comprobación de semejanzas entre cadenas de caracteres). Estos dos componentes se unen a la API de JDave para generar un *framework* que facilita la claridad de código de lo que se escribe en las clases de prueba.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
JDave	No	Si	No	Si	No	No	Si	No	3

Tabla 16 JDave

4.1.16. JExample

JExample [24] *framework* está destinado a generar relaciones productor-consumidor dentro de las clases de prueba de JUnit.

Se basa en generar dependencias entre los tests mediante anotaciones, así los resultados de un test se inyectan dentro de los consumidores de esa prueba pasándose como parámetros; mediante estas funcionalidades JExample crea una visión diferente de los tests unitarios donde el aislamiento entre los tests no tiene ninguna importancia ya que potencia todo lo contrario.

JExample cuenta con la total integración con JUnit y también con un *plug-in* para Eclipse.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
JExample	No	Si	No	No	No	No	Si	No	2

Tabla 17 JExample

4.1.17. JMeter

JMeter [25] es un *software* para el desarrollo de pruebas funcionales y de carga para servidores web y aplicaciones de todo tipo pero que cuenta con muchas funcionalidades añadidas.

Este *software* se integra con JUnit para realizar pruebas unitarias, también en remoto, conectándose a los proyectos. Puede realizar *testing* distribuido de aplicaciones como otra función/cosa.

Sus características principales son:

- Capacidad de probar múltiples protocolos (HTTP, SOAP, FTP, JDBC, SMTP…).
- Completamente en Java.
- Soporte multihilo para generar pruebas concurrentes en hilos diferentes.
- Interfaz propia.
- Core extensible y múltiples *plug-ins* para la aplicación.

Todas estas características hacen a JMeter un *software* muy potente para los desarrollos guiados por pruebas.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
JMeter	Si	No	Si	No	No	Si	No	Si	4

Tabla 18 JMeter

4.1.18. JMock

JMock [26] es un *framework* para la realización de mocks en las pruebas de desarrollo. Permite la definición de mocks de manera muy sencilla y determinar las interacciones entre los objetos.

Este *framework* se adapta bien a las habilidades de refactorización que tienen los IDEs y puede ser agregado fácilmente a otro *framework* de *testing*.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
JMock	-	-	-	No	No	Si	No	No	1

Tabla 19 JMock

4.1.19. JMockit

JMockit [27] es un *framework* de mocking que presume de ser el *framework* de este tipo más completo que hay para Java [28].

Sus principales características son:

- Generación de mocks de clases static, final y constructores.
- Generación de mocks de enums, anotaciones e interfaces.
- Matching de argumentos.
- API consistente.

Destaca por sus múltiples funcionalidades que puede aportar a la generación de mocks e integración con los *frameworks* de *testing*.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
JMockit	-	-	-	No	No	Si	No	No	1

Tabla 20 JMockit

4.1.20. Jnario

Jnario [29] es un *framework* de BDD que destaca por tener su propio metalenguaje de programación sobre Java.

El funcionamiento de este *framework* es similar al de Cucumber, tal y como dicen en su página, pero con menos necesidades de mantenimiento del código ejecutable.

Las principales características de Jnario son:

- Los tests generados a través de las descripciones se pueden ejecutar directamente mediante JUnit.
- Jnario tiene soporte para Xtend, un generador de código Java a partir de metalenguaje de programación.
- Posibilidad de generar las pruebas mediante matrices.

Este *framework* aporta sencillez a la hora de generar los tests de manera descriptiva para una documentación más fácilmente legible.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
Jnario	Si	Si	No	Si	Si	No	Si	No	5

Tabla 21 Jnario

4.1.21. JSST

JSST (Java Server-side *Testing framework*) [30] es un *framework* de *testing* que dice ser similar a Apache CACTUS (visto anteriormente) solo que éste no puede ser usado con cualquier otro *framework*, solo están soportados TestNG y JUnit4.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
JSST	No	Si	No	No	No	No	Si	No	2

Tabla 22 JSST

4.1.22. JTest

JTest [31] es un *software* propietario de pago para el análisis de aplicaciones, control de calidad de código y generación de código de tests.

Las principales cualidades de este *framework* de *testing* son:

- Tiene soporte para tecnologías Spring, Hibernate, JSF, Struts, JSPs y más.
- Posee plantillas para realizar pruebas de seguridad mediante estándares como OWASP.
- Provee un *plug-in* para Eclipse a la generación código, tanto de tests unitarios como pruebas funcionales.
- Creación automática de tests regresivos que se ejecutan solos en el IDE para controlar que los cambios incrementales no generen fallos en el funcionamiento global ni daños colaterales.
- Generación de código extendido de JUnit y Cactus.

JTest es un *framework* bastante completo con todo tipo de mejoras para generar pruebas con alto valor funcional y práctico para proyectos complejos.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
JTest[3]	No	Si	No	No	No	No	Si	No	2

Tabla 23 JTest

[3] **El color de las entradas de este framework es diferente debido a que es de pago, se ha querido hacer así para reflejar una diferencia entre él y el resto.*

4.1.23. Jukito

Jukito [32] es un *framework* de *testing* que engloba las características de JUnit, Guice y Mokito. JUnit, como se explica más adelante, es el *framework* base para las pruebas unitarias. Guice [33] es una API de Google para gestionar la inyección de dependencias. Finalmente Mockito es un *framework* de mocking, el cual se verá más adelante.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
Jukito	No	Si	No	No	No	Si	Si	No	3

Tabla 24 Jukito

4.1.24. JUnit

JUnit [34] es la base de prácticamente todos los *frameworks* de *testing*.

Se usa para realizar pruebas unitarias de código. Lleva en continuo desarrollo desde 2002 y desde entonces ha sido utilizado como base para extender su funcionalidad en diversos entornos.

Este *framework* es una implementación de la arquitectura tipo xUnit de *frameworks* de *testing* para la JVM.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
JUnit	No	Si	No	No	No	No	Si	No	2

Tabla 25 JUnit

4.1.25. JUnitEE

JUnitEE [35] es una extensión de JUnit que abre la posibilidad a realizar pruebas unitarias para que sean lanzadas como servlets dentro del contenedor en el que se despliega la aplicación principal.

Este *framework* otorga la facilidad de que puedan ser lanzados los casos de prueba mediante una interfaz web y así se puedan introducir los valores de los casos de prueba a ejecutar manualmente. Los tests son empaquetados para que puedan ser desplegados junto con la aplicación que se está probando.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
JUnitEE	No	Si	No	No	No	No	Si	Si	3

Tabla 26 JUnitEE

4.1.26. JWalk

JWalk [36] es una herramienta para generar pruebas automatizadas bajo el paradigma llamado *Lazy Systematic Unit Testing* [37], basado en la habilidad de poder recuperar las especificaciones internas de una clase dinámicamente mediante análisis exhaustivo de ella y todos sus métodos.

Los test se pueden realizar de manera automática o manual, y la herramienta genera código en función de los protocolos y comunicaciones que tienen las clases de la aplicación a probar.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
JWalk	No	Si	No	No	No	No	Si	No	2

Tabla 27 JWalk

4.1.27. Mockito

Mockito [38] es un *framework* de mocking para Java que ofrece una API muy sencilla e intuitiva. Integra Hamcrest como *framework* de *matching.* Este *framework* tiene una sintaxis muy similar a EasyMock.

Las características más reseñables que tiene este *framework* son:

- Poder hacer mocking de clases e interfaces.
- Azúcar sintáctico en forma de anotaciones.
- Simple verificación de errores.
- Integra Hamcrest como *framework* de matching pero también permite crear patrones de matching propios.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota

Mockito	-	-	-	No	No	Si	No	No	1

Tabla 28 Mockito

4.1.28. Mockrunner

Mockrunner [39] es un *framework* de mocking orientado al entorno J2EE. Es compatible con tecnologías como servlets, taglibs o Struts. Utilizando la API se pueden probar actions, servlet y filtros sin necesidad de tener que leer el web.xml o struts.xml.

Éste *framework* utiliza varias tecnologías de Java, en concreto JDBC y JMS.

- JDBC para simular la conexión a base de datos. Aunque no se puedan ejecutar sentencias SQL directamente, se pueden especificar los resultados que devolvería QL en función de diferentes consultas.
- JMS para simular la conexión con un servidor de mail. Se pueden recuperar los mensajes mediante texto plano y Java beans.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
Mockrunner	-	-	-	No	No	Si	No	No	1

Tabla 29 Mockrunner

4.1.29. Needle

Needle [40] es un *framework* de *testing* ligero para componentes de la arquitectura Java EE que permite la realización de las pruebas fuera de cualquier contenedor de aplicaciones. Este *framework* analiza las dependencias entre las clases e inyecta automáticamente los objetos de mock necesarios

Las características principales con las que cuenta Needle son:

- Azúcar sintáctico añadido mediante anotaciones (e.j.: @ObjectUnderTest)
- Inyección de constructores, métodos y campos por defecto.
- Inyección de objetos de mock por defecto.
- *Testing* de la capa de persistencia mediante JPA.
- Utilidades de transacción.
- Posee utilidades de reflexión.
- Puede ser utilizado con JUnit o TestNG
- Soporta a mayores EasyMock y Mockito.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota

Needle	No	Si	Si	No	Si	Si	Si	No	5

Tabla 30 Needle

4.1.30. NU Tester

NU Tester [41] es un *framework* muy básico de *testing*, para aquellas personas que se están iniciando en Java. Este *framework* cuenta con unas funcionalidades limitadas frente a JUnit, lo que le hace muy fácil y sencillo de usar y entender.

Está orientado para el ámbito académico, donde los alumnos aún no conocen el sentido de los Objetos, interfaces y relaciones de herencia complejas.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
NU Tester	No	Si	No	No	No	No	Si	No	2

Tabla 31 NU Tester

4.1.31. Pax EXAM

Pax Exam [42] es un *framework* de *testing* que sirve para diferentes arquitecturas de ejecución basadas en Java.

Los modos de funcionamiento de este *frameworks* son:

- Modo OSGi: provee soporte para el *framework* OSGi otorgando opciones de configuración para éste.
- Modo Java EE: ejecuta tests para los servidores de aplicaciones que dan soporte a la arquitectura de Java EE.
- Modo CDI [43]: otorga funcionalidades de inyección de contextos para java cuando no se requiere completamente la arquitectura de ejecución de Java EE.
- Modo Web: este modo funciona para contenedores de servlets tales como Tomcat o Jetty. También se puede usar con CDI y con la inyección de dependencias de Spring.

Este *framework* funciona tanto con JUnit 4 como con TestNG 6.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
Pax EXAM	No	Si	No	No	Si	No	Si	Si	4

Tabla 32 Pax EXAM

4.1.32. PowerMock

PowerMock [44] es un *framework* de mocking para Java que está enfocado a resolver problemas que normalmente son muy difíciles de probar y reducir la necesidad de utilización de inyección de dependencias en los tests.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
PowerMock	-	-	-	No	No	Si	No	No	1

Tabla 33 PowerMock

4.1.33. Selenium

Selenium [45] es una herramienta de pruebas de *software* orientado a las aplicaciones web. Tiene la capacidad de generar pruebas funcionales automatizadas.

Este *framework* dispone de una API para java que hace llamadas mediante un web driver al cliente.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
Selenium	Si	No	No	No	No	No	No	Si	2

Tabla 34 Selenium

4.1.34. SpryTest

SpryTest [46] es un *plug-in* para Eclipse que sirve de motor de generación de tests para Java, basado en JUnit.

Las características más destacadas de este *plug-in* son:

- Crea tests automáticamente.
- Permite configurar mocks para reducir las dependencias externas
- Potente API para realizar tests regresivos.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
SpryTest	No	Si	No	No	Si	Si	Si	No	4

Tabla 35 SpryTest

4.1.35. Sureassert UC

Sureassert [47] es un *plug-in* para Eclipse que genera código de pruebas sobre proyectos. Este *plug-in* es un *framework* basado en JUnit que cuenta con:

- Anotaciones extendidas.
- *Testing* continuo automático, al detectar cambios en el proyecto, el *framework* realiza todos los tests de manera automática.
- Interfaz gráfica ampliada al propio Eclipse para mejorar la información de la cobertura de tests sobre el código.
- Integración automática con el código de pruebas legado.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
Sureassert UC	No	Si	No	No	Si	Si	Si	No	4

Tabla 36 Sureassert UC

4.1.36. TestNG

TestNG [48] es un *framework* de *testing* basado en JUnit y NUnit que añade nuevas funcionalidades para hacerlo más potente y fácil de usar.

También dispone de un *plug-in* para Eclipse de manera que hace más fácil la ejecución de los tests, su visualización de los resultados, así como la capacidad de transformar tests de JUnit a TestNG.

Cuenta con funcionalidades añadidas como probar conexión a bases de datos, inyección de dependencias e integración de Selenium.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
TestNG	Si	Si	Si	No	Si	Si	Si	No	6

Tabla 37 TestNG

4.1.37. UISpec4J

UISpec4J [49] es una biblioteca para *testing* funcional y unitario para aplicaciones basadas en Java y Swing. Este *framework* está creado sobre JUnit y TestNG.

UISpec4J permite interactuar y analizar la capa de presentación de swing, con sus componentes y eventos para realizar pruebas funcionales.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
UISpec4J	Si	Si	No	No	No	No	Si	Si	4

Tabla 38 UISpec4J

4.1.38. Unitils

Unitils [50] es un *framework* de *testing* destinado a la realización de pruebas unitarias y de integración.

Cuenta con módulos muy variados y la integración de otros *frameworks* que completan su funcionamiento. Éstos son por ejemplo DbUnit, EasyMock o un módulo de Spring.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
Unitils	Si	Si	Si	No	Si	Si	Si	Si	7

Tabla 39 Unitils

4.1.39. XMLUnit

XMLUnit [51] es un *framework* de *testing* que provee a JUnit de funcionalidades para poder probar estructuras de datos en XML. Estas funcionalidades son:

- Marcar las diferencias entre dos XML.
- Comprobar el resultado de un XML usando hojas de transformación.
- Evaluar mediante XPath.
- Validación.

Este *framework* puede probar contenido HTML como si fuera XML para probar los contenidos de las páginas web.

Evaluación:

Las características con las que cuenta este *framework* basándose en los criterios descritos anteriormente y la nota otorgada bajo éstos son:

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
XMLUnit	No	Si	No	No	No	No	Si	No	2

Tabla 40 XMLUnit

4.2. Evaluación exhaustiva

4.2.1. Criterios seguidos

Tras la primera evaluación de todos los *frameworks*, se han establecido una serie de criterios de evaluación dentro de cuatro dimensiones diferentes para adecuarse a los requisitos de la arquitectura J-everis.

Las cuatro dimensiones son: Ejecución, Operación, Desarrollo y Soporte.

4.2.1.1. Criterios de Ejecución

La dimensión **Ejecución** se basa en criterios para poder ejecutar diferentes entornos de programación, adecuarse a otros *frameworks* y otras características referentes al uso de APIs y adaptaciones de éstas para la integración en dicho *framework*.

Dentro de esta dimensión, podemos diferenciar 7 criterios de evaluación que se han considerado:

- **Integración con Spring:** Un concepto fundamental de los *frameworks* que se necesitan integrar con J-everis es Spring, éste *framework* es la base de las funcionalidades que otorgan la potencia del *framework* corporativo. Por ello es necesario que un buen *framework* pudiera disponer de esta característica.
- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** Tanto JUnit como TestNG son los dos *frameworks* de partida con los que suelen ampliar las funcionalidades los *frameworks* de *testing*, aunque como ya se verá más adelante TestNG es más que un *framework* unitario. Las capacidades que pueda tener un *framework* de *testing* hacen que sea necesaria la utilización de uno de éstos porque son la base de todas las pruebas actuales sobre aplicaciones Java.
- **Inyección de dependencias (CDI):** Esta cualidad es una de las que más facilitan los desarrollos, debido a la sencillez que aporta a la hora de inyectar las entidades que haya en los proyectos.
- **Capa de presentación (Integración de web drivers o *frameworks*):** La capa de presentación es una de las capas que menos se suelen tener en cuenta a la hora de probar. Es importante porque es necesario en las pruebas funcionales y de aceptación.
- **Contenedores de aplicaciones:** Las aplicaciones empresariales suelen tener contenedores donde desplegarse para que puedan funcionar y en ocasiones no se pueden probar todos los tests dentro porque el *framework* no tiene capacidad de integración con dicho contenedor, es ahí donde entran las cualidades que tenga cada

framework para adaptarse a los contenedores de aplicaciones o incluso no necesitarlos

- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...):** El acceso a los datos es algo que prácticamente toda aplicación tiene, en este caso hay muy pocos *frameworks* que tengan una funcionalidad importante a la hora de poder probar los componentes que se hayan desarrollado para el acceso a datos. Por lo tanto, es importante que un *framework* pudiera disponer de características en este ámbito.
- **Más utilidades:** En este este apartado entran cualquier funcionalidad añadida que no esté contemplada en los demás, puede haber diversas funcionalidades que le den un toque muy especial al *framework*, por eso es será una cualidad con una importancia necesaria en las evaluaciones.

4.2.1.2. Criterios de Desarrollo

La dimensión **Desarrollo** se basa en criterios para facilitar la vida al desarrollador a la hora de integrar herramientas y los módulos mediante *plug-ins* y gestión de las dependencias. Los dos criterios que forman la dimensión son:

- **Herramientas para ayuda al desarrollo:** son las funcionalidades que están especializadas en cada *framework* para hacer el desarrollo más preciso, sencillo y fácil. Esto se puede reflejar en herramientas externas a un IDE en concreto o en forma de *plug-ins* para ellos.
- **Herramientas de gestión y configuración de dependencias (Maven):** La gestión y configuración de dependencias es necesaria para un control centralizado de las dependencias de un proyecto ya que en ocasiones puede volverse muy difícil, por lo tanto, herramientas como Maven o Ant para la gestión de dependencias y control de proyecto es muy importante.

4.2.1.3. Criterios de Operación

La dimensión **Operación** se basa en criterios que den utilidad para la visualización de los resultados que aporten los *frameworks* a estudiar. En este caso solo se dispone de un solo criterio que es la **Monitorización de los test (IDE o WEB).**

4.2.1.1. Criterios de Soporte

La dimensión **Soporte** se basa en diferentes criterios como el control de la calidad de la documentación o las garantías de que el *software* que se puede utilizar tenga continuidad en el tiempo. Concretamente hay 4 criterios que son:

- **Facilidad de uso/aprendizaje:** Característica necesaria para que un *framework* pueda ser utilizado debido a que su uso puede agilizar o entorpecer el desarrollo de un proyecto.
- **Calidad de la documentación:** La calidad de la documentación es muy valiosa cuando se está trabajando con sistemas muy complejos y con muchísimas funcionalidades, ahí intervienen los JavaDocs generados por las APIs que hacen que leer determinadas clases o interfaces sea más sencillo y facilite el trabajo.
- **Entidades detrás del proyecto:** El hecho de disponer de entidades, grupo de personas o comunidad que den soporte al proyecto del *framework* hace que éste tenga

una garantía de que hay un desarrollo perfeccionado bajo las buenas prácticas de una gran empresa o asociación, la garantía de robustez que puede dar una buena comunidad de gente dispuesta a mejorar el proyecto o una gran ayuda en foros.

- **Evolución (evolución histórica y actividad actual):** Esto es importante para que no se esté trabajando sobre un *framework* que, aunque esté disponible y se use, no disponga de una continuación a lo largo del tiempo y esté obsoleto o incompleto.

4.2.2. Ponderación de los criterios

Coeficiente de peso del criterio

Cada criterio tiene una ponderación para luego pesar más o menos a la hora de establecer las notas a cada *framework*. Los coeficientes de los pesos en las ponderaciones son:

- Peso 1. Criterio suficiente.
- Peso 2. Criterio importante.
- Peso 3. Criterio prácticamente necesario.

Grado de cobertura

Por cada *framework* y en cada apartado anteriormente descrito, se le dará un porcentaje de cobertura por el criterio seleccionado. Se han creado 5 grados de cobertura que son:

Ninguna (N): 0% cobertura.
Baja (B): 25% cobertura.
Media (M): 50% cobertura.
Alta (A): 75% cobertura.
Total (T): 100% cobertura.

Cada uno de estos criterios tiene una ponderación sobre la puntuación total que se establece en función a la multiplicación del coeficiente de peso del criterio por el grado de cobertura que éste tenga.

Ejemplo:

		Ejemplo	
Característica	**Coeficiente de peso**	**Puntuaciones**	
Integración con Spring	**3**	0%	0,0
Integración con *frameworks* de tests unitarios (Junit, TestNG)	**3**	25%	0,8
Inyección de dependencias (CDI)	**1**	0%	0,0
Capa de presentación (Integración de web drivers o *frameworks*)	**2**	75%	1,5
Contenedores de aplicaciones	**3**	100%	3,0
Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...)	**3**	0%	0,0
Más utilidades	**1**	25%	0,3
	16		**34%**

Tabla 41 ejemplo de puntuaciones de los criterios.

Como se puede observar en la **¡Error! No se encuentra el origen de la referencia.**, se han dispuesto unos coeficientes de peso a cada característica o criterio a evaluar y después se añaden las puntuaciones (en formato porcentual y en nota) de cada característica sobre el *framework* evaluado.

Finalmente se suman todas las puntuaciones y se dividen entre el total de los coeficientes de peso para generar una media de cobertura en dicha dimensión.

4.2.3. Selección de los *frameworks*

A continuación se describen los *frameworks* seleccionados del punto Listado, descripción y primera evaluación de los *frameworks* que han tenido una puntuación de 4 puntos o más en los primeros criterios de evaluación. La lista de todos los *frameworks* ordenados por nota está en el

Anexo I: Tabla comparativa global primeros criterios de selección.

4.2.4. Evaluación de los *frameworks* seleccionados

Para cada *framework* mostrado en este punto, se ha descrito cada característica que posee en función a los criterios anteriormente descritos y también se ha añadido la tabla con las puntuaciones asignadas a cada *framework*.

4.2.4.1. Arquillian

Ejecución

- **Integración con Spring:** Se integra con Spring mediante un módulo que provee de todas las características que el core de Spring tiene.
- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** Posee completa integración tanto con JUnit como con TestNG y adapta sus anotaciones y descripciones a él. También tiene en forma de módulo otro TestRunner llamado Spock.
- **Inyección de dependencias (CDI):** Genera directamente inyección de dependencias mediante el uso de ShrinkWrap en las clases de test, creando dinámicamente un jar con todos los componentes que se necesita en el caso de prueba.
- **Capa de presentación (Integración de web drivers o *frameworks*):** Cuenta con Drone, una potente extensión que permite elegir el web driver con el que realizar las pruebas. Drone por defecto cuenta con Arquillian Graphene 2, lo cual es una implementación para Java de Selenium web driver, que añade funcionalidades como Ajax y JQuery a la potencia de los test. También dispone de módulos como AngularJs, una extensión de Graphene para AngularJs; y entre muchos otros más también destaca Arquillian Warp, una extensión para realizar pruebas en cliente para *frameworks* como Servlet API, JSF 2, Spring MVC y REST.
- **Contenedores de aplicaciones:** Tiene adaptadores para los contenedores de aplicaciones que van desde IBM WebSphere hasta Tomcat, pasando por JBoss y muchos más. Oficialmente cuenta con 30 adaptadores para su uso en aplicaciones
- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...):** Provee *testing* sobre la capa de persistencia mediante diferentes implementaciones de JPA y también puede ejecutar accesos con Hibernate.
- **Más utilidades:** Posee un vasto número de extensiones como Droidium, es un contenedor de Android para probar aplicaciones de forma nativa; Recorder, para la

generación de informes; Jacobo, una utilidad para control de la cantidad de código cubierto por tests; Spacelift, para gestionar las tareas programadas en los tests.

Desarrollo

- **Herramientas para ayuda al desarrollo:** Cuenta con un *plug-in* básico para Eclipse que facilita la integración de Arquillian a proyectos así como un wizard para la generación de clases de prueba.
- **Herramientas de gestión y configuración de dependencias (Maven):** Todo Arquillian está gestionado mediante el control de dependencias de Maven.

Operación

- **Monitorización de los test (IDE o WEB):** Arquillian cuenta con la monitorización básica que provee JUnit a los entornos de desarrollo.

Soporte

- **Facilidad de uso/aprendizaje:** La facilidad de uso viene determinada por el número de módulos o extensiones que tiene este *framework*, así Arquillian tiene una dificultad de uso bastante avanzada debido al alto número de configuraciones y gestión de dependencias que se deben de hacer para cubrir un proyecto completo con él. La curva de aprendizaje es alta porque no es un sistema sencillo pero sin embargo la comunidad de Arquillian tiene a disposición de todo el mundo muchos ejemplos de implementaciones y configuraciones. Aun así todos los ejemplos no son suficientes para todo el conjunto de configuraciones que hay que tocar.
- **Calidad de la documentación:** Existe una amplia documentación sobre Arquillian y sobre cada módulo que es actualizado cada poco tiempo completándose con las correcciones a lo largo del tiempo del *framework*. Todas las incidencias son gestionadas con Jira y tienen soporte en foros y wiki.
- **Entidades detrás del proyecto:** En este caso Arquillian pertenece al proyecto JBoss que a su vez es parte de la compañía Red Hat, una gran compañía de *software*.
- **Evolución (evolución histórica y actividad actual):** JBoss lleva con el proyecto Arquillian desde el 2010 y durante todo este tiempo sigue desarrollándose con nuevas funcionalidades y módulos.

En función a todas las descripciones que se han hecho de cada apartado, se ha generado la tabla con las puntuaciones siguientes:

Dimensión	**Característica**	**Cobertura**	
Ejecución	**Integración con Spring**	T	**100%**
	Integración con *frameworks* de tests unitarios (Junit, TestNG)	T	**100%**
	Inyección de dependencias (CDI)	T	**100%**
	Capa de presentación (Integración de web drivers o *frameworks*)	T	**100%**
	Contenedores de aplicaciones	T	**100%**
	Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...)	M	**50%**
	Más utilidades	T	**100%**

Desarrollo	**Herramientas para ayuda al desarrollo**	M	**50%**
	Herramientas de gestión y configuración de dependencias (Maven)	T	**100%**
Operación	**Monitorización de los test (IDE o WEB)**	B	**25%**
Soporte	**Facilidad de uso/aprendizaje**	M	**50%**
	Calidad de la documentación	T	**100%**
	Entidades detrás del proyecto	T	**100%**
	Evolución (evolución histórica y actividad actual)	T	**100%**

Tabla 42 Puntuaciones Arquillian

4.2.4.2. Cucumber JVM

Ejecución

- **Integración con Spring:** Dispone de un Jar que es un módulo que provee de funcionalidades como la inyección de dependencias. También dispone del módulo para usar Arquillian y todas sus funcionalidades.
- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** Cucumber JVM tiene varios módulos para utilizar diferentes runners. En concreto dispone de módulo para JUnit y también para TestNG.
- **Inyección de dependencias (CDI):** Tiene módulos para realizar la inyección de dependencias. Dispone de diferentes contenedores como módulos para realizarlo que son: PicoContainer, Guice, OpenEJB, Spring, Weld y Needle.
- Capa de presentación (Integración de web drivers o *frameworks*): Tiene todas las implementaciones que pudiera tener Arquillian ya que, mediante un módulo se puede integrar con todas las funcionalidades de Arquillian.
- **Contenedores de aplicaciones:** De manera propia dispone de contenedores como PicoContainer, Guice y Weld. También dispone de los módulos para usar Needle y Arquillian, junto con todos los módulos que ellos tienen.
- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis…):** Dispone de todas las funcionalidades que tiene Arquillian.
- **Más utilidades:** Dispone de todas las funcionalidades que tiene Arquillian.

Desarrollo

- **Herramientas para ayuda al desarrollo:** No cuenta con ninguna extensión para IDE.
- **Herramientas de gestión y configuración de dependencias (Maven):** Todos los módulos están disponibles mediante el gestor de dependencias Maven.

Operación

- **Monitorización de los test (IDE o WEB):** No cuenta con ninguna monitorización propia para IDEs salvo la del propio JUnit.

Soporte

- **Facilidad de uso/aprendizaje:** La curva de aprendizaje del uso de herramientas de BDD suele ser bajo ya que son *frameworks* orientados a facilitar el desarrollo de las pruebas. La dificultad está en la complejidad de las pruebas que se creen y en la profundidad de estas usando otros módulos.
- **Calidad de la documentación:** La documentación es amplia y los ejemplos de las

funcionalidades del *framework* son muy extensos pero aún le queda por desarrollar a este *framework* en lo que se refiere a manuales y uso.

- **Entidades detrás del proyecto:** Cucumber es una implementación de Cukes que es un proyecto propio de una persona llamada Aslak Hellessøy, aunque cuenta con unos 200 colaboradores, no tiene el poder
- **Evolución (evolución histórica y actividad actual):** Cucumber lleva en desarrollo desde 2007 y en 2012 se sacó a la luz la primera release estable (v1.0.0). Actualmente sigue teniendo mejoras en la integración con otros *frameworks* para aumentar las funcionalidades de este teniendo una comunidad muy activa en lo que a desarrollo se refiere.

En función a todas las descripciones que se han hecho de cada apartado, se ha generado la tabla con las puntuaciones siguientes:

Dimensión	Característica	Cobertura	
Ejecución	Integración con Spring	T	100%
	Integración con *frameworks* de tests unitarios (Junit, TestNG)	T	100%
	Inyección de dependencias (CDI)	T	100%
	Capa de presentación (Integración de web drivers o *frameworks*)	T	100%
	Contenedores de aplicaciones	T	100%
	Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...)	M	50%
	Más utilidades	T	100%
Desarrollo	Herramientas para ayuda al desarrollo	B	25%
	Herramientas de gestión y configuración de dependencias (Maven)	T	100%
Operación	Monitorización de los test (IDE o WEB)	B	25%
Soporte	Facilidad de uso/aprendizaje	M	50%
	Calidad de la documentación	A	75%
	Entidades detrás del proyecto	A	75%
	Evolución (evolución histórica y actividad actual)	T	100%

Tabla 43 Puntuaciones Cucumber JVM

4.2.4.3. Unitils

Ejecución

- **Integración con Spring:** Tiene un módulo para dar soporte a Spring, pudiendo inyectar Beans, gestionar el *ApplicationContext*, usar el *SessionFactory* de Hibernate e integrar algunas funcionalidades más de Unitils con Spring.
- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** Soporta tanto TestNG como JUnit.
- **Inyección de dependencias (CDI):** Dispone de un módulo para inyección de dependencias de clases e instancias.
- **Capa de presentación** (Integración de web drivers o *frameworks*): Tiene un módulo que integra Selenium con Unitils.
- **Contenedores de aplicaciones**: Posibilidad de integrar Unitils en cualquier

contenedor.

- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis…):** Unitils integra las funcionalidades de DbUnit y por lo tanto puede emular todo lo que éste hace. Como revertir estados de bases de datos o conectarse a diferentes tipos de gestores.
- **Más utilidades:** Una de las utilidades más destacadas es que contiene un módulo de utilidades para Reflection en Java y también tiene a EasyMock como *framework* de mocking.

Desarrollo

- **Herramientas para ayuda al desarrollo:** No dispone de ningún soporte para IDEs de los que ya dispone JUnit.
- **Herramientas de gestión y configuración de dependencias (Maven):** Unitils está gestionado mediante el control de dependencias de Maven y por SVN también.

Operación

- **Monitorización de los test (IDE o WEB):** No dispone de ningún soporte para IDEs de los que ya dispone JUnit.

Soporte

- **Facilidad de uso/aprendizaje:** Este es un *framework* bastante complejo que integra mucha funcionalidad pero ésta al ser modular no se necesita disponer de todo en todo momento así que finalmente es fácil de usar por la buena documentación.
- **Calidad de la documentación:** En lo que a documentación respecta, dispone de una vasta documentación y muy detallada de cada componente y manuales de uso.
- **Entidades detrás del proyecto:** Este proyecto está formado por 6 miembros que desarrollan solos y sin ayuda de la comunidad.
- **Evolución (evolución histórica y actividad actual):** El proyecto empezó en 2006 y ha ido desarrollándose hasta que en 2011 dejó de tener mejoras para solo tener mantenimiento. Aunque actualmente parece que se está preparando una nueva versión.

En función a todas las descripciones que se han hecho de cada apartado, se ha generado la tabla con las puntuaciones siguientes:

Dimensión	**Característica**	**Cobertura**	
Ejecución	**Integración con Spring**	T	**100%**
	Integración con *frameworks* de tests unitarios (Junit, TestNG)	T	**100%**
	Inyección de dependencias (CDI)	T	**100%**
	Capa de presentación (Integración de web drivers o *frameworks*)	A	**75%**
	Contenedores de aplicaciones	T	**100%**
	Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...)	T	**100%**
	Más utilidades	M	**50%**
Desarrollo	**Herramientas para ayuda al desarrollo**	B	**25%**
	Herramientas de gestión y configuración de dependencias (Maven)	T	**100%**
Operación	**Monitorización de los test (IDE o WEB)**	B	**25%**

Soporte	**Facilidad de uso/aprendizaje**	T	**100%**
	Calidad de la documentación	T	**100%**
	Entidades detrás del proyecto	B	**25%**
	Evolución (evolución histórica y actividad actual)	A	**75%**

Tabla 44 Puntuaciones Unitils

4.2.4.4. Cactus

Ejecución

- **Integración con Spring:** No tiene módulos conocidos de Spring
- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** Posee la implementación de JUnit 3.x pero no la de JUnit 4.
- **Inyección de dependencias (CDI):** No tiene inyección de dependencias
- **Capa de presentación (Integración de web drivers o *frameworks*):** Cuenta con StrutsTestCase, JSFUnit e integra Jetty.
- **Contenedores de aplicaciones:** Tiene integrado Cargo, un gestor de contenedores de aplicaciones que hace que gestione todos los contenedores de la misma manera. Cactus funciona con Servlets, por lo tanto, cualquier contenedor de aplicaciones que pueda ejecutar servlets es bien recibido. http://cargo.codehaus.org/
- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...):** No dispone de una implementación para tests en la capa de persistencia.
- **Más utilidades:** Cactus cuenta con integración para Ant y extiende las funcionalidades de JUnitEE.

Desarrollo

- **Herramientas para ayuda al desarrollo:** Cuenta con un *plug-in* básico para Eclipse que facilita la ejecución de los tests.
- **Herramientas de gestión y configuración de dependencias (Maven):** Todo Cactus está gestionado mediante el control de dependencias de Maven.

Operación

- **Monitorización de los test (IDE o WEB):** Cactus cuenta con la monitorización básica que provee JUnit a los entornos de desarrollo.

Soporte

- **Facilidad de uso/aprendizaje:** Aunque el proyecto es grande, está bien estructurado para que cada una de las funcionalidades tenga sus propios ejemplos y puedan ser utilizadas desde cero fácilmente.
- **Calidad de la documentación:** Todo el documento está muy bien documentado aunque hay módulos que ya no están disponibles en la web ya que hace tiempo que dejó de ser usado.
- **Entidades detrás del proyecto:** En el caso de Cactus, el proyecto pertenecía a la Apache *Software* Foundation nombrado dentro del macro proyecto Jakarta.
- **Evolución (evolución histórica y actividad actual):** El proyecto comenzó en 2001, en 2008 se hizo la última modificación y en 2011 Apache lo dio por abandonado.

En función a todas las descripciones que se han hecho de cada apartado, se ha generado la tabla con las puntuaciones siguientes:

Dimensión	Característica	Cobertura	
Ejecución	Integración con Spring	N	0%
	Integración con *frameworks* de tests unitarios (Junit, TestNG)	B	25%
	Inyección de dependencias (CDI)	N	0%
	Capa de presentación (Integración de web drivers o *frameworks*)	A	75%
	Contenedores de aplicaciones	T	100%
	Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...)	N	0%
	Más utilidades	B	25%
Desarrollo	Herramientas para ayuda al desarrollo	M	50%
	Herramientas de gestión y configuración de dependencias (Maven)	T	100%
Operación	Monitorización de los test (IDE o WEB)	B	25%
Soporte	Facilidad de uso/aprendizaje	B	75%
	Calidad de la documentación	M	50%
	Entidades detrás del proyecto	T	100%
	Evolución (evolución histórica y actividad actual)	N	0%

Tabla 45 Puntuaciones Cactus

4.2.4.5. FitNesse

Ejecución

- **Integración con Spring:** Contiene un módulo que realiza la ejecución de transacciones e inyección del contexto como lo haría Spring.
- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** Tiene la posibilidad de ejecutar tests con JUnit y también puede ejecutar los tests sin él.
- **Inyección de dependencias (CDI):** Dispone de la inyección de dependencias que puede proveer el *plug-in* de Spring, no más como Weld o Needle.
- **Capa de presentación (Integración de web drivers o *frameworks*):** FitNesse se integra mediante *plug-ins* con Selenium, tanto con el web drivers como con el IDE.
- **Contenedores de aplicaciones:** Este *framework* funciona aisladamente de manera que tiene su propio servidor llamado FitServer. Utiliza las aplicaciones en remoto por lo tanto no necesitaría de ningún contenedor.
- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...):** Puede funcionar con cualquier implementación de JPA u otros sobre la capa de persistencia ya que al ser un servidor externo, solo se trata de publicar el proyecto a probar dentro de FitServer.
- **Más utilidades:** Tiene utilidades en forma de *plug-ins* para probar conexiones a LDAP, APIs Rest, sistemas distribuidos y acceso a datos.

Desarrollo

- **Herramientas para ayuda al desarrollo:** El formato de FitNesse es en forma de Wiki, por lo tanto es una manera más sencilla de realizar las pruebas y organizar las pruebas de una manera más estructurada y visiblemente atractiva.

- **Herramientas de gestión y configuración de dependencias (Maven):** Dispone de las dependencias del proyecto en formato Maven, Ivy y Gradle.

Operación

- **Monitorización de los test (IDE o WEB**): Los resultados de los tests ejecutados se ven a través de la Wiki de una forma similar a cómo se verían en un IDE como Eclipse.

Soporte

- **Facilidad de uso/aprendizaje:** Este *framework* está orientado a la facilidad de uso y para tener una curva de aprendizaje baja sobre todo para que pueda ser utilizado también por los clientes del negocio.
- **Calidad de la documentación:** La documentación del proyecto es muy amplia y bien descrita, Dispone de manuales y guías de uso de todas sus funcionalidades de las que dispone.
- **Entidades detrás del proyecto:** No tiene entidades detrás del proyecto pero lo que sí que tiene es a uno de los más famosos desarrolladores del mundo durante varias décadas llamado Robert C. Martin. También tiene 60 personas más que le apoyan de forma altruista en este proyecto a través de GitHub.
- **Evolución (evolución histórica y actividad actual):** Este proyecto lleva desde 2005 publicado en GitHub con actualizaciones del *software* muy a menudo.

En función a todas las descripciones que se han hecho de cada apartado, se ha generado la tabla con las puntuaciones siguientes:

Dimensión	Característica	Cobertura	
Ejecución	Integración con Spring	T	100%
	Integración con *frameworks* de tests unitarios (Junit, TestNG)	M	50%
	Inyección de dependencias (CDI)	M	50%
	Capa de presentación (Integración de web drivers o *frameworks*)	A	75%
	Contenedores de aplicaciones	N	0%
	Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...)	T	100%
	Más utilidades	A	75%
Desarrollo	Herramientas para ayuda al desarrollo	A	75%
	Herramientas de gestión y configuración de dependencias (Maven)	T	100%
Operación	Monitorización de los test (IDE o WEB)	A	75%
Soporte	Facilidad de uso/aprendizaje	T	100%
	Calidad de la documentación	T	100%
	Entidades detrás del proyecto	T	100%
	Evolución (evolución histórica y actividad actual)	T	100%

Tabla 46 Puntuaciones FitNesse

4.2.4.6. TestNG

Ejecución

- **Integración con Spring:** TestNG se puede agregar como un componente más de Spring.
- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** Es una extensión de JUnit que agrega funcionalidades a éste, pero se suele tener en cuenta como *framework* base diferente a JUnit.
- **Inyección de dependencias (CDI):** Dispone de inyección de dependencias de manera nativa.
- **Capa de presentación (Integración de web drivers o *frameworks*):** Posee una biblioteca concreta para crear tests con Selenium.
- **Contenedores de aplicaciones:** Funciona para cualquier contenedor que funcione ejecute JUnit.
- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...):** Se integra con los drivers que estén disponibles en la aplicación a probar pero solo se usa para inyectarlos en los servicios de los tests.
- **Más utilidades:** Puede ser utilizado con *frameworks* de mocking como PowerMock, Mockito; puede transformar tests de JUnit en tests de TestNG y también tiene soporte para muchos más lenguajes como Groovy, Python o Ruby.

Desarrollo

- **Herramientas para ayuda al desarrollo:** Tiene un *plug-ins* para los más conocidos IDEs como son Eclipse e InteliJ Idea que sirve para transformar los tests de JUnit en tests de este *framework* y también sirve como wizard de creación de tests desde cero.
- **Herramientas de gestión y configuración de dependencias (Maven):** Se pueden insertar todas las dependencias de los paquetes mediante Maven.

Operación

- **Monitorización de los test (IDE o WEB):** No cuenta con ninguna monitorización propia para IDEs salvo la del propio JUnit.

Soporte

- **Facilidad de uso/aprendizaje:** Es un *framework* muy completo y a la vez fácil de usar por su sencillez.
- **Calidad de la documentación:** Tiene una documentación muy amplia que se complementa perfectamente con todos los artículos y manuales publicados para el uso de este *framework* y todas sus características añadidas.
- **Entidades detrás del proyecto:** En un principio fue iniciado por un solo hombre, pero con el paso del tiempo ha llegado a hacerse una comunidad de más de 45 personas alrededor de este *framework*.
- **Evolución (evolución histórica y actividad actual):** Fue creado en 2004 y ha ido evolucionando a lo largo de los años hasta ahora. Actualmente se encuentra en la versión 6.8. y sigue en desarrollo.

En función a todas las descripciones que se han hecho de cada apartado, se ha generado la tabla con las puntuaciones siguientes:

Dimensión	Característica	Cobertura	
Ejecución	Integración con Spring	T	100%

	Integración con *frameworks* de tests unitarios (Junit, TestNG)	T	**100%**
	Inyección de dependencias (CDI)	T	**100%**
	Capa de presentación (Integración de web drivers o *frameworks*)	A	**75%**
	Contenedores de aplicaciones	T	**100%**
	Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...)	M	**50%**
	Más utilidades	A	**75%**
Desarrollo	**Herramientas para ayuda al desarrollo**	T	**100%**
	Herramientas de gestión y configuración de dependencias (Maven)	T	**100%**
Operación	**Monitorización de los test (IDE o WEB)**	B	**25%**
Soporte	**Facilidad de uso/aprendizaje**	T	**100%**
	Calidad de la documentación	T	**100%**
	Entidades detrás del proyecto	T	**100%**
	Evolución (evolución histórica y actividad actual)	T	**100%**

Tabla 47 Puntuaciones TestNG

4.2.4.7. Jnario

Ejecución

- **Integración con Spring:** No tiene módulos conocidos de Spring.
- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** Solo tiene soporte para JUnit.
- **Inyección de dependencias (CDI):** No tiene módulos conocidos para CDI.
- **Capa de presentación (Integración de web drivers o *frameworks*):** No necesita tener ningún módulo de integración pero aun así tiene ejemplos del uso en este *framework*.
- **Contenedores de aplicaciones:** No se integra con ningún contenedor de aplicaciones. Funciona como una aplicación Java estándar.
- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...):** No se integra con ningún *framework* de persistencia.
- **Más utilidades:** Se programa mediante un metalenguaje de java llamado Xtend que aporta mucha sencillez al lenguaje.

Desarrollo

- **Herramientas para ayuda al desarrollo:** Dispone de Xtend como metalenguaje de programación de los tests.
- **Herramientas de gestión y configuración de dependencias (Maven):** Tiene integración mediante bibliotecas de Maven.

Operación

- **Monitorización de los test (IDE o WEB):** No cuenta con ninguna monitorización propia para IDEs salvo la del propio JUnit.

Soporte

- **Facilidad de uso/aprendizaje:** Este *framework* tiene una sencillez de uso muy alta

y donde apenas hay curva de aprendizaje habiendo visto varios ejemplos de uso.

- **Calidad de la documentación:** La documentación del proyecto es correcta, está bien organizada y descrita y contiene numerosos ejemplos de cada
- **Entidades detrás del proyecto:** Ha sido desarrollado en la cuna de la compañía BMW Car IT, la rama de BMW encargada del *software*.
- **Evolución (evolución histórica y actividad actual):** Este proyecto lleva desde 2012 desarrollándose y en julio de 2014 ha salido la versión 1.0 del *software*.

En función a todas las descripciones que se han hecho de cada apartado, se ha generado la tabla con las puntuaciones siguientes:

Dimensión	Característica	Cobertura	
Ejecución	Integración con Spring	N	0%
	Integración con *frameworks* de tests unitarios (Junit, TestNG)	M	50%
	Inyección de dependencias (CDI)	N	0%
	Capa de presentación (Integración de web drivers o *frameworks*)	B	25%
	Contenedores de aplicaciones	N	0%
	Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...)	N	0%
	Más utilidades	N	0%
Desarrollo	Herramientas para ayuda al desarrollo	A	75%
	Herramientas de gestión y configuración de dependencias (Maven)	T	100%
Operación	Monitorización de los test (IDE o WEB)	N	0%
Soporte	Facilidad de uso/aprendizaje	T	100%
	Calidad de la documentación	T	100%
	Entidades detrás del proyecto	A	75%
	Evolución (evolución histórica y actividad actual)	T	100%

Tabla 48 Puntuaciones Jnario

4.2.4.8. Needle

Ejecución

- **Integración con Spring:** No tiene módulos conocidos de Spring.
- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** Tiene soporte tanto para JUnit como para TestNG.
- **Inyección de dependencias (CDI):** Dispone de clases para realizar Inyección de dependencias mediante anotaciones.
- **Capa de presentación (Integración de web drivers o *frameworks*):** Dispone de los complementos de Selenium Web Driver si se utiliza con TestNG.
- **Contenedores de aplicaciones:** No necesita de ningún contenedor de aplicaciones para probar las aplicaciones ya que funciona de manera aislada.
- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...):** Tiene la posibilidad de probar la capa de persistencia mediante proveedores de implementaciones de JPA, como Hibernate.
- **Más utilidades:** Posee utilidades como un conjunto de clases para mejorar la

Reflexión de Java, posibilidad de agregar EasyMock y Mockito fácilmente o ejecutar operaciones de BBDD.

Desarrollo

- **Herramientas para ayuda al desarrollo:** No cuenta con ninguna herramienta de ayuda al desarrollo.
- **Herramientas de gestión y configuración de dependencias (Maven):** Tiene integración mediante bibliotecas de Maven.

Operación

- **Monitorización de los test (IDE o WEB):** No cuenta con ninguna monitorización propia para IDEs salvo la del propio JUnit.

Soporte

- **Facilidad de uso/aprendizaje:** Needle es muy sencillo de utilizar y además facilita las cosas gracias a anotaciones en Java propia y a una API muy bien documentada.
- **Calidad de la documentación:** La API está perfectamente documentada en formato JavaDoc, así como los numerosos ejemplos tanto de uso como de instalación y configuración que hay en la página de la documentación.
- **Entidades detrás del proyecto:** Ha sido desarrollado por una empresa llamada Akquinet pero en el que solo trabajan 4 personas en éste.
- **Evolución (evolución histórica y actividad actual):** El proyecto lleva en marcha desde 2011 y ha tenido muchas revisiones y cambios desde el principio, pero ahora mismo lleva medio año parado sin noticias de la continuación de este *framework*.

En función a todas las descripciones que se han hecho de cada apartado, se ha generado la tabla con las puntuaciones siguientes:

Dimensión	**Característica**	**Cobertura**	
Ejecución	**Integración con Spring**	N	**0%**
	Integración con *frameworks* de tests unitarios (Junit, TestNG)	T	**100%**
	Inyección de dependencias (CDI)	A	**75%**
	Capa de presentación (Integración de web drivers o *frameworks*)	M	**50%**
	Contenedores de aplicaciones	N	**0%**
	Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...)	T	**100%**
	Más utilidades	A	**75%**
Desarrollo	**Herramientas para ayuda al desarrollo**	N	**0%**
	Herramientas de gestión y configuración de dependencias (Maven)	T	**100%**
Operación	**Monitorización de los test (IDE o WEB)**	B	**25%**
Soporte	**Facilidad de uso/aprendizaje**	T	**100%**
	Calidad de la documentación	T	**100%**
	Entidades detrás del proyecto	M	**50%**
	Evolución (evolución histórica y actividad actual)	M	**50%**

Tabla 49 Puntuaciones Needle

4.2.4.9. DbUnit

Ejecución

- **Integración con Spring:** Existe una implementación de DbUnit configurada para el uso con Spring.
- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** Solamente sirve con JUnit ya que es una extensión de este *framework*.
- **Inyección de dependencias (CDI):** No tiene inyección de dependencias. Salvo las que provee Spring.
- **Capa de presentación (Integración de web drivers o *frameworks*):** No está orientado a este ámbito y por lo tanto no dispone de esta funcionalidad.
- **Contenedores de aplicaciones:** No necesita de ningún contenedor de aplicaciones para probar las aplicaciones ya que funciona de manera aislada mediante sus conectores de bases de datos.
- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...):** Al tratarse de un *framework* orientado a las pruebas contra bases de datos, tiene una alta disponibilidad de conectores.
- **Más utilidades:** Posee funcionalidades orientadas a la transaccionalidad de las operaciones y también poder dejar el estado de la base intacto después de las pruebas.

Desarrollo

- **Herramientas para ayuda al desarrollo:** No cuenta con ninguna herramienta de ayuda al desarrollo.
- **Herramientas de gestión y configuración de dependencias (Maven):** Tiene integración mediante bibliotecas de Maven.

Operación

- **Monitorización de los test (IDE o WEB):** No cuenta con ninguna monitorización propia para IDEs salvo la del propio JUnit.

Soporte

- **Facilidad de uso/aprendizaje:** El *framework* no es demasiado fácil ya que requiere demasiada configuración para hacerlo funcionar y además no es que sea sencillo de usar.
- **Calidad de la documentación:** La API está perfectamente documentada en formato JavaDoc, así como los numerosos ejemplos de uso y un amplio FAQ.
- **Entidades detrás del proyecto:** Ha sido desarrollado por un grupo de programadores Java, en total los desarrolladores del proyecto son 11 junto con 7 personas que han contribuido al proyecto de una manera u otra.
- **Evolución (evolución histórica y actividad actual):** El proyecto lleva funcionando desde 2002 que salió la primera versión estable hasta ahora. Ha estado unos años parado pero parece ser que ha vuelto a arrancar porque en Mayo del 2014 sacaron una nueva versión a la luz.

En función a todas las descripciones que se han hecho de cada apartado, se ha generado la tabla con las puntuaciones siguientes:

Dimensión	Característica	Cobertura	
Ejecución	Integración con Spring	T	100%
	Integración con *frameworks* de tests unitarios (Junit, TestNG)	M	50%
	Inyección de dependencias (CDI)	B	25%
	Capa de presentación (Integración de web drivers o *frameworks*)	N	0%
	Contenedores de aplicaciones	N	0%
	Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...)	T	100%
	Más utilidades	M	50%
Desarrollo	Herramientas para ayuda al desarrollo	N	0%
	Herramientas de gestión y configuración de dependencias (Maven)	T	100%
Operación	Monitorización de los test (IDE o WEB)	B	25%
Soporte	Facilidad de uso/aprendizaje	M	50%
	Calidad de la documentación	T	100%
	Entidades detrás del proyecto	M	50%
	Evolución (evolución histórica y actividad actual)	A	75%

Tabla 50 Puntuaciones DbUnit

4.2.4.10. JMeter

Ejecución

- **Integración con Spring:** Puede probar cualquier aplicación pero de manera funcional solamente.
- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** No depende de ninguna de los dos *frameworks* de *testing* básicos.
- **Inyección de dependencias (CDI):** No dispone de inyección de dependencias.
- **Capa de presentación (Integración de web drivers o *frameworks*):** Se pueden realizar pruebas de carga y funcionales contra páginas web mediante el uso de web drivers para Chrome, Firefox. Dispone también de un plug-in para realizar tests con Selenium.
- **Contenedores de aplicaciones**: No aplica esta funcionalidad en esta herramienta.
- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...):** Tiene la habilidad de lanzar consultas a todas las bases de datos que utilicen JDBC como conector de Java.
- **Más utilidades:** Posee soporte para todo tipos de protocolos web, desde LDAP hasta FTP, HTTP, SMTP…Ejecuta también pruebas de carga distribuida y concurrente.

Desarrollo

- **Herramientas para ayuda al desarrollo:** La herramienta consiste en una interfaz gráfica externa a cualquier IDE donde se pueden configuran los tests a realizar.
- **Herramientas de gestión y configuración de dependencias (Maven):** No depende de ningún gestor de dependencias. Es una herramienta que se descarga y ejecuta aisladamente.

Operación

- **Monitorización de los test (IDE o WEB):** Los resultados de los tests se muestran a través de la interfaz gráfica que habilita este programa.

Soporte

- **Facilidad de uso/aprendizaje:** El uso de esta herramienta es sencillo ya que todos los tests se configuran a través de wizards que orientan lo que hacer al probador.
- **Calidad de la documentación:** JMeter dispone de un manual y una wiki muy amplios para consultar su funcionamiento mediante ejemplos y tutoriales. También aporta una documentación completísima de la API para programar clientes que consuman o extiendan sus funcionalidades.
- **Entidades detrás del proyecto:** La entidad que apoya a este proyecto es la Fundación de *Software* Apache. Es llevado por un grupo pequeño de personas pero tiene muchos contribuyentes que aportan mejoras y reportan fallos para que se siga desarrollando.
- **Evolución (evolución histórica y actividad actual):** El proyecto lleva en marcha desde 2001 en una versión estable. Sigue en desarrollo ya que este año 2014 se ha sacado una nueva release y sigue habiendo commits en su proyecto de GitHub.

En función a todas las descripciones que se han hecho de cada apartado, se ha generado la tabla con las puntuaciones siguientes:

Dimensión	**Característica**	**Cobertura**	
Ejecución	**Integración con Spring**	M	**50%**
	Integración con *frameworks* de tests unitarios (Junit, TestNG)	N	**0%**
	Inyección de dependencias (CDI)	N	**0%**
	Capa de presentación (Integración de web drivers o *frameworks*)	T	**100%**
	Contenedores de aplicaciones	N	**0%**
	Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...)	T	**100%**
	Más utilidades	M	**50%**
Desarrollo	**Herramientas para ayuda al desarrollo**	T	**100%**
	Herramientas de gestión y configuración de dependencias (Maven)	N	**0%**
Operación	**Monitorización de los test (IDE o WEB)**	A	**75%**
Soporte	**Facilidad de uso/aprendizaje**	T	**100%**
	Calidad de la documentación	T	**100%**
	Entidades detrás del proyecto	T	**100%**
	Evolución (evolución histórica y actividad actual)	T	**100%**

Tabla 51 Puntuaciones JMeter

4.2.4.11. Pax EXAM

Ejecución

- **Integración con Spring:** Uno de los modos de ejecución de Pax es para aplicaciones web, de esta forma permite la integración con Spring y su inyección de dependencias.

- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** Tiene soporte tanto para JUnit y como para TestNG.
- **Inyección de dependencias (CDI):** Otorga funcionalidad de CDI donde no requiere tener un entorno para Java EE configurado.
- **Capa de presentación (Integración de web drivers o *frameworks*):** Dispone de los complementos de Selenium Web Driver si se utiliza con TestNG.
- **Contenedores de aplicaciones:** Es capaz de ejecutar contenedores para Java EE, aplicaciones web y aplicaciones basadas en OSGi.
- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...):** Dentro del proyecto contiene componentes para JPA e Hibernate.
- **Más utilidades:** Lo más destacado es la ejecución de tests para aplicaciones con arquitectura OSGi. También tiene un contenedor propio para los tests si no se desea ejecutar dentro de los contenedores de los proyectos a probar.

Desarrollo

- **Herramientas para ayuda al desarrollo:** No tiene ninguna herramienta de ayuda al desarrollo para ningún IDE.
- **Herramientas de gestión y configuración de dependencias (Maven):** Se integra en las aplicaciones mediante gestión de dependencias de Maven.

Operación

- **Monitorización de los test (IDE o WEB):** No cuenta con ninguna monitorización propia para IDEs salvo la del propio JUnit.

Soporte

- **Facilidad de uso/aprendizaje:** El uso de Pax EXAM es complicado debido a las múltiples funcionalidades que tiene. Aunque cuenta con ejemplos del uso y configuración, no cubren todas las funcionalidades.
- **Calidad de la documentación:** Tiene una amplia documentación pero está demasiado desorganizada.
- **Entidades detrás del proyecto:** La organización detrás del proyecto se llama Open Participation *Software* for Java, es una comunidad de desarrolladores de utilidades para el entorno de Java (como bien indica su nombre) que está bajo el apoyo de Organizaciones como Atlassian, JetBrains o JRebel entre otros.
- **Evolución (evolución histórica y actividad actual):** Este *software* lleva desarrollándose desde antes del 2009, año en que salió la primera versión estable. Actualmente se llegan en la versión 4.1 del *software* publicando nuevas versiones muy a menudo.

En función a todas las descripciones que se han hecho de cada apartado, se ha generado la tabla con las puntuaciones siguientes:

Dimensión	**Característica**	**Cobertura**	
Ejecución	**Integración con Spring**	T	**100%**
	Integración con *frameworks* de tests unitarios (Junit, TestNG)	T	**100%**
	Inyección de dependencias (CDI)	T	**100%**

	Capa de presentación (Integración de web drivers o *frameworks*)	M	**50%**
	Contenedores de aplicaciones	T	**100%**
	Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...)	A	**75%**
	Más utilidades	M	**50%**
Desarrollo	**Herramientas para ayuda al desarrollo**	N	**0%**
	Herramientas de gestión y configuración de dependencias (Maven)	T	**100%**
Operación	**Monitorización de los test (IDE o WEB)**	B	**25%**
Soporte	**Facilidad de uso/aprendizaje**	M	**50%**
	Calidad de la documentación	M	**50%**
	Entidades detrás del proyecto	T	**100%**
	Evolución (evolución histórica y actividad actual)	T	**100%**

Tabla 52 Puntuaciones Pax EXAM

4.2.4.12. SpryTest

Ejecución

- **Integración con Spring:** No dispone de integración con Spring.
- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** Es una extensión de JUnit pero no permite TestNG.
- **Inyección de dependencias (CDI):** No dispone de Inyección de dependencias.
- **Capa de presentación (Integración de web drivers o *frameworks*):** No tiene integración con *frameworks* de la capa visual.
- **Contenedores de aplicaciones:** Funciona con cualquier contenedor que pueda ejecutar JUnit.
- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...):** No tiene la funcionalidad de generar tests para la capa de persistencia.
- **Más utilidades:** Generación automática de tests basados en JUnit y ejecución de tests de regresión automáticos.

Desarrollo

- **Herramientas para ayuda al desarrollo:** Este *framework* es un plug-in para Eclipse para la generación de tests automáticos sobre proyectos Java.
- **Herramientas de gestión y configuración de dependencias (Maven):** No se integra mediante Maven ya que se instala como plug-in en Eclipse.

Operación

- **Monitorización de los test (IDE o WEB):** No cuenta con ninguna monitorización propia para IDEs salvo la del propio JUnit.

Soporte

- **Facilidad de uso/aprendizaje:** SpryTest es muy sencillo de utilizar porque cuenta con wizards de configuración para la generación de los tests.
- **Calidad de la documentación:** Cuenta con muy poca documentación publicada.
- **Entidades detrás del proyecto:** Fue creado por una empresa de *software* llamada Sprystone *Software*. Esta empresa no es conocida y además no aparece como empresas

activas.

- **Evolución (evolución histórica y actividad actual):** El proyecto empezó en 2009 y tuvo mantenimiento hasta 2011.

En función a todas las descripciones que se han hecho de cada apartado, se ha generado la tabla con las puntuaciones siguientes:

Dimensión	Característica	Cobertura	
Ejecución	Integración con Spring	N	0%
	Integración con *frameworks* de tests unitarios (Junit, TestNG)	M	50%
	Inyección de dependencias (CDI)	N	0%
	Capa de presentación (Integración de web drivers o *frameworks*)	N	0%
	Contenedores de aplicaciones	T	100%
	Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...)	N	0%
	Más utilidades	A	75%
Desarrollo	Herramientas para ayuda al desarrollo	T	100%
	Herramientas de gestión y configuración de dependencias (Maven)	N	0%
Operación	Monitorización de los test (IDE o WEB)	B	25%
Soporte	Facilidad de uso/aprendizaje	T	100%
	Calidad de la documentación	B	25%
	Entidades detrás del proyecto	N	0%
	Evolución (evolución histórica y actividad actual)	N	0%

Tabla 53 Puntuaciones SpryTest

4.2.4.13. Sureassert UC

Ejecución

- **Integración con Spring:** No dispone de funcionalidades para Spring.
- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** El *framework* se puede integrar con JUnit.
- **Inyección de dependencias (CDI):** No permite la Inyección de dependencias.
- **Capa de presentación (Integración de web drivers o *frameworks*):** No tiene integración con web drivers de ningún tipo.
- **Contenedores de aplicaciones:** Sureassert realiza las pruebas de manera externa a los contenedores de aplicaciones.
- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...):** No dispone de componentes para realizar pruebas sobre la capa de persistencia.
- **Más utilidades:** Cuenta con anotaciones extendidas, *testing* automático en función a los cambios que se realicen en el proyecto e integración automática con el código existente.

Desarrollo

- **Herramientas para ayuda al desarrollo:** Este *framework* es una herramienta para Eclipse que analiza el código existente y genera clases de prueba para este. También

aporta una visualización de la cobertura de los tests del proyecto.

- **Herramientas de gestión y configuración de dependencias (Maven):** No tiene gestión de dependencias con Maven porque es un plug-in gestionado por el Marketplace de Eclipse.

Operación

- **Monitorización de los test (IDE o WEB):** Añade visualización de la cobertura de los tests a la interfaz que aporta JUnit sobre Eclipse.

Soporte

- **Facilidad de uso/aprendizaje:** Es fácil de usar ya que tiene una guía muy bien hecha que disminuye la curva de aprendizaje.
- **Calidad de la documentación:** La documentación está muy organizada pero tiene pocas clases documentadas con JavaDoc.
- **Entidades detrás del proyecto:** El proyecto ha sido desarrollado solamente por una persona, con la ayuda de algunos contribuyentes que han aportado mejoras.
- **Evolución (evolución histórica y actividad actual):** Sureassert salió a la luz en mayo de 2011 y dejó de publicarse en él nuevas funcionalidades en 2012.

En función a todas las descripciones que se han hecho de cada apartado, se ha generado la tabla con las puntuaciones siguientes:

Dimensión	Característica	Cobertura	
Ejecución	Integración con Spring	N	0%
	Integración con *frameworks* de tests unitarios (Junit, TestNG)	M	50%
	Inyección de dependencias (CDI)	N	0%
	Capa de presentación (Integración de web drivers o *frameworks*)	N	0%
	Contenedores de aplicaciones	N	0%
	Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...)	N	0%
	Más utilidades	M	50%
Desarrollo	Herramientas para ayuda al desarrollo	A	75%
	Herramientas de gestión y configuración de dependencias (Maven)	N	0%
Operación	Monitorización de los test (IDE o WEB)	M	50%
Soporte	Facilidad de uso/aprendizaje	T	100%
	Calidad de la documentación	A	75%
	Entidades detrás del proyecto	B	25%
	Evolución (evolución histórica y actividad actual)	N	0%

Tabla 54 Puntuaciones Sureassert UC

4.2.4.14. UISpec4J

Ejecución

- **Integración con Spring:** Puede utilizar las funcionalidades de Spring que dispone TestNG.

- **Integración con *frameworks* de tests unitarios (JUnit, TestNG):** Se puede integrar tanto con JUnit y TestNG.
- **Inyección de dependencias (CDI):** Dispone de inyección de dependencias de manera nativa si se usa con TestNG.
- **Capa de presentación (Integración de web drivers o *frameworks*):** Tiene posibilidad de probar interfaces de Java Swing y si se utiliza con TestNG también interfaces web mediante Selenium.
- **Contenedores de aplicaciones:** No hace falta porque se ejecuta fuera del contenedor.
- **Integración con *frameworks* de acceso a datos (Hibernate, JPA, myBatis...):** No dispone de implementaciones para el acceso a la capa de persistencia salvo la que provee TestNG.
- **Más utilidades:** Implementa funcionalidades para el *testing* de componentes de las interfaces hechas en Swing.

Desarrollo

- **Herramientas para ayuda al desarrollo:** No dispone de ayudas para el desarrollo.
- **Herramientas de gestión y configuración de dependencias (Maven):** Tiene disponibilidad de gestionar las dependencias por Maven.

Operación

- **Monitorización de los test (IDE o WEB):** No cuenta con ninguna monitorización propia para IDEs salvo la del propio JUnit.

Soporte

- **Facilidad de uso/aprendizaje:** Este *framework* es fácil de usar si se ha utilizado alguna vez JUnit y Swing porque básicamente sigue con los mismos comandos.
- **Calidad de la documentación:** La documentación es muy amplia sobre JavaDoc pero hay pocos manuales y ejemplos de uso.
- **Entidades detrás del proyecto:** El grupo de desarrolladores que ha creado este *framework* está formado por dos personas más otros 5 colaboradores.
- **Evolución (evolución histórica y actividad actual):** UISpec4J empezó en 2006 y continuó vivo hasta 2011 que dejó de recibir mejoras. Actualmente el proyecto está congelado en una versión estable del mismo.

En función a todas las descripciones que se han hecho de cada apartado, se ha generado la tabla con las puntuaciones siguientes:

Dimensión	**Característica**	**Cobertura**	
Ejecución	**Integración con Spring**	M	**50%**
	Integración con *frameworks* de tests unitarios (Junit, TestNG)	T	**100%**
	Inyección de dependencias (CDI)	M	**50%**
	Capa de presentación (Integración de web drivers o *frameworks*)	T	**100%**
	Contenedores de aplicaciones	N	**0%**
	Integración con *frameworks* de acceso a datos (Hibernate,	B	**25%**

	JPA, myBatis...)		
	Más utilidades	B	25%
Desarrollo	Herramientas para ayuda al desarrollo	N	0%
	Herramientas de gestión y configuración de dependencias (Maven)	T	100%
Operación	Monitorización de los test (IDE o WEB)	B	25%
Soporte	Facilidad de uso/aprendizaje	A	75%
	Calidad de la documentación	M	50%
	Entidades detrás del proyecto	B	25%
	Evolución (evolución histórica y actividad actual)	B	25%

Tabla 55 Puntuaciones UISpec4J

4.3. Análisis de los resultados

Tras el análisis exhaustivo de los *frameworks* que salieron de la primera evaluación con criterios, los datos han sido expuestos en una tabla con las puntuaciones de cada dimensión, la media y la media ponderada. La clasificación final delos *frameworks* queda como se puede ver en la Tabla 56 Resumen global de las puntuaciones ordenadas.

	Dimensión					
	Ejecución	Desarrollo	Operación	Soporte	Media	Media ponderada
TestNG	86%	100%	25%	100%	77,73%	91,18%
Arquillian	91%	80%	25%	88%	70,78%	86,03%
Unitils	94%	70%	25%	75%	65,94%	81,62%
FitNesse	64%	90%	75%	100%	82,27%	80,88%
Cucumber JVM	91%	70%	25%	75%	65,16%	80,15%
PaxEXAM	86%	60%	25%	75%	61,48%	76,47%
JMeter	44%	40%	75%	100%	64,69%	63,97%
Needle	53%	60%	25%	75%	53,28%	61,03%
DbUnit	52%	60%	25%	69%	51,33%	58,09%
Jnario	13%	90%	0%	94%	49,06%	52,21%
Cactus	34%	80%	25%	56%	48,91%	48,53%
UISpec4J	50%	60%	25%	44%	44,69%	48,53%
SpryTest	33%	40%	25%	31%	32,27%	33,09%
SureAssert UC	13%	30%	50%	50%	35,63%	29,41%
Promedio	57%	66%	32%	74%		

Tabla 56 Resumen global de las puntuaciones ordenadas

Como se aprecia en la tabla, los colores en azul en cada apartado muestran los *frameworks* que han sacado mejor nota en la dimensión y en color rojo los que peor.

Finalmente tras evaluar cada framework y puntuarlo, los tres que mejor nota han sacado según la media ponderada son TestNG, Arquillian y Unitils.

TestNG ha vencido gracias a que en Ejecución cubre casi todas las funcionalidades que se requerían para J-everis, tiene un conjunto de herramientas básicas muy buenas para el IDE Eclipse y tiene una espléndida documentación que le hace ser diferencial frente al resto de los *frameworks* a su altura.

Arquillian ha quedado segundo puesto que ha dado buena talla en todos los aspectos menos en Operación, pero la documentación de éste ha sido crítica para no poderle quitar el puesto a TestNG.

Finalmente Unitils tiene la mejor puntuación en lo que a Ejecución se basa y en las demás dimensiones obtiene unas notas aceptables que le hacen ser el 3er mejor framework de la comparativa. Cabe destacar que FitNesse ha quedado a la altura de Unitils ya que tiene una puntuación prácticamente similar y el equilibrio que hay en todas las dimensiones es mucho más alto que cualquier otro framework estudiado.

5. Conclusiones

Con el objetivo de descubrir el *framework* de testing que pueda cubrir más cualidades para la arquitectura J-everis se han analizado un conjunto muy extenso de *frameworks* de formas muy diversas y rigurosas.

Los *frameworks* se han evaluado en una primera iteración bajo ocho cualidades que de una manera muy abstracta describen su funcionalidad y su ámbito de aplicación. En total en esta primera iteración se analizaron 39 *frameworks* que van desde sencillos *frameworks* de *mock* hasta complejos *frameworks* que abarcan múltiples funcionalidades.

En la segunda iteración de evaluación se escogieron de entre los 39 *frameworks* un total de 14, que fueron los que obtuvieron una puntuación igual o superior a 4 puntos en la primera evaluación. Para crear los criterios de análisis se tuvo que estudiar qué partes serían más importantes a la hora de realizar un desarrollo con la arquitectura J-everis y tras haber pensado varias formas de estudio, se decidió tomar la opción del estudio en base a cuatro dimensiones.

Lo que se ha podido extraer del estudio de estos *frameworks* es que, aparte de los resultados finales, hay una gran cantidad de *frameworks* que sirven para realizar tareas para las que apenas se sabía que podían estar automatizadas o cubiertas.

Al final se ha proclamado TestNG como el *framework* más destacado de este estudio, gracias a lo completo y cuidado que es y como una alternativa real a poder sustituir a Junit (y a prácticamente el resto de los *frameworks* que están por encima en nivel de abstracción) gracias a la integración que éste tiene con los sistemas y la ayuda de la comunidad que lo mantiene. De todos modos, tampoco hay que perder de vista a los otros dos *frameworks* que han quedado después ya que son potentísimos y multifuncionales, incluso más que el ganador, pero que en algunos aspectos no han dado la talla.

Gracias a este estudio, se espera que Everis pueda tener un punto de partida serio para poder integrar *frameworks* de testing serios y darle la competitividad que necesita a la arquitectura de J-everis para estar a la altura de las circunstancias que el mercado necesita en los desarrollos ágiles. Para los equipos que requieran aplicar pruebas complejas servirá de manera importante a medio-largo plazo ya que supondrá una mejora en la calidad y robustez del código y también un acortamiento de los tiempos de entrega.

6. Bibliografía

[1] K. Beck, «Embrace Change,» de *Extreme Programming Explained*, Addison-Wesley professional, 1999.

[2] «Framework J-everis,» [En línea]. Available: http://quark.everis.int/confluence/display/ARCHEX/Framework+j-everis.

[3] «Interfaz de Programación de Aplicaciones,» [En línea]. Available: http://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones.

[4] «Behavior driven development,» [En línea]. Available: http://en.wikipedia.org/wiki/Behavior-driven_development.

[5] «Acceptance test driven development,» [En línea]. Available: http://en.wikipedia.org/wiki/Acceptance_test-driven_development.

[6] «Mock Object,» [En línea]. Available: http://en.wikipedia.org/wiki/Mock_object.

[7] «Unit Testing,» [En línea]. Available: http://en.wikipedia.org/wiki/Unit_testing.

[8] «Arquillian,» [En línea]. Available: http://arquillian.org/.

[9] «Arquillian Invasion!,» [En línea]. Available: http://arquillian.org/invasion/.

[10] «beanSpec: Project Web Hosting - Open Source Software,» [En línea]. Available: http://beanspec.sourceforge.net/.

[11] «Jakarta Cactus - Jakarta Cactus Project,» [En línea]. Available: https://jakarta.apache.org/cactus/.

[12] «Concordion,» [En línea]. Available: http://concordion.org/.

[13] «Concutest,» [En línea]. Available: http://www.cs.rice.edu/~mgricken/research/concutest/intro.shtml.

[14] «Cucumber JVM,» [En línea]. Available: https://github.com/cucumber/cucumber-jvm.

[15] «DbUnit - About DbUnit,» [En línea]. Available: http://dbunit.sourceforge.net/.

[16] «EasyMock,» [En línea]. Available: http://easymock.org/.

[17] «FitNesse FrontPage,» [En línea]. Available: http://www.fitnesse.org/.

[18] «GrandTestAuto,» [En línea]. Available: http://grandtestauto.org/.

[19] «GroboUtils Home Page,» [En línea]. Available: http://groboutils.sourceforge.net/.

[20] «HavaRunner,» [En línea]. Available: https://github.com/havarunner/havarunner.

[21] «Instinct - Instinct is a Behaviour Driven Development (BDD) framework for Java,» [En línea]. Available: https://code.google.com/p/instinct/.

[22] «jbehave - What is JBehave?,» [En línea]. Available: http://jbehave.org/.

[23] «JDave,» [En línea]. Available: http://jdave.org/.

[24] «SCGi: JExample,» [En línea]. Available: http://scg.unibe.ch/research/jexample.

[25] «Apache JMeter - Apache JMeter (TM),» [En línea]. Available: https://jmeter.apache.org/.

[26] «JMock,» [En línea]. Available: http://jmock.org/.

[27] «The JMockit testing toolkit,» [En línea]. Available: http://jmockit.github.io/.

[28] «The JMockit testing toolkit comparison matrix,» [En línea]. Available: http://jmockit.github.io/MockingToolkitComparisonMatrix.html.

[29] «Jnario - Executable Specifications for Java,» [En línea]. Available: http://jnario.org/.

[30] «JSST,» [En línea]. Available: https://github.com/shyiko/jsst.

[31] «Java testing tools: Static Analysis, code review, unit testing,» PARASOFT, [En línea]. Available: http://www.parasoft.com/jtest.

[32] «Jukito,» [En línea]. Available: https://github.com/ArcBees/Jukito.

[33] «Guice,» [En línea]. Available: https://github.com/google/guice.

[34] «JUnit - About,» [En línea]. Available: http://junit.org/.

[35] «JUnitEE,» [En línea]. Available: http://www.junitee.org/.

[36] «JWalk :: Lazy systematic testing tools suite,» [En línea]. Available: http://staffwww.dcs.shef.ac.uk/people/A.Simons/jwalk/.

[37] A. J. H. Simons, «JWalk: a tool for lazy, systematic testing of java classes by design introspection and user interaction,» *Automated Software Engineering,* vol. 14, nº 4, pp. 369-418, 2007.

[38] «mockito - simpler & better mocking,» [En línea]. Available: https://code.google.com/p/mockito/.

[39] «Mockrunner,» [En línea]. Available: http://mockrunner.sourceforge.net/.

[40] «Needle - Effective Unit Testing for Java EE - Overview,» [En línea]. Available: http://needle.spree.de/overview.

[41] «NU Tester: Educational Alternative to JUnit,» [En línea]. Available: https://code.google.com/p/nutester/.

[42] «Pax Exam - OPS4J Pax Exam 4.x - OPS4J Wiki,» [En línea]. Available: https://ops4j1.jira.com/wiki/display/PAXEXAM4/Pax+Exam.

[43] «Context and Dependency Inyection,» [En línea]. Available: http://www.oracle.com/technetwork/articles/java/cdi-javaee-bien-225152.html.

[44] «PowerMock,» [En línea]. Available: https://code.google.com/p/powermock/.

[45] «Selenium - Web Browser Automation,» [En línea]. Available: http://docs.seleniumhq.org/.

[46] «SpryTest,» [En línea]. Available: http://marketplace.eclipse.org/content/sprytest#.UyCgrfl5OL0.

[47] [En línea]. Available: Sureassert UC.

[48] «TestNG,» [En línea]. Available: http://testng.org/doc/index.html.

[49] «UISpec4J | Java/Swing GUI Testing Made Simple!,» [En línea]. Available:

http://www.uispec4j.org/.

[50] «Unitils,» [En línea]. Available: http://www.unitils.org/summary.html.

[51] «XMLUnit,» [En línea]. Available: http://xmlunit.sourceforge.net/.

[52] «Graphene,» [En línea]. Available: http://arquillian.org/blog/tags/graphene/. [Último acceso: 2014 4 25].

[53] D. Riehle, «Framework Design: A Role Modeling Approach,» 1999. [En línea]. Available: http://dirkriehle.com/computer-science/research/dissertation/index.html.

7. Anexos

7.1. Anexo I: Tabla comparativa global primeros criterios de selección

Nombre	Capa visual	Capa negocio	Capa datos	BDD	ATDD	Mock	Test Unitarios	Test Funcionales	Nota
Arquillian	Si	Si	Si	Si	Si	Si	Si	Si	8
Cucumber JVM	Si	Si	Si	Si	Si	Si	Si	Si	8
Unitils	Si	Si	Si	No	Si	Si	Si	Si	7
Cactus	Si	Si	No	No	Si	Si	Si	Si	6
FitNesse	Si	Si	Si	No	Si	No	Si	Si	6
TestNG	Si	Si	Si	No	Si	Si	Si	No	6
Jnario	Si	Si	No	Si	Si	No	Si	No	5
Needle	No	Si	Si	No	Si	Si	Si	No	5
DbUnit	No	Si	Si	No	Si	No	Si	No	4
JMeter	Si	No	Si	No	No	Si	No	Si	4
PaxEXAM	No	Si	No	No	Si	No	Si	Si	4
SpryTest	No	Si	No	No	Si	Si	Si	No	4
SureAssert UC	No	Si	No	No	Si	Si	Si	No	4
UISpec4J	Si	Si	No	No	No	No	Si	Si	4
Concordion	No	Si	No	No	Si	No	Si	No	3
Concutest	No	Si	No	No	No	No	Si	Si	3
Instinct	No	Si	No	Si	No	No	Si	No	3
JBehave	No	Si	No	Si	No	No	Si	No	3
JDave	No	Si	No	Si	No	No	Si	No	3

Jukito	No	Si	No	No	No	Si	Si	No	3
JUnitEE	No	Si	No	No	No	No	Si	Si	3
GrandTestAuto	No	Si	No	No	No	No	Si	No	2
GroboUtils	No	Si	No	No	No	No	Si	No	2
HavaRunner	No	Si	No	No	No	No	Si	No	2
JExample	No	Si	No	No	No	No	Si	No	2
JSST	No	Si	No	No	No	No	Si	No	2
JTest	**No**	**Si**	**No**	**No**	**No**	**No**	**Si**	**No**	2
Junit	No	Si	No	No	No	No	Si	No	2
Jwalk	No	Si	No	No	No	No	Si	No	2
NUTester	No	Si	No	No	No	No	Si	No	2
Selenium	Si	No	No	No	No	No	No	Si	2
XMLUnit	No	Si	No	No	No	No	Si	No	2
BeanSpec	No	No	No	Si	No	No	No	No	1
EasyMock	-	-	-	No	No	Si	No	No	1
JMock	-	-	-	No	No	Si	No	No	1
JMockit	-	-	-	No	No	Si	No	No	1
Mockito	-	-	-	No	No	Si	No	No	1
Mockrunner	-	-	-	No	No	Si	No	No	1
PowerMock	-	-	-	No	No	Si	No	No	1

Tabla 57 Tabla comparativa global primeros criterios de selección

7.2. Anexo IIa: Comparativa exhaustiva conjunta de los *frameworks*, parte 1

Área	Característica	Coeficiente de peso	Arquillian		Cactus		Unitils		FitNesse		Needle		Cucumber JVM		DbUnit	
			Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones	
Ejecución	Integración con Spring	3	100%	3,0	0%	0,0	100%	3,0	100%	3,0	0%	0,0	100%	3,0	100%	3,0
	Integración con frameworks de tests unitarios (Junit, TestNG)	3	100%	3,0	25%	0,8	100%	3,0	50%	1,5	100%	3,0	100%	3,0	50%	1,5
	Inyección de dependencias (CDI)	1	100%	1,0	0%	0,0	100%	1,0	50%	0,5	75%	0,8	100%	1,0	25%	0,3
	Capa de presentación (Integración de WebDrivers o frameworks)	2	100%	2,0	75%	1,5	75%	1,5	75%	1,5	50%	1,0	100%	2,0	0%	0,0
	Contenedores de aplicaciones	3	100%	3,0	100%	3,0	100%	3,0	0%	0,0	0%	0,0	100%	3,0	0%	0,0
	Integración con frameworks de acceso a datos (Hibernate, JPA, myBatis...)	3	50%	1,5	0%	0,0	100%	3,0	100%	3,0	100%	3,0	50%	1,5	100%	3,0
	Más utilidades	1	100%	1,0	25%	0,3	100%	1,0	75%	0,8	75%	0,8	100%	1,0	50%	0,5
		16		91%		34%		97%		64%		53%		91%		52%

Área	Característica	Coeficiente de peso	Arquillian		Cactus		Unitils		FitNesse		Needle		Cucumber JVM		DbUnit	
			Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones	
Desarrollo	Herramientas para ayuda al desarrollo	2	50%	1,0	50%	1,0	25%	0,5	75%	1,5	0%	0,0	25%	0,5	0%	0,0
	Herramientas de gestión y configuración de dependencias (Maven)	3	100%	3,0	100%	3,0	100%	3,0	100%	3,0	100%	3,0	100%	3,0	100%	3,0
		5		80%		80%		70%		90%		60%		70%		60%

Área	Característica	Coeficiente de peso	Arquillian		Cactus		Unitils		FitNesse		Needle		Cucumber JVM		DbUnit	
			Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones	
Operación	Monitorización de los test (IDE o WEB)	1	25%	0,3	25%	0,3	25%	0,3	75%	0,8	25%	0,3	25%	0,3	25%	0,3
		1		25%		25%		25%		75%		25%		25%		25%

Área	Característica	Coeficiente de peso	Arquillian		Cactus		Unitils		FitNesse		Needle		Cucumber JVM		DbUnit	
			Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones		Puntuaciones	
Soporte	Facilidad de uso/aprendizaje	3	50%	1,5	75%	2,3	100%	3,0	100%	3,0	100%	3,0	50%	1,5	50%	1,5
	Calidad de la documentación	3	100%	3,0	50%	1,5	100%	3,0	100%	3,0	100%	3,0	75%	2,3	100%	3,0
	Entidades detrás del proyecto	3	100%	3,0	100%	3,0	25%	0,8	100%	3,0	50%	1,5	75%	2,3	50%	1,5
	Evolución (evolución histórica y actividad actual)	3	100%	3,0	0%	0,0	75%	2,3	100%	3,0	50%	1,5	100%	3,0	75%	2,3
		12		88%		56%		75%		100%		75%		75%		69%

Leyenda

Coeficiente de peso	
Necesario	3
Importante	2
Suficiente	1

Tabla 58 Comparativa exhaustiva conjunta de los frameworks, parte 1

7.3. Anexo IIb: Comparativa exhaustiva conjunta de los *frameworks*, parte 2

			JMeter		Jnario		PaxEXAM		SpryTest		TestNG		UISpec4J		SureAssert UC	
Área	**Característica**	**Coeficiente de peso**	**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**	
Ejecución	Integración con Spring	3	50%	1,5	0%	0,0	100%	3,0	0%	0,0	100%	3,0	50%	1,5	0%	0,0
	Integración con frameworks de tests unitarios (Junit, TestNG)	3	0%	0,0	50%	1,5	100%	3,0	50%	1,5	100%	3,0	100%	3,0	50%	1,5
	Inyección de dependencias (CDI)	1	0%	0,0	0%	0,0	100%	1,0	0%	0,0	100%	1,0	50%	0,5	0%	0,0
	Capa de presentación (Integración de WebDrivers o frameworks)	2	100%	2,0	25%	0,5	50%	1,0	0%	0,0	75%	1,5	100%	2,0	0%	0,0
	Contenedores de aplicaciones	3	0%	0,0	0%	0,0	100%	3,0	100%	3,0	100%	3,0	0%	0,0	0%	0,0
	Integración con frameworks de acceso a datos (Hibernate, JPA, myBatis...)	3	100%	3,0	0%	0,0	75%	2,3	0%	0,0	50%	1,5	25%	0,8	0%	0,0
	Más utilidades	1	50%	0,5	0%	0,0	50%	0,5	75%	0,8	75%	0,8	25%	0,3	50%	0,5
		16		44%		13%		86%		33%		86%		50%		13%

			JMeter		Jnario		PaxEXAM		SpryTest		TestNG		UISpec4J		SureAssert UC	
Área	**Característica**	**Coeficiente de peso**	**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**	
Desarrollo	Herramientas para ayuda al desarrollo	2	100%	2,0	75%	1,5	0%	0,0	100%	2,0	100%	2,0	0%	0,0	75%	1,5
	Herramientas de gestión y configuración de dependencias (Maven)	3	0%	0,0	100%	3,0	100%	3,0	0%	0,0	100%	3,0	100%	3,0	0%	0,0
		5		40%		90%		60%		40%		100%		60%		30%

			JMeter		Jnario		PaxEXAM		SpryTest		TestNG		UISpec4J		SureAssert UC	
Área	**Característica**	**Coeficiente de peso**	**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**	
Operación	Monitorización de los test (IDE o WEB)	1	75%	0,8	0%	0,0	25%	0,3	25%	0,3	25%	0,3	25%	0,3	50%	0,5
		1		75%		0%		25%		25%		25%		25%		50%

			JMeter		Jnario		PaxEXAM		SpryTest		TestNG		UISpec4J		SureAssert UC	
Área	**Característica**	**Coeficiente de peso**	**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**		**Puntuaciones**	
Soporte	Facilidad de uso/aprendizaje	3	100%	3,0	100%	3,0	50%	1,5	100%	3,0	100%	3,0	75%	2,3	100%	3,0
	Calidad de la documentación	3	100%	3,0	100%	3,0	50%	1,5	25%	0,8	100%	3,0	50%	1,5	75%	2,3
	Entidades detrás del proyecto	3	100%	3,0	75%	2,3	100%	3,0	0%	0,0	100%	3,0	25%	0,8	25%	0,8
	Evolución (evolución histórica y actividad actual)	3	100%	3,0	100%	3,0	100%	3,0	0%	0,0	100%	3,0	25%	0,8	0%	0,0
		12		100%		94%		75%		31%		100%		44%		50%

Leyenda

Coeficiente de peso	
Necesario	3
Importante	2
Suficiente	1

Tabla 59 Comparativa exhaustiva conjunta de los frameworks, parte 2