



1 Introducción y principios Ágiles

La metodología de Ágil se caracteriza por incorporar cambios con rapidez en cualquier fase del proyecto.

El objetivo de esta metodología de trabajo es que el equipo sea capaz de gestionar los proyectos de una forma más fácil. Algunas claves para conseguirlo son:

- Utilizar equipos pequeños y autoorganizados.
- Priorizar los puntos de vista del cliente, “QUÉ es lo que se desea realizar”.
- Entregas frecuentes.
- Reuniones diarias con el equipo de desarrollo.
- Transparencia total, buena comunicación entre los miembros del proyecto.

1.1 Disciplined Agile Delivery (DAD)

DAD es un framework híbrido creado por Scott Ambler, que se basa en los principios ágiles y combina prácticas extraídas de Scrum, XP, Kanban, Lean, DevOps y entrega y despliegue continuo.

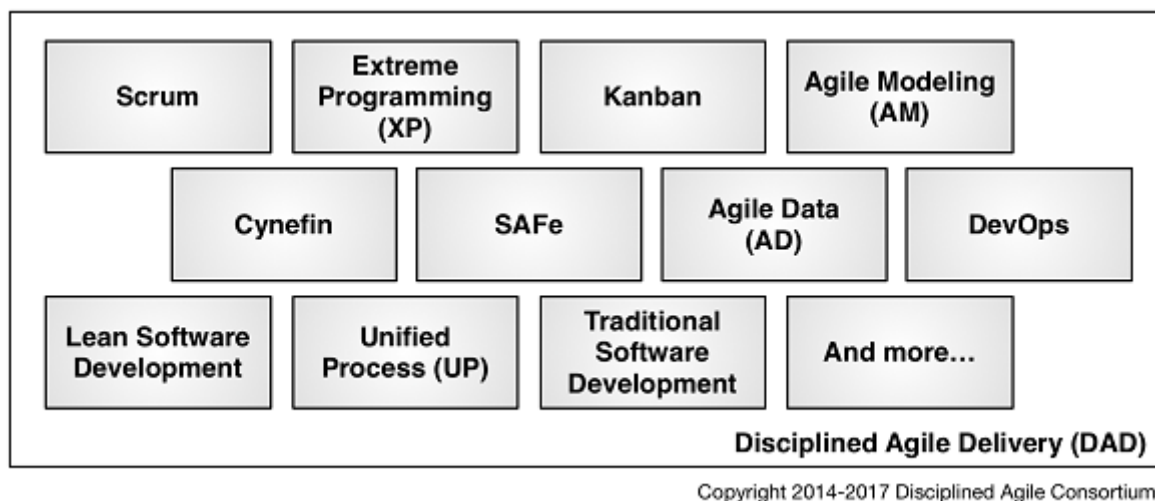


Figura 1 Un Marco de Trabajo Híbrido

Esta disciplina a diferencia de otras metodologías, no se centra exclusivamente en el proceso de construcción de un proyecto de software, sino que abarca todo su ciclo de vida, desde su concepción, hasta su puesta en producción.



Figura 2 *Ciclo de vida de entrega completa*

El objetivo de DAD es ayudar a las empresas a implantar los valores y principios ágiles desde la definición de los requisitos, pasando por el desarrollo hasta el despliegue y entrega del software al cliente.

1.1.1 Características de DAD

- **Centrado en las personas** (people first). En DAD se fomentan los equipos multidisciplinarios conformados por personas con habilidades distintas y cruzadas. Los miembros de los equipos deben tener múltiples habilidades y realizar actividades en distintas disciplinas fuera de su especialidad.
- **Orientado al aprendizaje.** Existen tres aspectos clave que deben ser atendidos por un ambiente de aprendizaje. El primero es el aprendizaje del dominio: ¿cómo exploras e identificas las necesidades del cliente?, ¿cómo fomentas que el equipo obtenga este aprendizaje? El segundo aspecto se enfoca en el aprendizaje para mejorar el proceso a todos los niveles: individual, de equipo y de organización. El tercer aspecto es el aprendizaje técnico que se enfoca en entender cómo trabajar de forma efectiva con las herramientas y tecnologías utilizadas para crear la solución.
- **Agilidad.** El framework DAD se adhiere y amplifica los valores y principios del Manifiesto Ágil.
- **Híbrido.** DAD es un framework híbrido porque adopta y personaliza técnicas de métodos ágiles existentes tales como Scrum, Extreme Programming (XP), Agile Data (AD), Agile Modeling (AM), Unified Process (UP) y Kanban, por nombrar algunos. DAD es un framework ágil de segunda generación, que aprende y aprovecha los frameworks ágiles anteriores.
- **Enfocado en soluciones.** DAD mueve el enfoque de tan solo producir software, a proveer soluciones que aportan verdadero valor de negocio, considerando las restricciones económicas, culturales y técnicas. Sí, por supuesto que el software es



importante, pero el despliegue exitoso de soluciones de TI típicamente involucra no solo software, sino también adquirir hardware, ajustar procesos de negocio y operativos, e incluso puede impactar la estructura organizacional de los involucrados.

- **Enfocado en la entrega.** El ciclo de vida básico de DAD, mostrado en la *figura 10*, considera desde el inicio del proyecto, pasando por la construcción del producto y la liberación de la solución en producción (incluso muestra algunas actividades de gestión de portafolio de proyectos previas a la iniciación del proyecto, así como algunas actividades posteriores a la liberación en producción). Esto difiere de los métodos ágiles de primera generación que típicamente se enfocan en las actividades de construcción del software, ignorando el resto del ciclo de vida necesario para desarrollar e implantar exitosamente una solución, especialmente en un contexto corporativo.
- **Dirigido por metas.** Uno de los retos de describir un marco de procesos es que necesitas proveer suficiente información que sirva como guía para ayudar a las personas a entender el marco, pero si se provee demasiada información entonces puede ser tomado como una “receta” y sabemos que eso no es bueno. Para resolver este reto, el framework DAD está dirigido por metas, que se resumen en la *figura 12*. Esta estrategia dirigida por metas, provee la cantidad adecuada de orientación para que el equipo comprenda el proceso, al mismo tiempo que es suficientemente flexible para que puedan personalizarlo al contexto correspondiente.
- **Dirigido por riesgo y valor.** El framework DAD adopta lo que se conoce como un ciclo de vida dirigido por riesgo y valor (risk/value lifecycle), que viene a ser una versión ligera de la estrategia propuesta por el Proceso Unificado. Los equipos DAD se enfocan en resolver los principales riesgos del proyecto, tales como: lograr consenso de la visión de la solución a desarrollar y probar su arquitectura en etapas tempranas del ciclo de vida. DAD también incorpora revisiones explícitas de la viabilidad del proyecto, si la funcionalidad provista es suficiente, y si la solución está lista para producción. El que también esté dirigido por valor provoca que en DAD los equipos buscan generar soluciones utilizables periódicamente.
- **Escalable.** DAD provee una base escalable para TIs ágiles y es una parte importante de la estrategia de IBM de “agilidad a escala”. Dicha estrategia enfatiza que escalar un proceso tiene que ver con muchos aspectos más allá del tamaño del equipo. Existen muchos otros factores que deben ser considerados al escalar procesos ágiles a nivel organizacional. Entre estos factores están la distribución geográfica del equipo, complejidad (técnica y del dominio), regulaciones que deben ser satisfechas (compliance), estructura organizacional, complejidad organizacional, y disciplina de la empresa. Cada equipo se encontrará en una situación única y debe personalizar su estrategia al contexto en que se encuentra.



1.1.2 El Manifiesto de la Disciplina Ágil (“The Disciplined Agile Manifesto”)

El Manifiesto Ágil Disciplinado es una extensión del original “Manifiesto para desarrollar Software Ágil” (escrito en 2001), en el que se reflejan las filosofías detrás del marco de trabajo DAD.

Los valores en que se centra la disciplina de entrega ágil son los siguientes:



1.1.2.1 Los Principios Detrás del Manifiesto de la Agilidad Disciplinada

1. La máxima prioridad es satisfacer a los interesados mediante la **temprana entrega** y continuada de soluciones con valor de negocio.
2. Se **adapta a los continuos cambios** en los requisitos, incluso al final del ciclo de vida de la entrega de la solución. Los procesos ágiles aprovechan el cambio para dar ventaja competitiva al cliente.
3. Se **entregan soluciones consumibles con frecuencia**, entre 2 semanas y 2 meses, prefiriendo acortar la escala de tiempo.



4. Los **interesados y los desarrolladores trabajan juntos** cada día durante el proyecto.
5. **Equipos pequeños autodisciplinados** y motivados.
6. El método más eficiente y efectivo para comunicar información al equipo y dentro de éste, es la conversación cara a cara.
7. Las soluciones consumibles son la principal medida del avance.
8. Los procesos ágiles promueven la **entrega sostenible**. Los promotores, desarrolladores y usuarios deberían poder mantener un **ritmo constante** indefinidamente.
9. La atención continua a la **excelencia técnica y al buen diseño** fundamenta la agilidad.
10. **Simplicidad** (el arte de maximizar la cantidad de trabajo no hecho) es fundamental.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. El equipo reflexiona como ser más efectivo a intervalos regulares, y entonces afina y ajusta su funcionamiento en consecuencia.
13. Aprovechar y evolucionar los activos del ecosistema de la organización, en colaboración con las personas responsables de éstos.
14. Visualizar el flujo de trabajo para ayudar a alcanzar un flujo de entrega uniforme y mantener una **velocidad de trabajo regular**.
15. El ecosistema organizativo debe evolucionar para reflejar y realzar los esfuerzos de los equipos ágiles, a la vez que sea suficientemente flexible para dar soporte a equipos no ágiles o híbridos.

1.1.3 Roles DAD

DAD se definen dos conjuntos de roles. El primer conjunto son los roles primarios que engloban a aquellos roles que aparecen sin importar el nivel de escalado aplicado en el método. El segundo conjunto lo conforman los roles secundarios que aparecen cuando empezamos a escalar el método.

1.1.3.1 Roles Primarios

- **Team Lead.** Es el líder del equipo, gestiona al equipo, planifica y estima las tareas y se encarga de crear y mantener las condiciones que permiten al equipo tener éxito. Es equivalente al rol de Scrum Master. Un aspecto importante de los equipos autoorganizados es que el líder del equipo facilita o guía al equipo en la realización de actividades de gestión técnica en lugar de asumir estas responsabilidades por sí mismo.
- **Team Member.** Los miembros del equipo son los desarrolladores de la solución para los interesados. Sin embargo, en DAD no todos los miembros del equipo necesariamente escriben código. Los miembros del equipo identificarán las tareas, estimarán las tareas, se "registrarán" para las tareas, realizarán las tareas y rastrearán su estado hasta su finalización.
- **Stakeholder.** Los interesados, ese gran grupo de todos los que son o creen ser afectados por el proyecto. En este sentido, la parte interesada es claramente más que un



usuario final: una parte interesada podría ser un usuario directo, usuario indirecto, administrador de usuarios, gerente sénior, miembro del personal de operaciones, profesionales de mantenimiento potencialmente afectados por el desarrollo y / o despliegue de un proyecto de software, etc.

- **Product Owner.** Es el responsable del producto cuya función es la de introducir las épicas conforme las necesidades y prioridades del cliente. Literalmente es “la única voz del cliente”. Él o ella representa las necesidades y deseos de la comunidad de partes interesadas para el equipo de entrega ágil. Tener un propietario de producto trabajando estrechamente con el equipo de desarrollo para responder cualquier pregunta sobre los elementos de trabajo a medida que se implementan, reduce sustancialmente la necesidad de requisitos, pruebas y documentación de diseño. Un objetivo secundario para el propietario de un producto es representar el trabajo del equipo ágil ante la comunidad de interesados. Esto incluye organizar demostraciones de la solución a medida que evoluciona y comunicar el estado del proyecto a las partes interesadas clave.
- **Architecture Owner.** La Arquitectura es una de las fuentes clave de riesgo del proyecto, por lo que este rol se encarga de que el equipo mitigue el riesgo y de definir la arquitectura a utilizar. El propietario de la arquitectura es la persona que decide la arquitectura para el equipo y que facilita la creación y la evolución del diseño de la solución general.

1.1.3.2 Roles Secundarios

- **Specialist.** Son especialistas en algunos temas de escalado como Analistas de negocio, etc.
- **Domain Expert.** Experto de un dominio es alguien que conoce mucho sobre un determinado tema. El propietario del producto a veces traerá expertos de dominio para trabajar con el equipo, por ejemplo, un experto para explicar los detalles de un requerimiento, etc.
- **Technical Expert.** Son expertos en determinados temas técnicos como base de datos, servidor de aplicaciones, redes, seguridad, calidad, etc.
- **Independent Tester.** Es un equipo de pruebas independiente que trabaja en paralelo al equipo de desarrollo para validar su trabajo durante todo el ciclo de vida del producto.
- **Integrator.** Cuando los sistemas son muy grandes y están divididos en subequipos suele ser necesario este rol para encargarse de generar el producto.

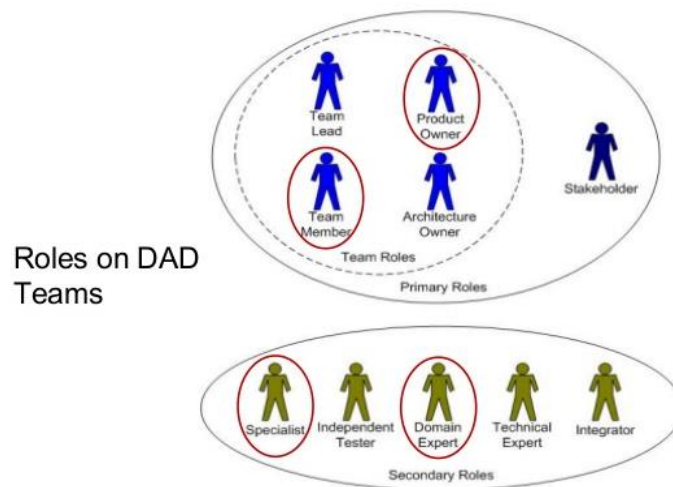


Figura 3 Roles DAD

1.1.4 Ciclo de Vida Básico de DAD

El ciclo de vida básico de DAD es una extensión del ciclo de vida de Scrum y provee el ciclo iterativo para la planificación y construcción de valor añadido.

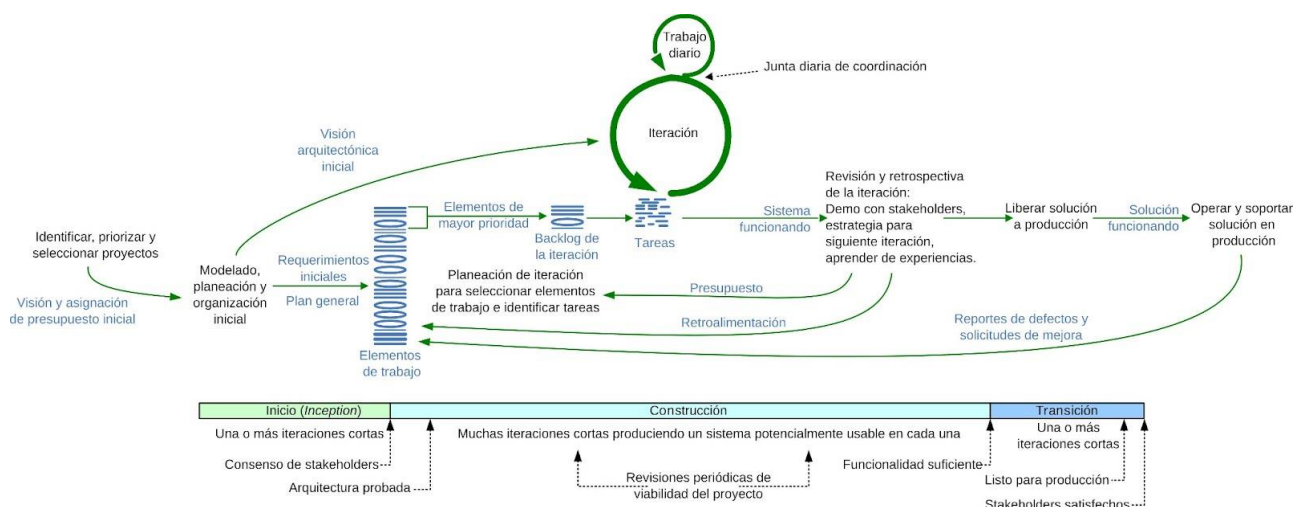


Figura 4. Ciclo de vida básico de DAD

Dicho ciclo de vida consta de 3 etapas:

- **Inicio / Concepción:** es la fase inicial de un proyecto DAD. Esta fase debe ser lo más corta posible y que el objetivo principal, es el de acordar una visión común a alto nivel sobre el proyecto con los stakeholders, o partes interesadas, de la manera más eficiente posible. En ella se reflejará una la visión del alcance, el cronograma, el presupuesto, posibles restricciones relevantes, riesgos clave y la arquitectura de la solución. También



se apremia a empezar a construir nuestro equipo en esta fase, planificando su composición y si hará falta contratar a algún miembro adicional.

- **Construcción:** es la fase de implementación del producto cuya premisa principal es “construir una solución consumible de manera incremental” siendo esta una extensión a Scrum.

Se podría dividir esta fase en tres tareas secuenciales:

- **Planificación de la Iteración:** El responsable de producto deberá indicar las historias de usuario que se van a desarrollar en la iteración, teniendo en cuenta la prioridad y el valor de negocio de las mismas.
La herramienta que ayudará a la planificación de las tareas será **Gestor de Proyectos / Tareas** a través de una pizarra de tareas “**Kanban**”.
- **Desarrollo:** Corresponde al periodo de terminar el trabajo de la iteración que corresponda. Es donde se implementan las diferentes funcionalidades indicadas en las historias de usuario.
La herramienta que se utilizará es un entorno de integración continua **Jenkins**.
- **Estabilización:** Es el conjunto de actividades que certifican que el sistema desarrollado es entregable (diferentes tipologías de pruebas). En este intervalo, también se harían las demostraciones y una retrospectiva de la iteración.

Algunos de los aceleradores para incrementar la productividad y satisfacción final del cliente son:

- **Tener un ejecutable en cada iteración.** Realizar una solución consumible es vital para garantizar la meta de la iteración.
- **Demostraciones frecuentes con las partes interesadas:** Las demos ofrecen información real del avance del sistema y una retroalimentación de todas las partes interesadas.
- **Transición:** En esta fase se deben superar las últimas pruebas de aceptación y preparar a las partes interesadas para la puesta en producción de la solución. Esto puede incluir actividades como asegurar los entornos de producción, avisar a los usuarios que van a disponer de una nueva versión de la solución, e incluso proporcionar formación si fuera necesario.

En un nivel más bajo de abstracción, se pueden observar dos etapas más en el ciclo de vida de DAD.

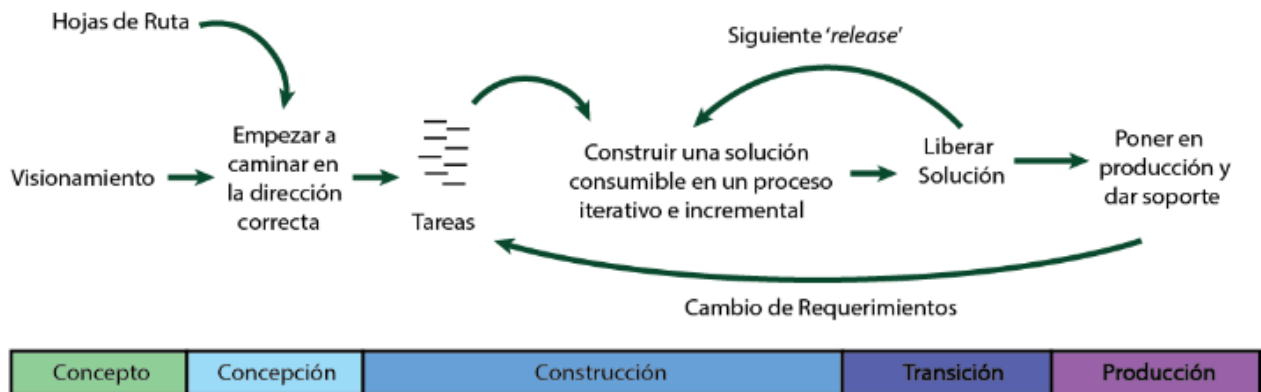


Figura 5. Ciclo de Vida de Entrega Extendido Completamente Ágil

Una etapa antes de la de concepción llamada **Concepto**. Hablando en términos empresariales, en ésta etapa se toman en cuenta proyectos potenciales y productos a ser definidos, identificados, priorizados y presentados a los inversores como propuesta, para posteriormente empezar la etapa de Concepción. Por otro lado, después de la Transición existe una etapa en la que la solución es puesta en **Producción** y el equipo se encarga de dar el soporte adecuado, ya sea para mantenerla o retirarla de producción.

1.1.5 Metas DAD

El marco de trabajo DAD se lleva a cabo mediante un desarrollo guiado por metas u objetivos, con el propósito de tenerlas como guías para todas las personas involucradas en el proyecto.



Figura 6. Impulsado por Metas

La *figura 12* resume las metas de proceso para toma de decisiones sugeridas por el marco de trabajo DAD. Existen en total **22 metas de proceso**, cada una es explicada en un diagrama de meta de proceso. El equipo de desarrollo ágil decidirá cómo aplicar cada meta de proceso, de manera que se refleje la situación en la que se encuentren.

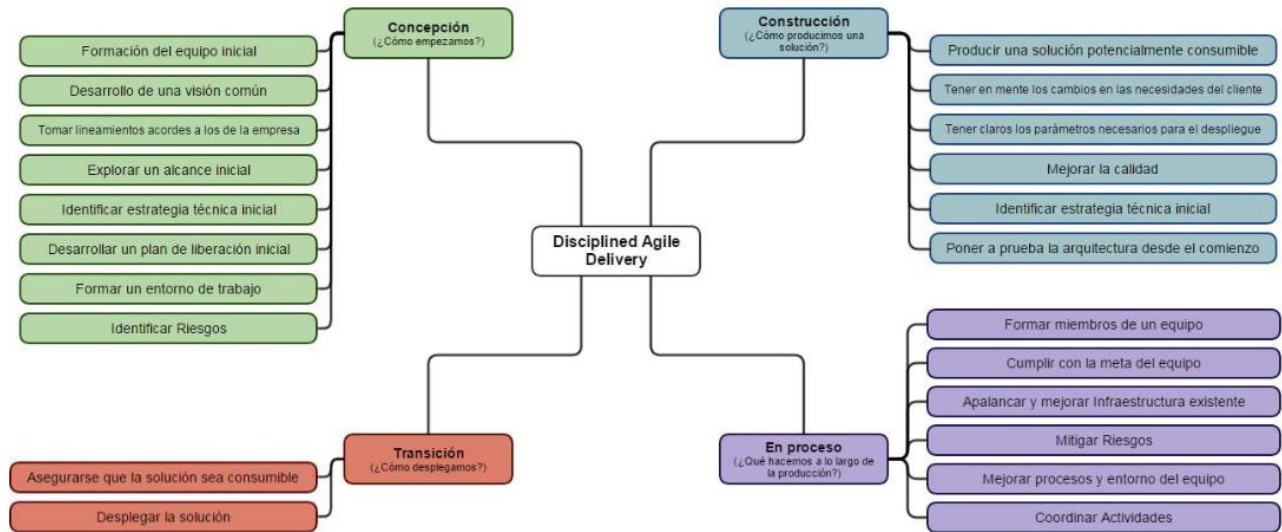


Figura 7. Metas de proceso a lo largo de un proyecto DAD

1.2 Aseguramiento de la Calidad

1.2.1 Mejora Continua

Constituye un elemento fundamental en los sistemas de gestión de calidad, y se conceptualiza mediante un Ciclo de Deming o PDCA (Plan-Do-Check-Act o traducido al castellano como **Planificar-Hacer-Verificar-Actuar**). En esencia, se trata de una filosofía en la que el error se asume como elemento clave del proceso de mejora hacia la excelencia en la gestión. De esta forma, realizando iteraciones del ciclo, se consigue aumentar progresivamente el nivel de calidad y la madurez de la organización.

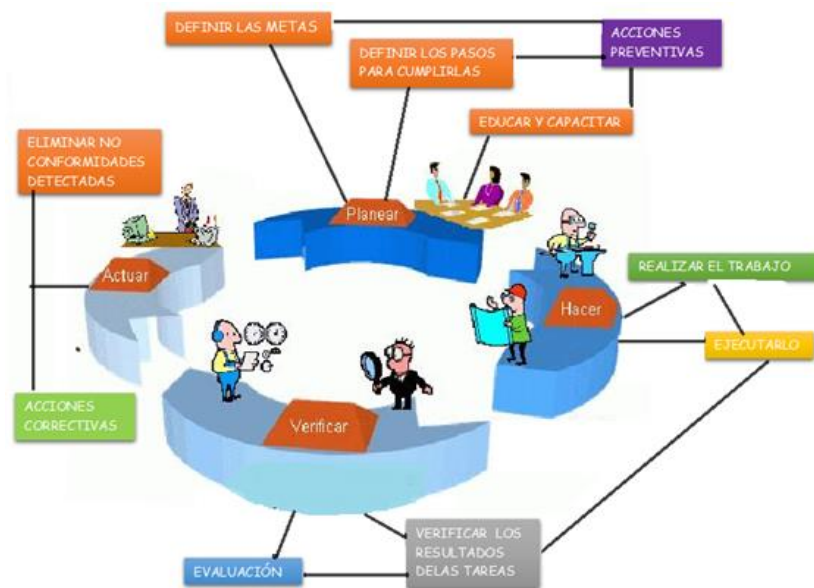


Figura 8. Ciclo de Deming o PDCA (Plan-Do-Check-Act)

Las actividades principales de este proceso, se centrarán en la evaluación de los productos contra los estándares y atributos de calidad previamente establecidos, así como la monitorización de los procesos para asegurar el cumplimiento de buenas prácticas de codificación.

1.2.1.1.1 Integración Continua

En la fase de construcción de la metodología DAD, se utilizará un entorno de integración continua que velará en todo momento que el código fuente que se desarrolla cumpla los más altos estándares de calidad. Dichos estándares estarán basados en las convenciones de código SUN y en la Guía de Buenas Prácticas Programación Ágil así como en la Guía de Buenas Prácticas Seguridad en Java.

El uso de una plataforma de integración continua, permite la automatización directa y planificación de diversos ámbitos del desarrollo como la compilación, pruebas y despliegue así como el mantenimiento de histórico de artefactos y versiones. Este conjunto de herramientas permiten la automatización y simplificación de las tareas de gestión y el control de calidad de todo el ciclo de vida de los proyectos de desarrollo software permitiendo:

- Conocer el **estado en tiempo real del proyecto** en desarrollo.
- **Generar la documentación técnica** y de seguimiento del proyecto.
- Modelar la información y sus flujos de trabajo.
- **Automatizar la estabilidad del código generado** en el proyecto.
- Poseer información en tiempo real de la **calidad del código generado** con indicadores configurables.



Toda empresa debería de tener una infraestructura como la siguiente para el apoyo a la gestión y aseguramiento de la calidad del proyecto:

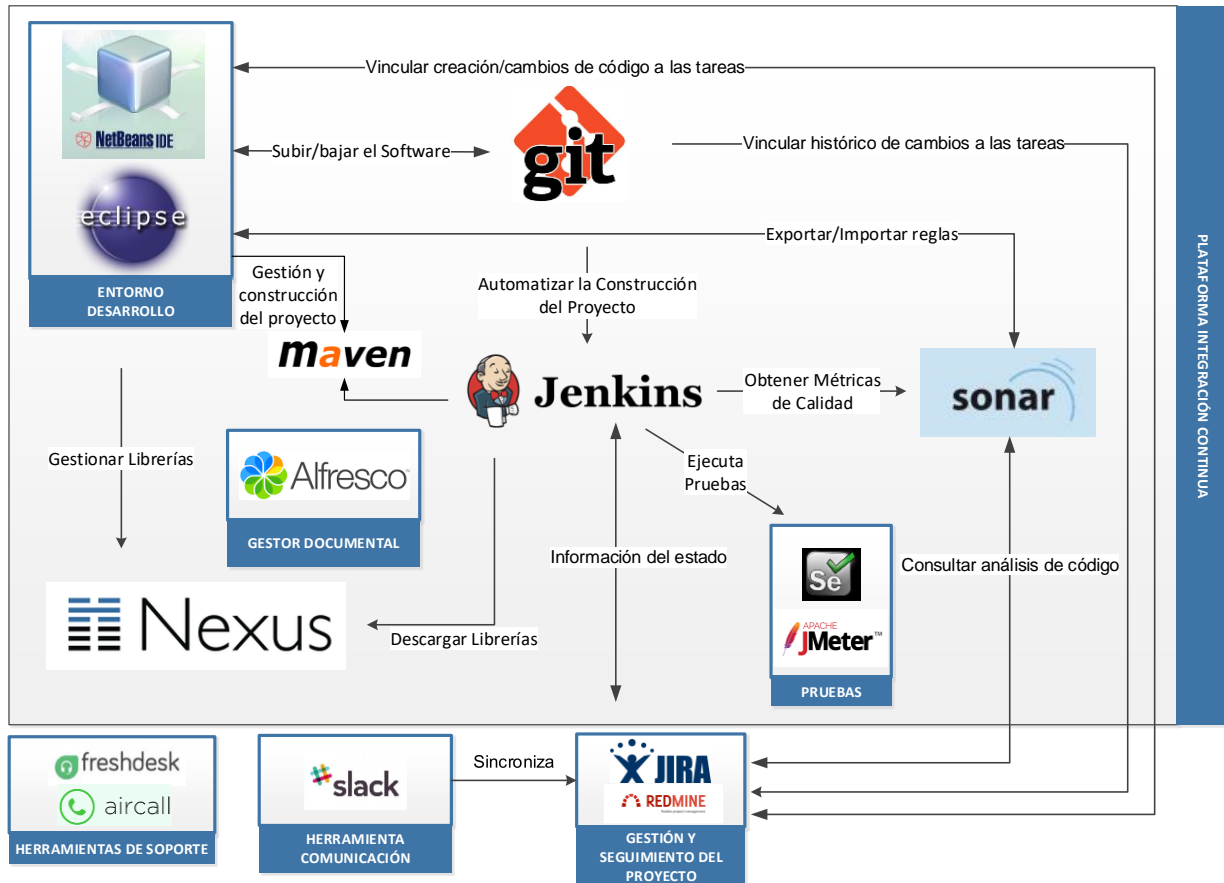


Figura 9 Infraestructura software para la prestación del servicio de aseguramiento de calidad

- Herramienta **Gestión de Proyecto.** → Jira Software | Redmine (Pizarra Kanban)
- Herramienta **Gestión Código Fuente.** → Git, SVN
- Herramienta **Gestión Dependencias.** → Nexus, Artifactory
- Herramienta **Integración Continua del Proyecto.** → Jenkins, Bambun
- Herramienta **Análisis Calidad Código.** → SonarQube
- Herramienta **Gestión Incidencias.** → Jira Software | Redmine
- Herramienta **Gestión Documental** → Alfresco
- Herramienta **Comunicación** → Slack
- Herramienta **Pruebas** → JMeter, SoapUI, Selenium GRID
- Herramienta **Soporte** → AirCall, FreshDesk

A continuación, se explica brevemente las herramientas propuestas para cubrir todo el ciclo de vida del producto:



- **ECLIPSE: entorno desarrollo.** Se configurará para la realización de técnicas avanzadas de refactorización y análisis de código.
- **MAVEN: construcción del proyecto,** abarcando desde la compilación hasta la distribución, despliegue y documentación de los proyectos, tratando de forma automática las dependencias del proyecto.
- **GIT:** repositorio de código del proyecto para el **control de versiones.**
- **NEXUS:** repositorio de dependencias del proyecto que permitirá gestionar eficientemente todos los artefactos que se necesitan y se generan durante el desarrollo del producto.
- **JENKINS:** herramienta para la construcción automática e **integración continua** del desarrollo software.
- **SONARQUBE:** herramienta para el control de la **calidad del código estático** con reporte de métricas y violaciones de código basadas en el estándar de convención de código SUN y OWASP.
- **JIRA | REDMINE:** Estas herramientas permiten una **gestión completa y seguimiento de los proyectos** mediante la configuración de las siguientes funcionalidades: cuadros de mando, planificación del proyecto, seguimiento de tareas, informes, imputación de horas, etc.
- **ALFRESCO:** **gestor documental** para llevar un versionado y un flujo aprobación de los documentos generados.
- **SLACK:** **sistema de mensajería** en tiempo real para la comunicación organizativa.
- **AIRCALL:** **sistema de Telefonía y centralita virtual** con integración telefónica (CTI) con su software de CRM y helpdesk favoritos.
- **HELPDESK:** **herramienta de soporte y de asistencia** que tiene como objetivo resolver incidencias TIC (Tecnologías de la información y comunicación) y productos similares.

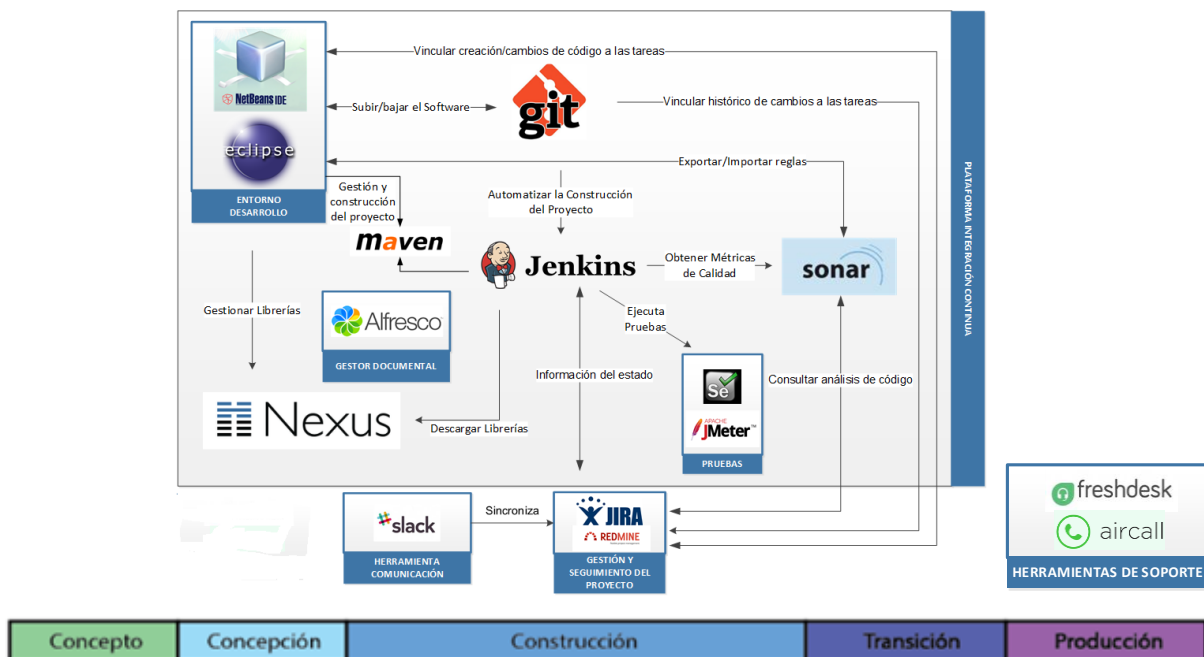


Figura 10. Ciclo de Vida de Entrega Extendido Completamente Ágil



- **Inicio / Concepción:** Para ayudar a gestionar el proyecto se utilizará por ejemplo Jira Software, que haciendo uso de pizarras de tareas se podrán controlar los Riesgos y el Listado de Ítems de Trabajo (en Scrum Product Backlog) del proyecto. Será en esta fase en la que el responsable del producto creó las épicas o historias de usuario (requisitos) para realizar una planificación en la fase posterior por Iteraciones (en Scrum Sprints). Dicha herramienta posee gráficas de indicadores para ver la velocidad que el equipo de desarrollo va logrando los objetivos marcados en la planificación del Sprint.

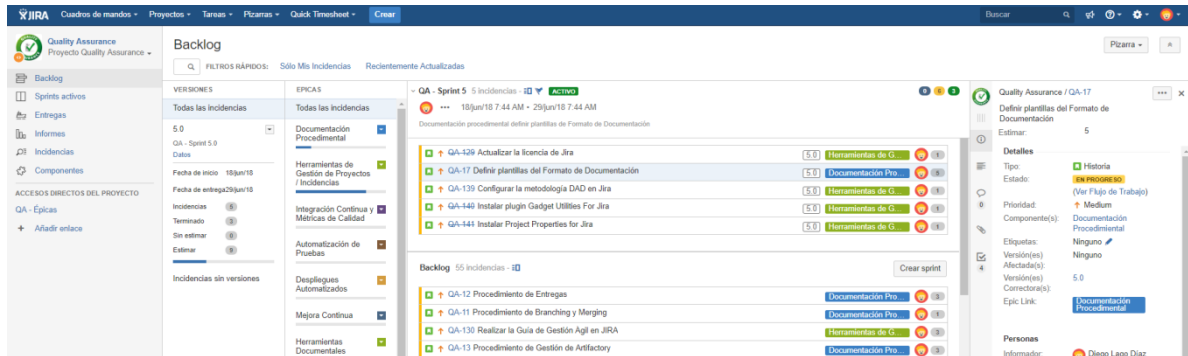


Figura 11. Gestión Proyecto – Jira

- **Construcción:** En esta fase se construirá el sistema. Para ello, se integrará en un Sistema de Integración Continua (Jenkins) que permitirá pasar las pruebas unitarias, integración, de rendimiento así como crear el ejecutable (war) y distribuirlo en Nexus (Herramienta de distribución de artefactos) para mantener un histórico de las versiones entregadas al cliente. Este sistema permitirá detectar errores en fases tempranas aumentando la productividad del equipo y dando como resultado un software fiable y robusto.

Además se integrará con SonarQube. Herramienta que permitirá pasar las métricas de calidad al código fuente desarrollado (líneas de código, documentación del código, duplicidades, porcentaje de cobertura de las pruebas unitarias y de integración) además del número de violaciones detectadas en el código fuente basadas en un perfil de reglas según el estándar de codificación de SUN. Todo ello, mejorará la mantenibilidad, fiabilidad y el rendimiento del software desarrollado. Además, indicará el número de dependencias cíclicas entre paquetes y entre ficheros para aumentar la cohesión y reducir el acoplamiento del código garantizando una arquitectura eficiente y fácilmente mantenible. También se deberá crear un perfil de reglas de Seguridad basadas en el estándar OWASP para prevenir de ataques más importantes (inyección de SQL, Cross-site scripting, comunicaciones inseguras...). Periódicamente se realizarán chequeos manuales donde se revisará si la arquitectura del sistema está bien definida (diferenciando correctamente las capas del modelo-vista-controlador), si se está realizando lógica de negocio en los Data Object, etc...

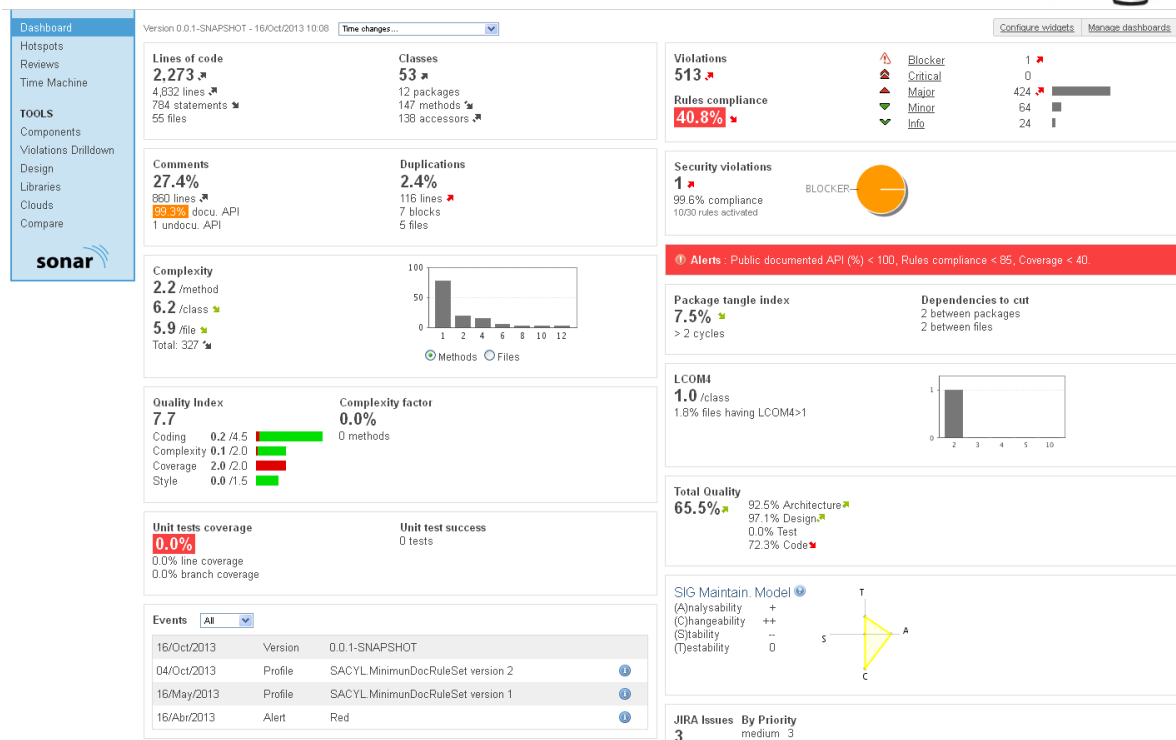


Figura 12. Análisis Código – SonarQube

A mayores, en cada iteración, se automatizará el despliegue en el entorno de preproducción para mostrar al cliente el avance del proyecto (demo) y poder tener un feedback del mismo con la finalidad de reducir los riesgos del proyecto y aumentar la productividad. Con ello, se validará que se han entendido correctamente los requisitos del cliente para lograr la satisfacción final del cliente.

Para gestionar todas las iteraciones (con la lista de tarea) y pruebas se utilizará Jira Software.

Es en esta fase donde se realizarán todas las pruebas indicadas en Plan de Pruebas. A continuación se detallan las herramientas a utilizar en las diferentes pruebas:

Para las pruebas de rendimiento, stress, carga se utilizará la herramienta Open Source JMeter, para las pruebas de integración con Webservice se utilizará la herramienta SoapUI y para las pruebas funcionales automatizadas se utilizará Selenium GRID.

Así mismo se irá subiendo el código fuente al repositorio Git y la documentación asociada al proyecto a Alfresco para llevar un control exhaustivo de las versiones lanzadas.

- **Transición:** Una vez que el sistema está construido, se entregaría la documentación (manuales explotación, manual de instalación, manual de usuario) y se establecería un plan de implementación para pilotar el sistema en producción.



- **Producción:** Una vez instalado el producto en explotación se podrán a disposición del cliente las herramientas de soporte AirCall y Freshdesk, para llevar un control de las llamadas y tickets atendidos.

1.2.2 Seguridad

Se podrá especial interés a la hora de construir el código fuente, utilizando las Buenas Prácticas Seguridad en Java y tratando de mitigar los 10 riesgos de seguridad más importantes en aplicaciones web que se encuentran contemplados en el estándar OWASP (Open Web Application Security Project). Para ello, se configura un perfil específico de reglas de Seguridad en SonarQube (PMD, Findbug) para aumentar la seguridad y fiabilidad. También se utilizará una herramienta de auditoria de seguridad para páginas web (OWASP ZAP), así como otras, como para analizar la accesibilidad de los sitios web TAW.

1.2.2.1 OWASP

El Proyecto de seguridad de aplicaciones web abiertas (**OWASP**) es una comunidad abierta dedicada a permitir que las organizaciones desarrollen, compren y mantengan aplicaciones en las que se pueda confiar. La comunidad incluye corporaciones, organizaciones educativas e individuos de todo el mundo con un enfoque en la creación de artículos, metodologías abiertas, documentación, herramientas y tecnologías disponibles para mejorar la seguridad del software web.

Los 10 riesgos más críticos de seguridad de aplicaciones web bajo el estándar OWASP son:

OWASP Top 10-2003	OWASP Top 10-2004	OWASP Top 10-2007	OWASP Top 10-2010	OWASP Top 10-2013	OWASP Top 10-2017
A1-Entrada no validada	A1-Entrada no validada	A1-Secuencia de comandos en sitios cruzados XSS	A1-Inyección	A1-Inyección	A1 - Inyección
A2-Control de acceso interrumpido	A2-Control de acceso interrumpido	A2-Fallas de inyección	A2-Secuencia de comandos en sitios cruzados XSS	A2-Pérdida de autenticación y gestión de sesiones	A2 - Pérdida de Autenticación
A3-Administración de cuentas y sesión interrumpida	A3-Administración de autenticación y sesión interrumpida	A3-Ejecución de ficheros malintencionados	A3-Pérdida de autenticación y gestión de sesiones	A3-Secuencia de comandos en sitios cruzados XSS	A3 - Exposición de datos sensibles
A4-Fallas de cross site scripting XSS	A4-Fallas de cross site scripting XSS	A4-Referencia insegura y directa a objetos	A4-Referencia directa insegura a objetos	A4-Referencia directa insegura a objetos	A4 - Entidades Externas XML (XXE)
A5-Desbordamiento de bufer	A5-Desbordamiento de bufer	A5-Falsificación de peticiones en sitios cruzados CSRF	A5-Falsificación de peticiones en sitios cruzados CSRF	A5-Configuración de seguridad incorrecta	A5 - Pérdida de Control de Acceso
A6-Fallas de inyección de comandos	A6-Fallas de inyección	A6-Revelación de información y gestión incorrecta de errores	A6-Defectuosa configuración de seguridad	A6-Exposición de datos sensibles	A6 - Configuración de Seguridad Incorrecta
A7-Problemas de manejo de errores	A7-Manejo inadecuado de errores	A7-Pérdida de autenticación y gestión de sesiones	A7-Almacenamiento criptográfico inseguro	A7-Ausencia de control de acceso a las funciones	A7 - Secuencia de Comandos en Sitios Cruzados (XSS)
A8-Uso inseguro de criptografía	A8-Almacenamiento inseguro	A8-Almacenamiento criptográfico inseguro	A8-Falla de restricción de acceso a URL	A8-Falsificación de peticiones en sitios cruzados CSRF	A8 - Deserialización Insegura
A9-Fallas de administración remota(no aplicable)	A9-Negación de servicio	A9-Comunicaciones inseguras	A9-Protección insuficiente en la capa de transporte	A9-Uso de componentes con vulnerabilidades conocidas	A9 - Componentes con vulnerabilidades conocidas
A10-Configuración indebida de servidor web y de aplicación	A10-Administración de configuración insegura	A10-Falla de restricción de acceso a URL	A10-Redirecciones y reenvíos no validados	A10-Redirecciones y reenvíos no validados	A10 - Registro y Monitoreo Insuficientes

Figura 13 OWASP TOP 10

1.2.2.2 CWE / SANS

CWE / SANS es el resultado de la colaboración entre el Instituto SANS, MITRE y muchos de los principales expertos en seguridad de software en los Estados Unidos y Europa. Aprovecha las experiencias en el desarrollo de los vectores de ataque Top 20 de SANS y la Enumeración de debilidad común de MITRE (CWE).



El Top 25 está organizado en tres categorías de alto nivel que contienen múltiples entradas CWE.

1.2.2.2.1 Interacción insegura entre componentes

Estas debilidades están relacionadas con formas inseguras en las que los datos se envían y reciben entre componentes, módulos, programas, procesos, subprocesos o sistemas separados. Para cada debilidad, su clasificación en la lista general se proporciona entre corchetes.

Rank	CWE ID	Name
[1]	CWE-79	Failure to Preserve Web Page Structure ("Cross-Site Scripting")
[2]	CWE-89	Improper Sanitization of Special Elements Used in an SQL Command ("SQL Injection")
[4]	CWE-352	Cross-Site Request Forgery (CSRF)
[8]	CWE-434	Unrestricted Upload of File with Dangerous Type
[9]	CWE-78	Improper Sanitization of Special Elements Used in an OS Command ("OS Command Injection")
[17]	CWE-209	Information Exposure Through an Error Message
[23]	CWE-601	URL Redirection to Un-trusted Site ("Open Redirect")
[25]	CWE-362	Race Condition

Figura 14 *CWE – Insecure Interaction Between Components*

1.2.2.2.2 Gestión de riesgos en recursos

Las debilidades en esta categoría están relacionadas con las formas en que el software no administra adecuadamente la creación, uso, transferencia o destrucción de recursos importantes del sistema.

Rank	CWE ID	Name
[3]	CWE-120	Buffer Copy Without Checking Size of Input ("Classic Buffer Overflow")
[7]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ("Path Traversal")
[12]	CWE-805	Buffer Access with Incorrect Length Value
[13]	CWE-754	Improper Check for Unusual or Exceptional Conditions
[14]	CWE-98	Improper Control of Filename for Include/Require Statement in PHP Program ("PHP File Inclusion")
[15]	CWE-129	Improper Validation of Array Index
[16]	CWE-190	Integer Overflow or Wraparound
[18]	CWE-131	Incorrect Calculation of Buffer Size
[20]	CWE-494	Download of Code Without Integrity Check
[22]	CWE-770	Allocation of Resources Without Limits or Throttling

Figura 15 *CWE – Risky Resource Management*

1.2.2.2.3 Defensas pobres

Las debilidades en esta categoría están relacionadas con las técnicas defensivas que a menudo se usan mal, se abusa o simplemente se ignoran.

Rank	CWE ID	Name
[5]	CWE-285	Improper Access Control (Authorization)
[6]	CWE-807	Reliance on Inputs in a Security Decision
[10]	CWE-311	Missing Encryption of Sensitive Data
[11]	CWE-798	Use of Hard-Coded Credentials
[19]	CWE-306	Missing Authentication for Critical Function
[21]	CWE-732	Incorrect Permission Assignment for Critical Resource
[24]	CWE-327	Use of a Broken or Risky Cryptographic Algorithm

Figura 16 *CEW - Defensas Pobres*




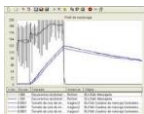


1.2.3 Pruebas

Se debe crear un Plan de Pruebas completo que permita asegurar la bondad del sistema construido, detectando y, en su caso, corrigiendo malos funcionamientos con la mayor antelación posible.

Este plan de pruebas deberá incluir las diferentes fases de las que consta (pruebas unitarias, pruebas de sistema, pruebas de rendimiento, pruebas de integración, etc).

El **Plan de Pruebas** que se ejecutará contará con la siguiente tipología de test:

	Tipo de test	Descripción
	Pruebas Unitarias	<p>Las pruebas unitarias tienen el mayor efecto en la calidad del código cuando son parte integral del flujo de trabajo de desarrollo de software. Las pruebas unitarias comprueban el comportamiento del código en respuesta a casos estándar, límite e incorrectos de datos de entrada, así como cualquier suposición explícita o implícita creada por el código.</p> <p><i>Herramientas:</i> JUnit, NUnit, TestNG)</p>
	Pruebas Funcionales	<p>Comprobación de que los desarrollos efectuados cumplen las especificaciones funcionales y los requisitos de los usuarios. Ayudan a detectar los posibles defectos derivados de errores en la fase de Construcción.</p> <p><i>Herramientas:</i> Selenium, Apium</p>
	Pruebas de Rendimiento	<p>Verificación de la velocidad de respuesta del sistema ante las peticiones del usuario en las diferentes áreas funcionales. Ayuda a determinar cuellos de botella y a poner solución antes de que se reciban quejas por parte de los usuarios.</p> <p><i>Herramientas:</i> Jmeter, SoapUI</p>
	Pruebas de Stress	<p>Pruebas que se efectúan para disponer de una estimación de cuánto se puede cargar el sistema antes de que sea inutilizable. Facilitan el correcto dimensionamiento del sistema.</p> <p><i>Herramientas:</i> Jmeter, SoapUI</p>



Pruebas de Carga

Una prueba de carga se realiza generalmente para observar el comportamiento de una aplicación bajo una cantidad de peticiones esperada. Esta carga puede ser el número esperado de usuarios concurrentes utilizando la aplicación y que realizan un número específico de transacciones durante el tiempo que dura la carga. Dan información de los tiempos de respuesta según la carga esperada.

Herramientas: **Jmeter**, SoapUI



Pruebas de Escalabilidad

El propósito de las pruebas de escalabilidad es identificar cargas de trabajo mayores y mitigar los cuellos de botella que pueden impedir la escalabilidad de la aplicación.

Herramientas: **Jmeter**



Pruebas de Seguridad

Se realizarán para detectar vulnerabilidades de seguridad en los sistemas y accesos no autorizados a la información.

Herramientas de Caja Blanca: **Sonarqube**

Herramientas de Caja Negra: entorno **Kali**



Pruebas de Usabilidad

Son pruebas de verificación de la usabilidad orientadas específicamente a los usuarios finales para verificar la facilidad de uso del nuevo sistema.

Manuales



Pruebas de Integración

Son pruebas encaminadas a validar que el nuevo sistema se integra perfectamente con el resto de los sistemas.

Herramientas: **Jmeter**, Postman, SoapUI

Tabla 1. Tipología de Pruebas