Atam

Integración / Entrega / Despliegue / Inspección Continua

Sesiones 5

18/05/2022

CEIIST -spa- v1.0

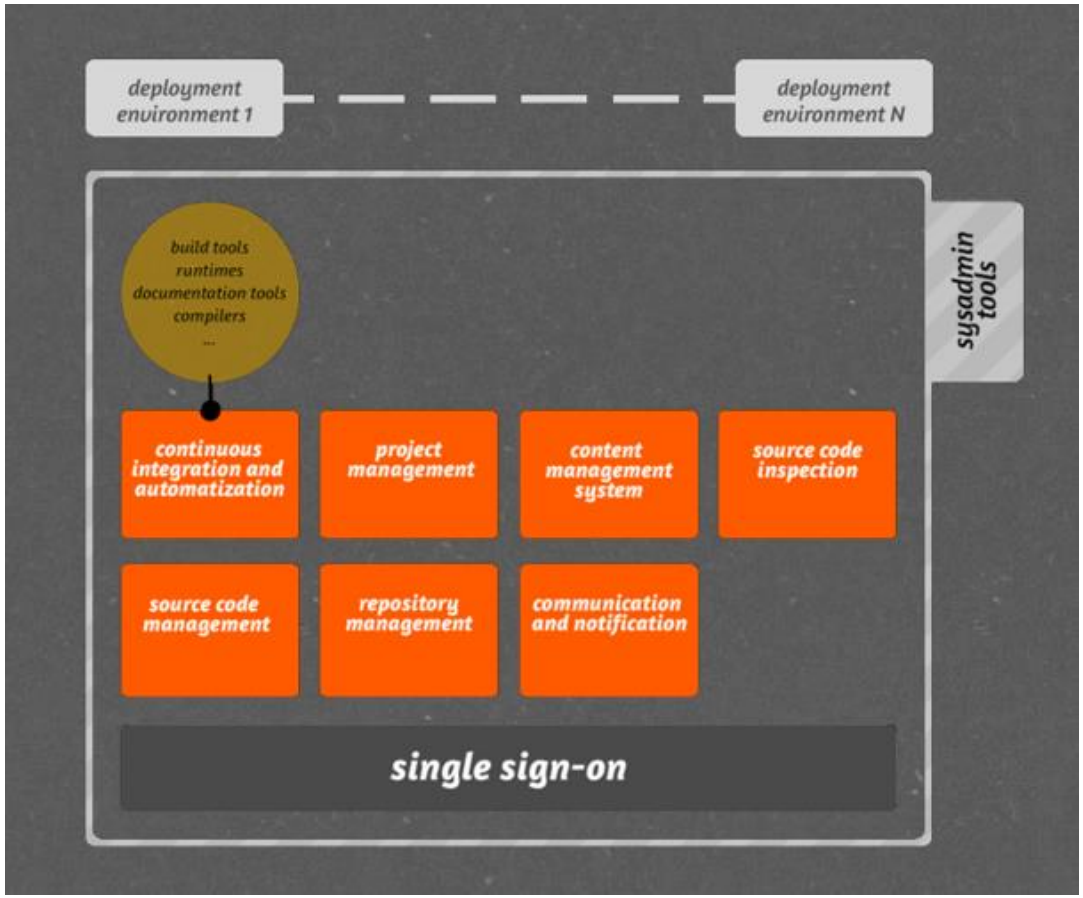
panel.es
Panel Sistemas Informáticos, S.L.
Consultoría, servicios y soluciones TI.

Agenda. Sesión 5

- **Ecosistema Software**
  - Modelo
  - Componentes
  - Ejemplo
- **Contexto Ciclo Desarrollo del Software**
- **Definiciones**
  - Integración
  - Entrega
  - Despliegue
- **Consideraciones Generales para Implantación**
- **Servidor de Integración Continua**
  - Jenkins
- **Inspección Continua. Análisis de Código.**
  - Sonarqube

Definición

- Espacio o plataforma de trabajo en el que conviven una serie de herramientas que acompañadas de unas buenas prácticas permiten a un equipo de desarrollo modelar una metodología de trabajo.
- Esta elección de herramientas está basada en el siguiente modelo conceptual:



Ecosistema Software. Componentes

Gestión de sesión unificado:

- Procedimiento de autenticación y autorización que habilita a un usuario determinado para acceder a varios sistemas con una sola instancia de identificación.

Gestión del Código fuente:

- Contendrá la gestión de los diversos cambios que se realizan sobre los elementos del proyecto proporcionando una visión unívoca del estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación.
- Permite resolver la práctica de seguimiento y administración de los cambios en el código fuente.

Gestión de Artefactos:

- Biblioteca de artefactos / librerías gestionados en nuestros procesos de fabricación software.
- Beneficios:
  - ✓ Centraliza el acceso a las librerías de desarrollo. Actúa como “proxy” de librerías (lo que supone optimización del ancho de banda y ahorro de tiempo en las descargas).
  - ✓ Favorece la distribución y búsqueda de componentes software.
  - ✓ Administración de repositorios privados.

Integración Continua y Herramientas de construcción:

- Herramienta que facilita la automatización de tareas que comprenden todo el ciclo de vida de un proyecto software. Debe dar información de forma explícita del estado del software en todo momento.

- Tiene que cumplir las siguientes restricciones:
  - ✓ Integración con herramienta de gestión unificado (autenticación y autorización).
  - ✓ Integración con control de versiones centralizados/distribuidos.
  - ✓ Integración con Repositorios de artefactos.
  - ✓ Soporte a las distintas herramientas de construcción/despliegue que soporten las distintas pilas tecnológicas de las aplicaciones de ST que faciliten un correcto modelado del proyecto.
  - ✓ Delegación de tareas en múltiples sistemas operativos.

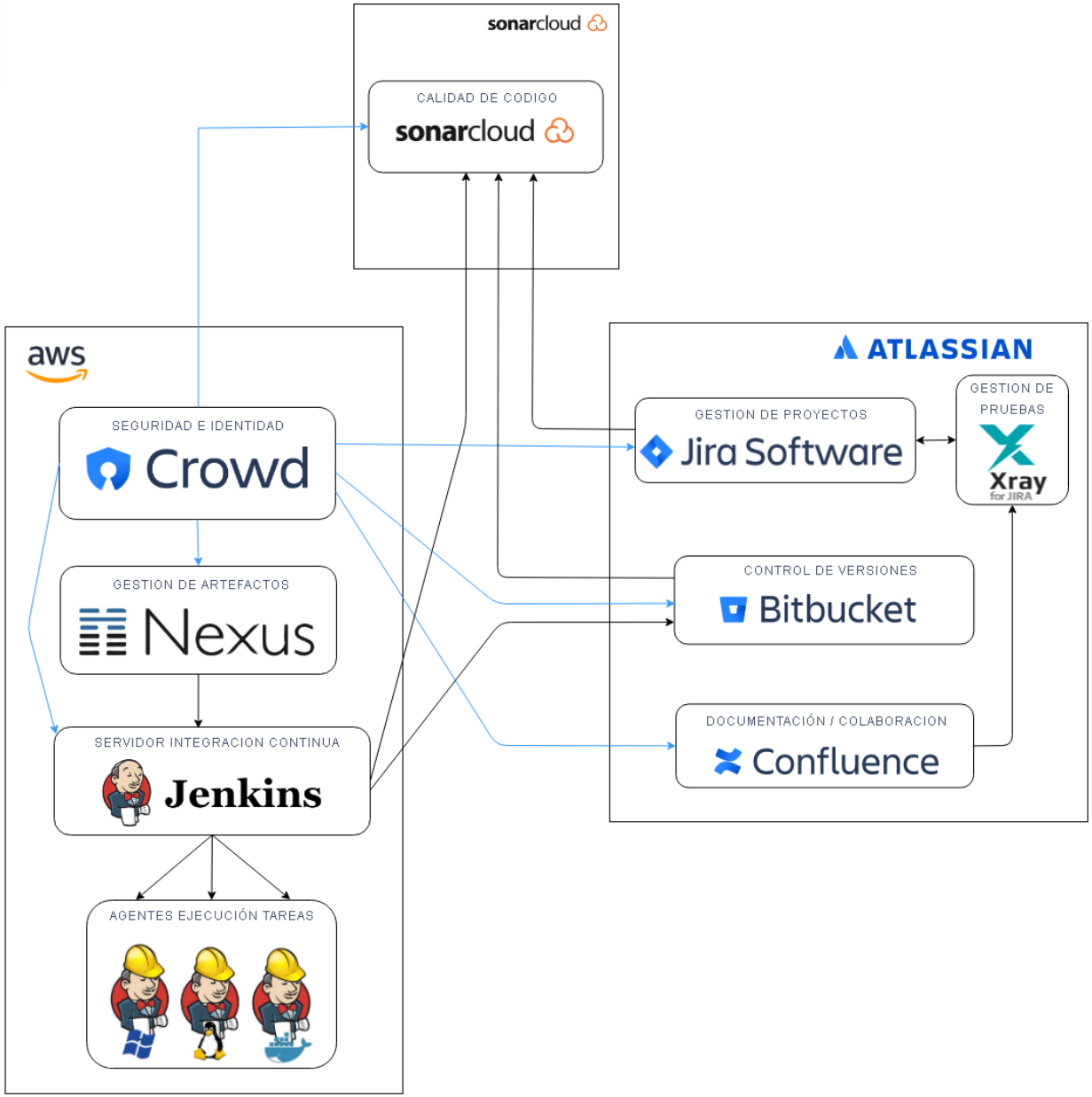
- Facilitar las principales fases de construcción del ciclo de vida
  - ✓ Compilación y empaquetado.
  - ✓ Lanzamiento de pruebas.
  - ✓ Análisis de código.
  - ✓ Despliegue

Análisis de la Calidad del Código fuente:

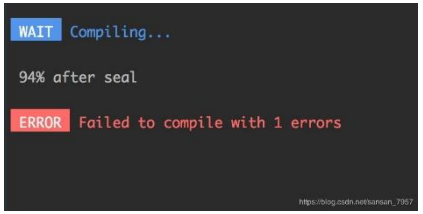
- Herramientas para evaluar forma objetiva la calidad del software generado conforme a los principales estándares del mercado.
- Nos debe permitir al menos la siguiente información:
  - ✓ Volumetría: Número de líneas / instrucciones/ clases / métodos, LOC, etc.
  - ✓ Código duplicado: Porcentaje, número de líneas / bloques / ficheros duplicados.
  - ✓ Complejidad ciclomática a nivel de método / clase / fichero.
  - ✓ Adherencia a estándares de codificación.
  - ✓ Detección de fallos potenciales, incluidos los de seguridad.
  - ✓ Información sobre Pruebas y Comentarios del proyecto.

- Restricciones:
  - ✓ Soporte para los lenguajes que den soporte a todos los lenguajes de la pila tecnológica Análisis de código.
  - ✓ Extensible a partir de complementos (plugins) tanto comerciales como de código abierto (opensource).

Ecosistema Software. Un ejemplo



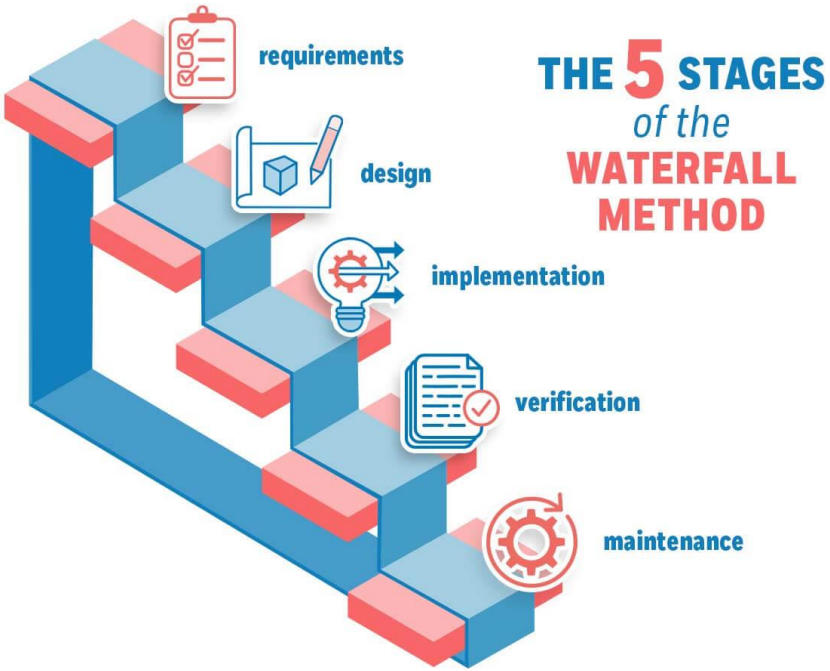
Dolores en el ciclo de vida del SW

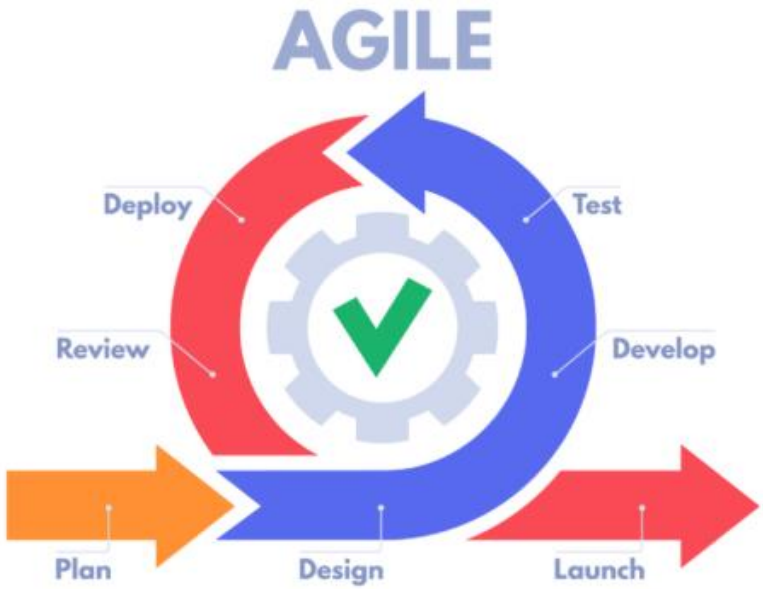


- En los proyectos se trabaja en equipo y suele haber problemas a la hora de integrar cambios con los demás.
- El entorno de integración suele estar inestable y hace difíciles las demostraciones (esfuerzo/coste).
- En mi local funciona, pero en integración no.
- La integración siempre la realiza Héctor.
- El proceso de despliegue es manual, tedioso y difícil.
- Nunca tenemos tiempo para solucionar los problemas.
- Resistencia al cambio.
- Los errores se detectan tarde y hay poca percepción de calidad.

Ciclo de vida del desarrollo de software

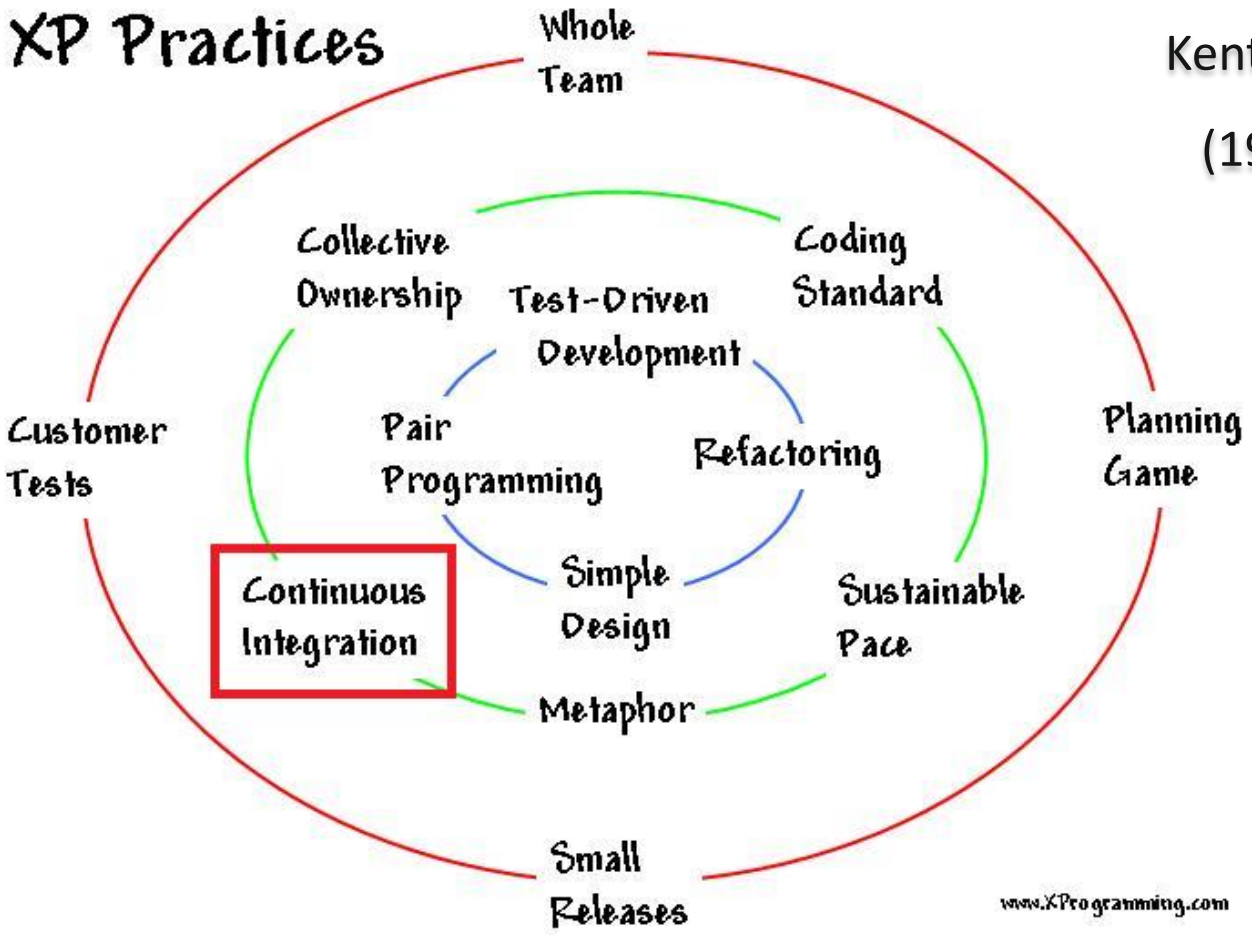
Contexto





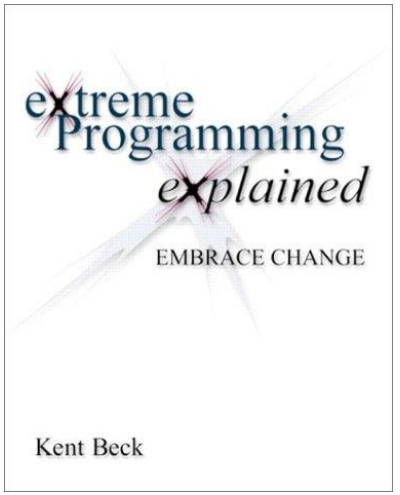
Definición de Integración Continua

Orígenes



Kent Beck

(1999)



Definición de Integración Continua

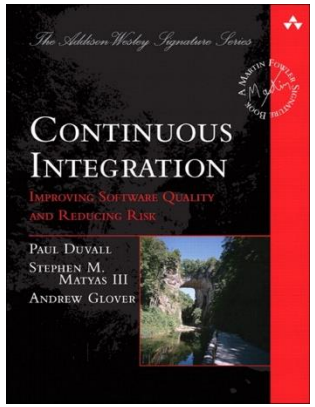
Desarrollo de la Práctica

- "La integración continua es una práctica de desarrollo de software en la que los miembros de un equipo integran su trabajo con frecuencia, normalmente cada persona integra al menos a diario, lo que da lugar a múltiples integraciones al día."

- "Cada integración se verifica mediante una compilación automatizada (**que incluye pruebas**) para detectar los errores de integración lo antes posible."

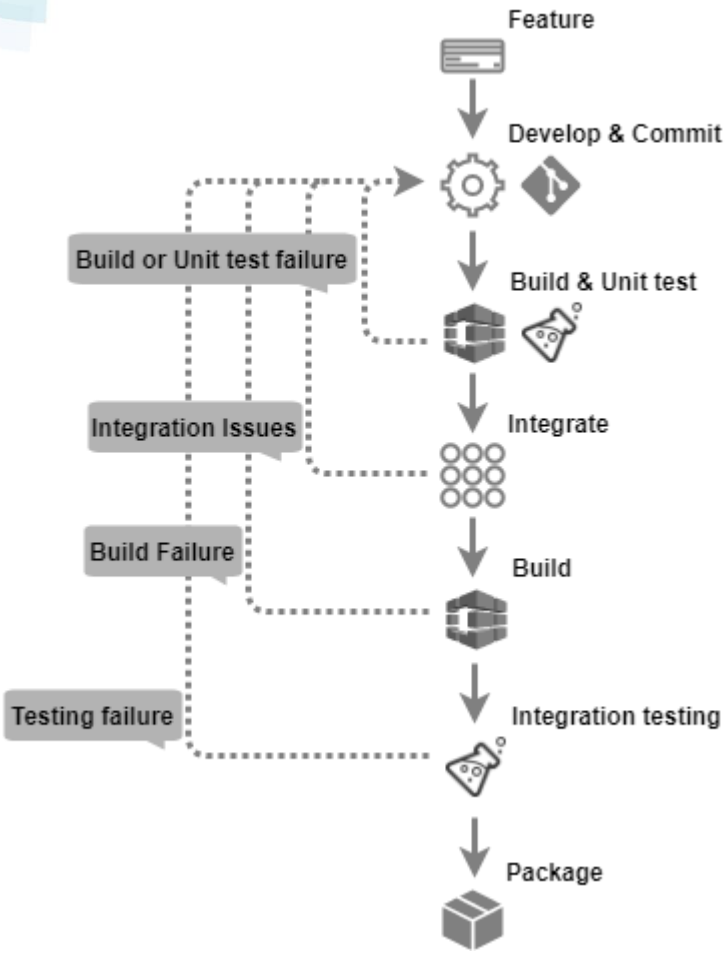
Martin Fowler

(2006)



Definición de Integración Continua

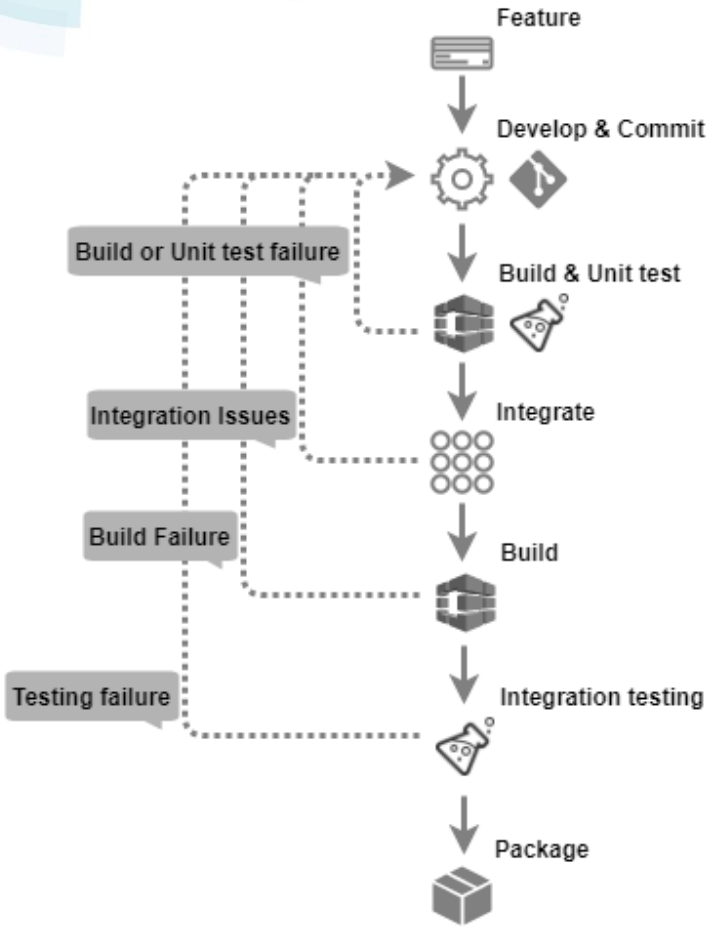
Creación de una funcionalidad con integración continua



- Afrontamos una funcionalidad que se puede acometar en un día.
- Descargo en mi entorno de desarrollo local, una copia actualizada de la rama de integración del control de versiones que utilicemos.
- Realizo todo lo necesario para completar mi tarea, incluyendo pruebas automáticas que hagan que mi código se autopruebe.
- Una vez que he terminado, realizo una compilación automatizada en mi entorno de desarrollo. Esto toma el código fuente en mi copia de trabajo, lo compila y enlaza en un ejecutable, y ejecuta las pruebas automatizadas. Sólo si todo se compila y se prueba sin errores se considera que la compilación general es válida.

Definición de Integración Continua

Creación de una funcionalidad con integración continua



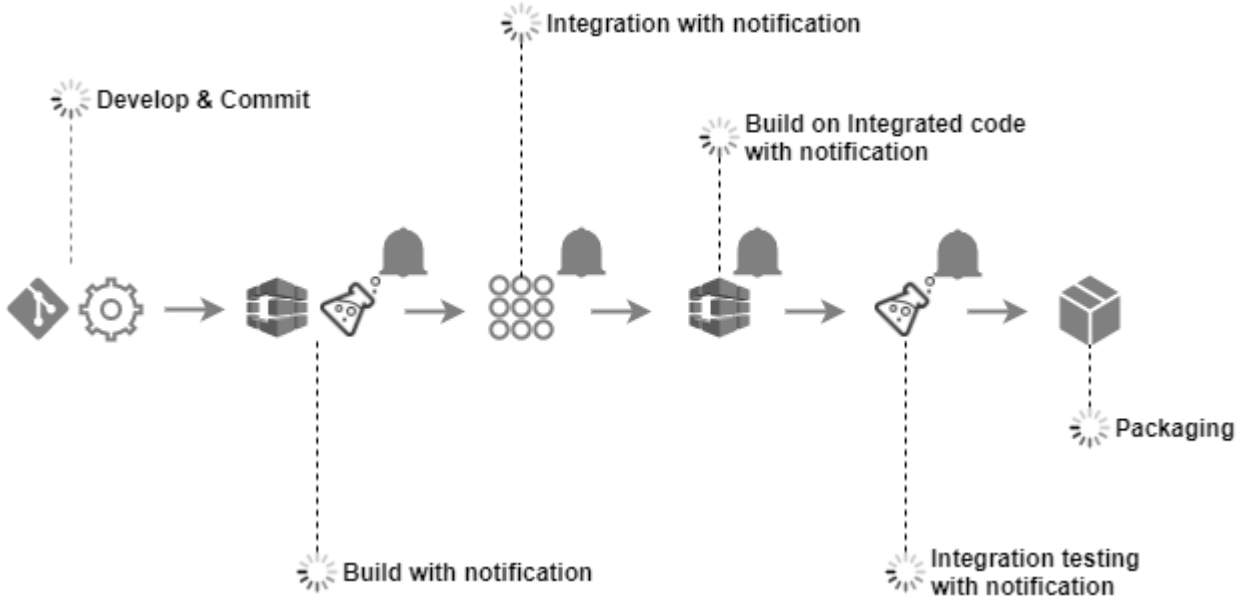
- Sólo cuando el proceso de contrucción es válido, puedo entonces pensar en subir mis cambios en el repositorio. Como otros compañeros puede que hayan contribuido código a la rama de integración, actualizo mi entorno de desarrollo en local con una copia de trabajo con sus cambios (merge) y reconstruyo (build&test).
- Si sus cambios chocan con los míos, se manifestará como un fallo en la compilación o en las pruebas. En este caso es mi responsabilidad arreglarlo y repetirlo hasta que pueda construir una copia de trabajo que esté correctamente sincronizada con rama de integración. Finalmente subo mi código a dicha rama.
- Se vuelve a repetir el proceso de construcción y ejecución de pruebas en la máquina de integración. Sólo cuando este proceso se realice con éxito habrá terminado mi tarea.
- Es responsabilidad del equipo mantener está línea sin fallos, que si existen deben ser rápidamente corregidos.

Definición de Integración Continua

Creación de una funcionalidad con integración continua

- El resultado de hacer esto es que hay una pieza de software estable que funciona correctamente (artefacto) y contiene pocos errores.

- Todo el mundo desarrolla a partir de esa base estable compartida y nunca se aleja tanto de esa base como para tardar en volver a integrarse en ella. Se invierte menos tiempo en buscar errores porque aparecen rápidamente.

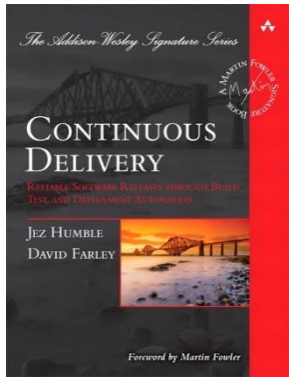


Prácticas de Integración Continua

- Mantener un único repositorio de código fuente.
- Automatizar el proceso de construcción.
- Haga que su proceso de construcción se autopruebe.
- Todo el equipo se compromete con la línea principal cada día.
- Cada nueva funcionalidad se debería construir sobre la línea principal en una máquina de integración.
- Es una prioridad para el equipo arreglar inmediatamente las construcciones rotas.
- Mantener la rapidez en el proceso de construcción.
- Prueba en un clon del entorno de producción.
- Debe ser sencillo y accesible para el equipo obtener un artefacto del software válido en todo momento.
- Todo el mundo puede ver lo que ocurre.
- Automatiza el despliegue.

Entrega / Despliegue Continuo

- “La entrega continua es una disciplina de desarrollo de software en la que se construye software de tal manera que el software puede ser liberado a la producción en cualquier momento”.



Usted está haciendo la entrega continua cuando:

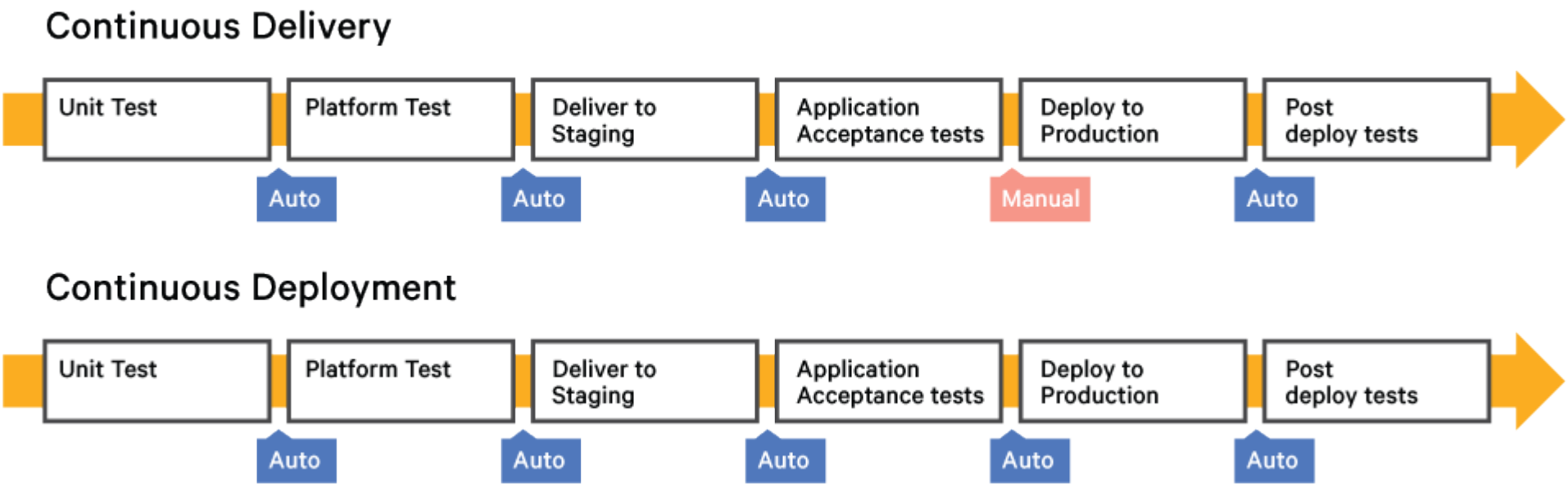
- Su software es desplegable durante todo su ciclo de vida.
- Su equipo da prioridad a mantener el software desplegable sobre el trabajo en nuevas características.
- Cualquiera puede obtener información rápida y automatizada sobre el estado de producción de sus sistemas cada vez que alguien realiza un cambio en ellos.
- Puede realizar despliegues de cualquier versión del software en cualquier entorno bajo demanda.

Martin Fowler

(2013)

Entrega / Despliegue Continuo

- “La entrega continua se confunde a veces con el despliegue continuo”.
- “El Despliegue Continuo significa que cada cambio pasa por el pipeline y se pone automáticamente en producción, resultando en muchos despliegues de producción cada día”.
- “La Entrega Continua sólo significa que usted es capaz de hacer despliegues frecuentes, pero puede elegir no hacerlo, por lo general debido a las empresas que prefieren un ritmo más lento de despliegue”.
- “Para hacer un Despliegue Continuo hay que hacer una Entrega Continua.”

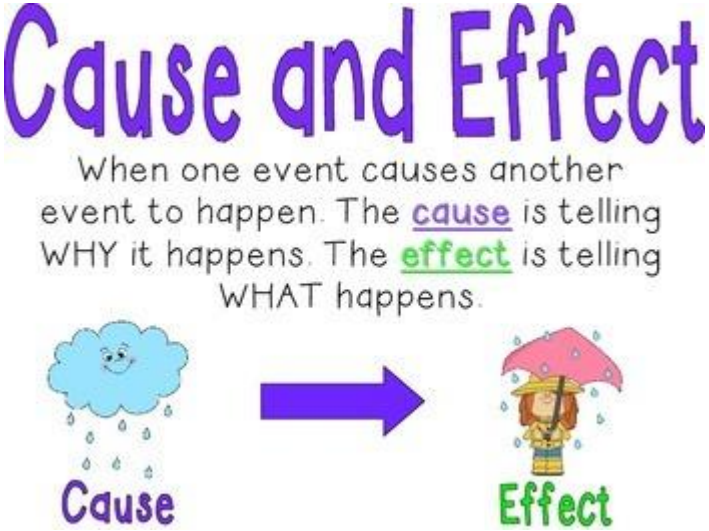


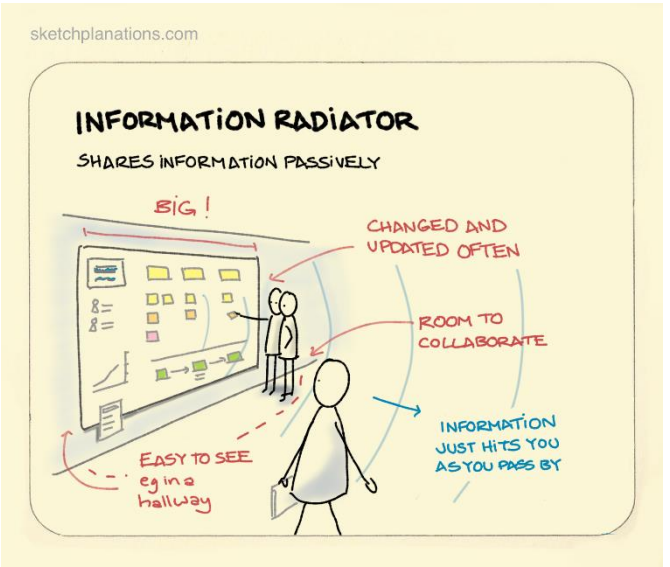
Consejos para la Implantación CI /CD

Manuel Recena

@recena

- Principio de Causalidad + Radiador de Información





Consejos para la Implantación CI /CD

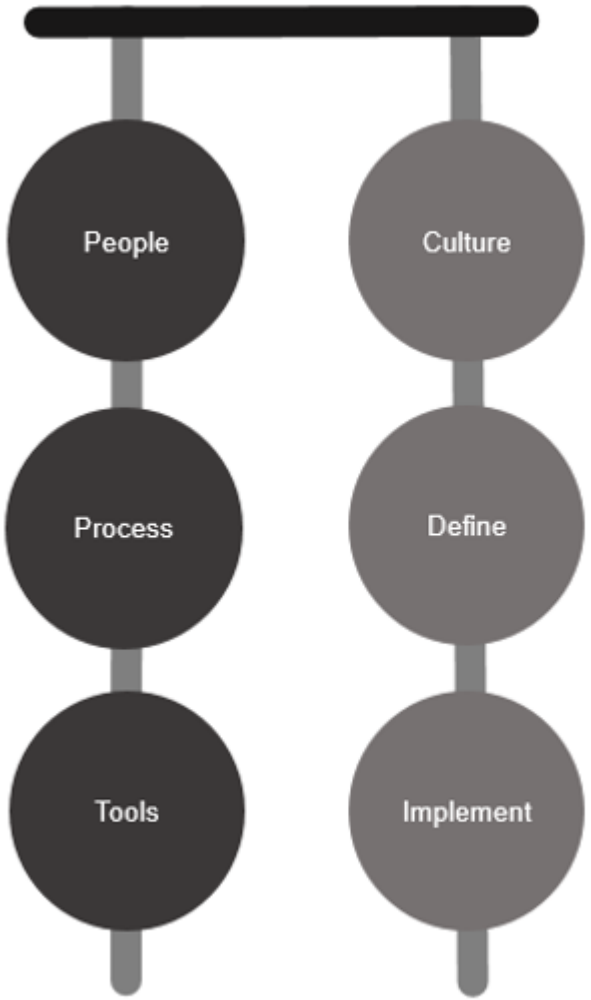


Figura 1: Personas - Procesos - Herramientas

Cuando empiezas cambiando la cultura organizativa pensando en las personas, la inercia te lleva a optimizar tus procesos y a implementar/alinear las herramientas. De esta forma, los riesgos económicos son mínimos.

Cultura

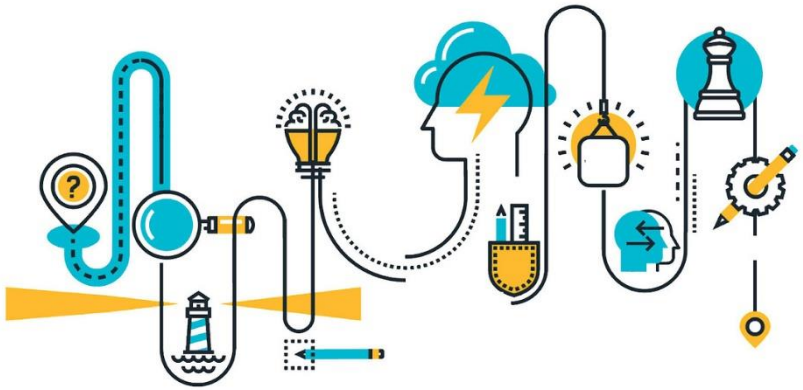
- Si estamos tratando con equipos u organizaciones con procesos de construcción ya en marcha, recomendamos minimizar la resistencia al cambio utilizando el -enfoque de mejora continua- basada en principios Lean para la implantación de esta práctica como :
  - ✓ Cambios progresivos y evolutivos.
  - ✓ Establecer las reglas de forma explícita.
  - ✓ Potenciar el equipo: el equipo de desarrollo acepta salir de su zona de confort, adoptando y tomando como propios los cambios en su forma de trabajo.
  - ✓ Dotar al equipo de holguras que permitan afrontar los cambios.
  - ✓ Ampliar el aprendizaje, con retroalimentación del equipo y del cliente.

- Como punto de partida se debe identificar el método actual de trabajo, los roles y personas implicadas. A continuación, se debe analizar la situación al objeto de dotar al equipo de principios que sirvan de sustento para afrontar las prácticas (metodología / proceso) como:
  - ✓ Reducción de riesgos en el proceso de construcción.
  - ✓ Feedback temprano.
  - ✓ Calidad desde el inicio.
  - ✓ Propiedad colectiva del código.
  - ✓ Importancia de la metodología / procesos.

Consejos para la Implantación CI /CD

- De esta forma, conseguiremos que las buenas prácticas propias de la Integración Continua sean entendidas y aceptadas por el equipo , llegando a convertirse en un hábito.

Procesos



- Una vez que tenemos establecida la cultura del equipo, podemos definir los procesos que permitirán minimizar el esfuerzo y maximizar el rendimiento obtenido. A continuación se indican algunas de las buenas prácticas que nos guiarán durante esta fase:
  - ✓ Desde un punto de vista de Organización.
  - ✓ Desde un punto de vista de Tareas.
  - ✓ Desde un punto de vista de Radiadores de información.

Consejos para la Implantación CI /CD

Organización

- Respetar una estructura para facilite navegar entre proyectos.

- Establecer las líneas de automatización a aplicar. En general, las líneas de automatización consideradas suelen ser:
  - ✓ Compilación y empaquetado: actividades hasta que genero cada artefacto.
  - ✓ Distribución: son las tareas que suben los artefactos a la biblioteca.
  - ✓ Generación de documentación.
  - ✓ Pruebas: desde pruebas unitarias a pruebas de rendimiento.
  - ✓ Inspección de código.
  - ✓ Despliegue (deployment): instalar el resultado a algún entorno (pre, test, nopro), no solo producción.

Consejos para la Implantación CI /CD

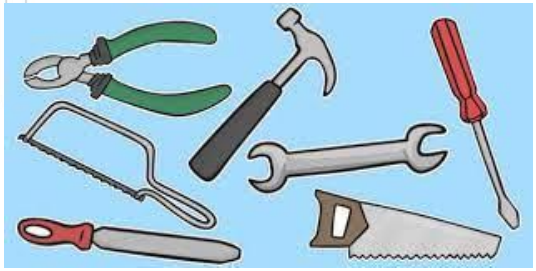
Tareas

- ❑ Crear tareas por cada “causa” que se quiera controlar.
- ❑ Definir convenciones en el nombrado.
- ❑ Definir políticas de ejecución:
  - ✓ Esta política condiciona el coste que va a suponer el uso de las herramientas.
  - ✓ Por ejemplo: Inspección de código por la noche o empaquetar con cada commit.
- ❑ Utilizar listas de correo para las notificaciones.
- ❑ Aplicar definición temprana.

Radiares de Información

- ❑ Para dar valor, hay que medir.
- ❑ Si no se cuida, se genera información dispersa y desorganizada.
- ❑ Definir los canales de comunicación.

Consejos para la Implantación CI /CD



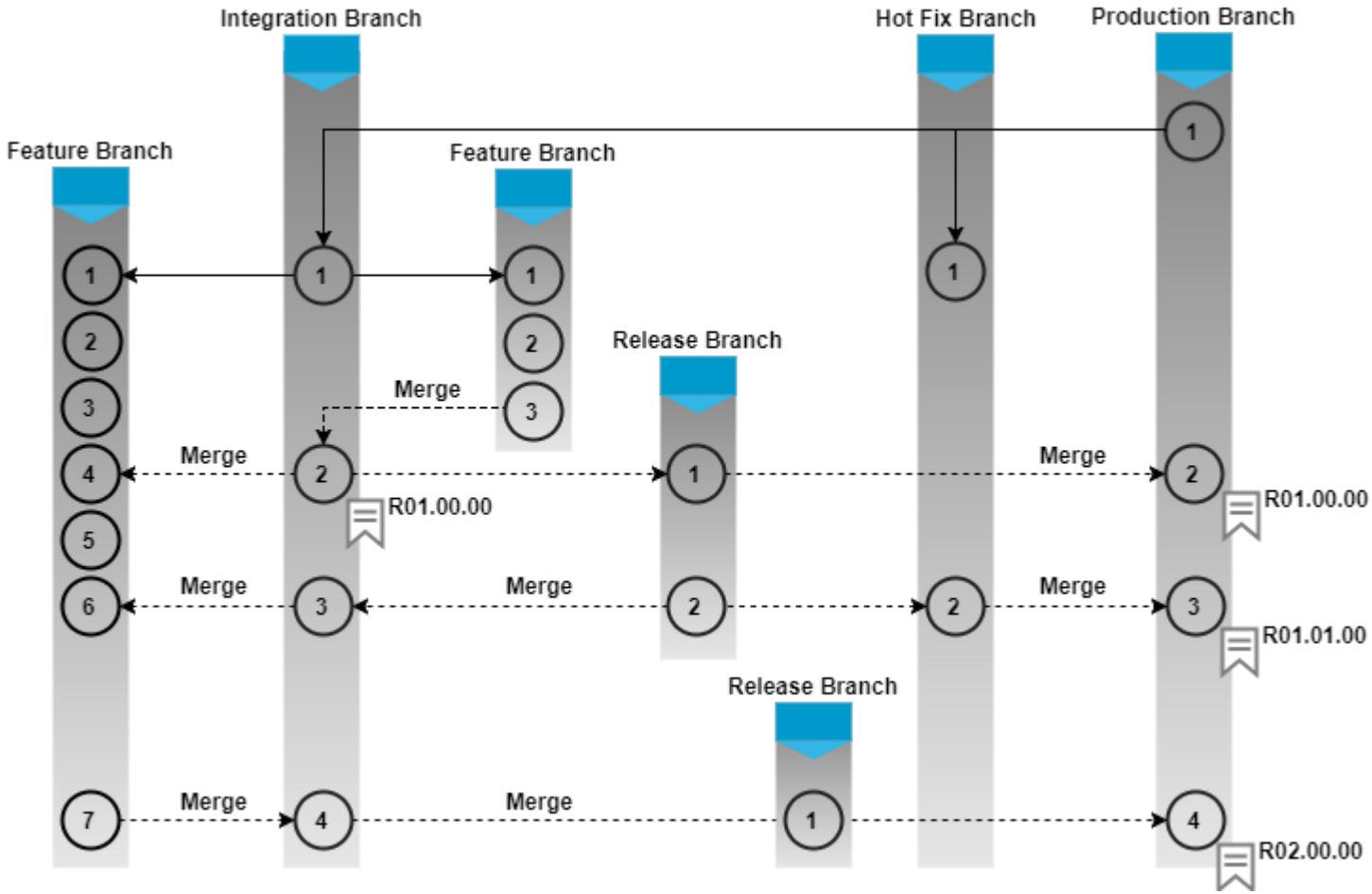
Herramientas

- Por último, desde un punto de vista puramente técnico tendremos que las alternativas tecnológicas van creciendo constantemente. Diversos fabricantes software tienen disponibles "suites" comerciales e incluso es necesario elegir entre escenarios “on premise” (en local) o en la nube ("cloud").

- Algunos aspectos a considerar son:
  - ✓ ¿Cómo se integran estos productos con las herramientas que tenemos?. El coste no está en la instalación y configuración, sino en el mantenimiento e integraciones.
  - ✓ ¿Ofrece soporte para múltiples pilas ("stacks") tecnológicas? No hay que olvidar que se trata de un sistema de propósito general.
  - ✓ ¿Ofrece soporte para distintas herramientas de construcción?.
  - ✓ ¿Cómo afecta si aumenta el número de usuarios y proyectos?.
  - ✓ ¿Qué garantías de rendimiento o escalabilidad tenemos?
  - ✓ ¿Tienes infraestructura y gente con experiencia administrando (sysadmin) este tipo de herramientas?.
  - ✓ ¿Podemos unificar el control de acceso?.
  - ✓ ¿Tienes restricciones contractuales impuestas?

Gestión de la Configuración

- La madurez en el uso del control de versiones es un elemento clave. Ten definida, documentada y automatizada todos los pasos de la **estrategia de ramas** seleccionada en el proyecto.



Gestión de la Configuración

- El modelado del proyecto es clave, usa las herramientas de construcción más adecuadas para la pila tecnológica que usa tu proyecto.

- Recuerda que la herramienta de construcción es el lenguaje que entiende tu herramienta de integració continua.

- Externaliza los elementos de configuración a una fuente externa, siempre que puedas.

Análisis de la Calidad del Código fuente

- Herramientas para evaluar de forma objetiva la calidad del software generado conforme a los principales estándares del mercado.
- SonarQube es una plataforma de código abierto para la gestión de la calidad del software, dedicada a analizar y medir de forma continua la calidad técnica, desde el portfolio de un proyecto hasta el método.
- Es un proyecto extensible a través de plugins tanto opensource como comerciales, que soporta más de 25 lenguajes de programación que cubren las principales pilas tecnológicas de las organizaciones.

Ejes de Calidad o los 7 pecados capitales del desarrollo del Software

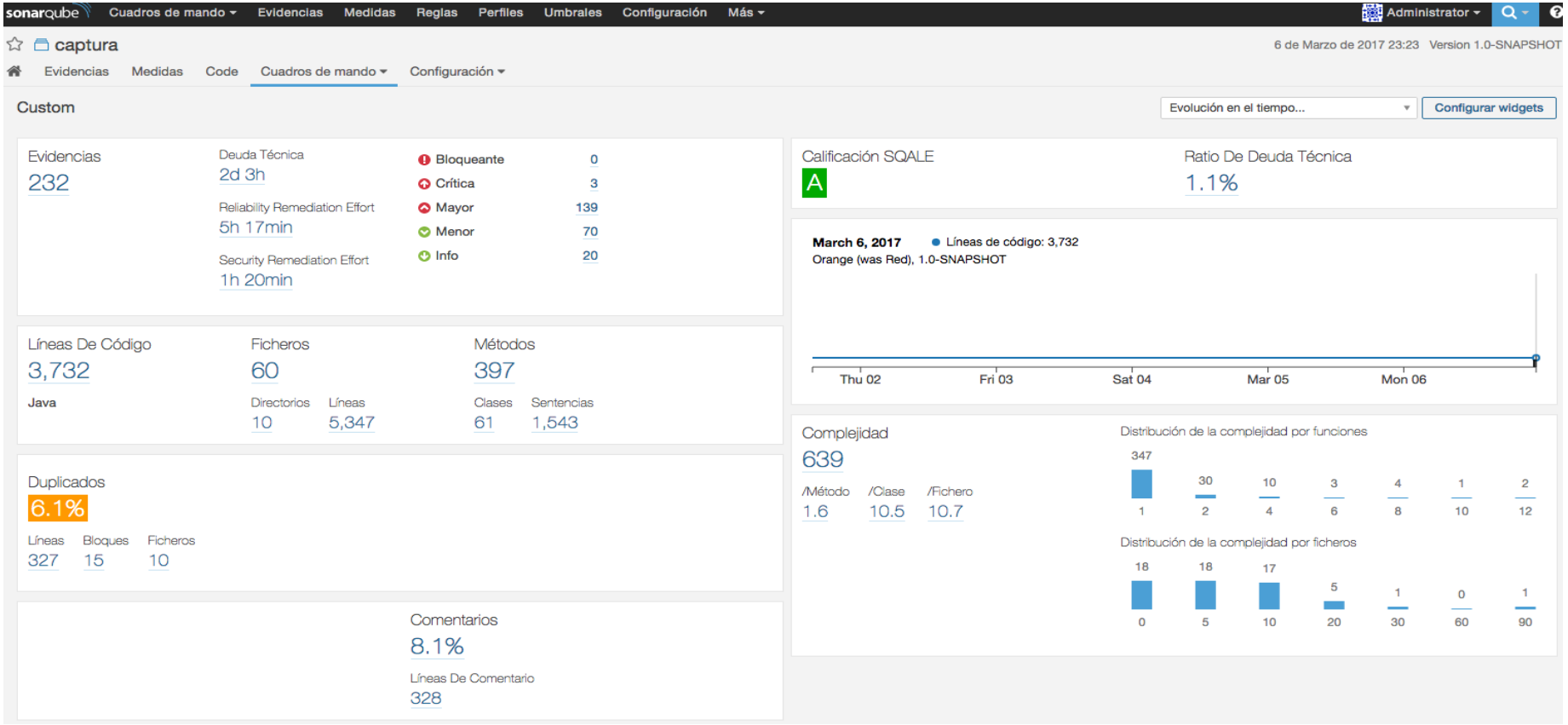
- Al igual que existen los siete pecados capitales espirituales, que pueden "amargarte" la vida si no los cumples, en el mundo del desarrollo de software también existen siete pecados capitales que pueden hacerte imposible la evolución y el mantenimiento del software.

- Cada uno de los pecados capitales está representado en el cuadro de mandos por defecto de SonarQube, y la mayoría pueden seguirse a través de las evidencias, añadiendo las reglas del repositorio de reglas a los perfiles de calidad utilizados en los análisis.

- Estos son los pecados capitales o los 7 ejes de calidad:
  - ✓ Defectos y defectos potenciales -> Dejar fallos potenciales sin analizar.
  - ✓ Incumplimiento de estándares -> No respetar los estándares de codificación y las mejores prácticas establecidas (“Coding standards”).
  - ✓ Código Duplicado -> Tener líneas de código duplicado.
  - ✓ Falta de pruebas unitarias -> Tener una cobertura baja (o nula) de pruebas unitarias, especialmente en partes complejas del programa.
  - ✓ Mala distribución de la complejidad -> Tener componentes complejos y/o una mala distribución de la complejidad entre los componentes.
  - ✓ Código spaghetti -> Tener el temido diseño spaghetti, con multitud de dependencias cíclicas.
  - ✓ Insuficientes o demasiados comentarios -> Falta de comentarios en el código fuente, especialmente en las APIs públicas.

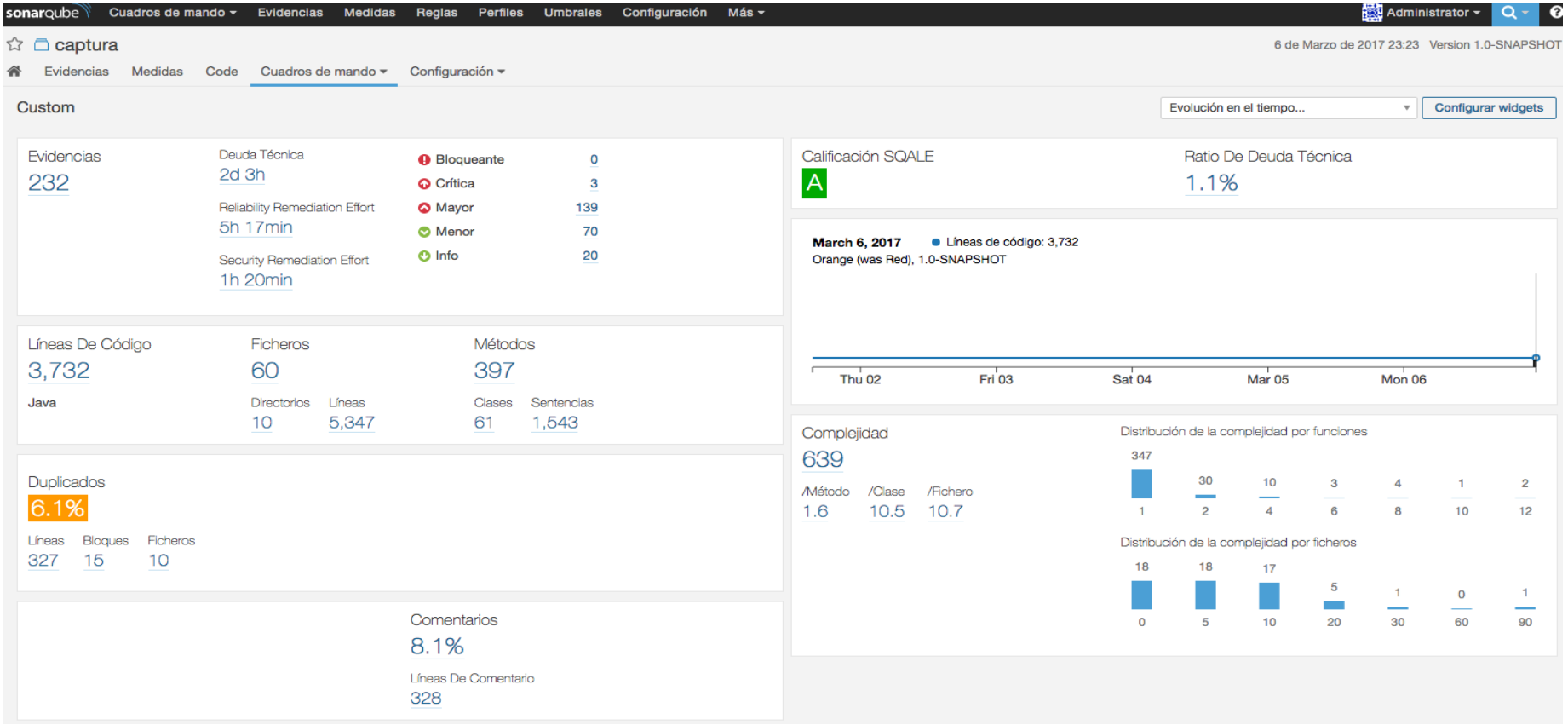
Ejes de Calidad o los 7 pecados capitales del desarrollo del Software

- Cada vez que se comete un pecado capital se incrementa la deuda técnica, que es el esfuerzo requerido para remediar los pecados. Asumiendo que has añadido las reglas en el perfil de calidad, esa deuda técnica puede gestionarse y seguirse a través del mecanismo de evidencias de SonarQube.



Ejes de Calidad o los 7 pecados capitales del desarrollo del Software

- Cada vez que se comete un pecado capital se incrementa la deuda técnica, que es el esfuerzo requerido para remediar los pecados. Asumiendo que has añadido las reglas en el perfil de calidad, esa deuda técnica puede gestionarse y seguirse a través del mecanismo de evidencias de SonarQube.



Perfiles de Calidad

- Describimos como Sonarqube organiza las reglas entorno a repositorios de reglas y como se asocian a los Perfiles de Calidad de cada pila tecnológica.

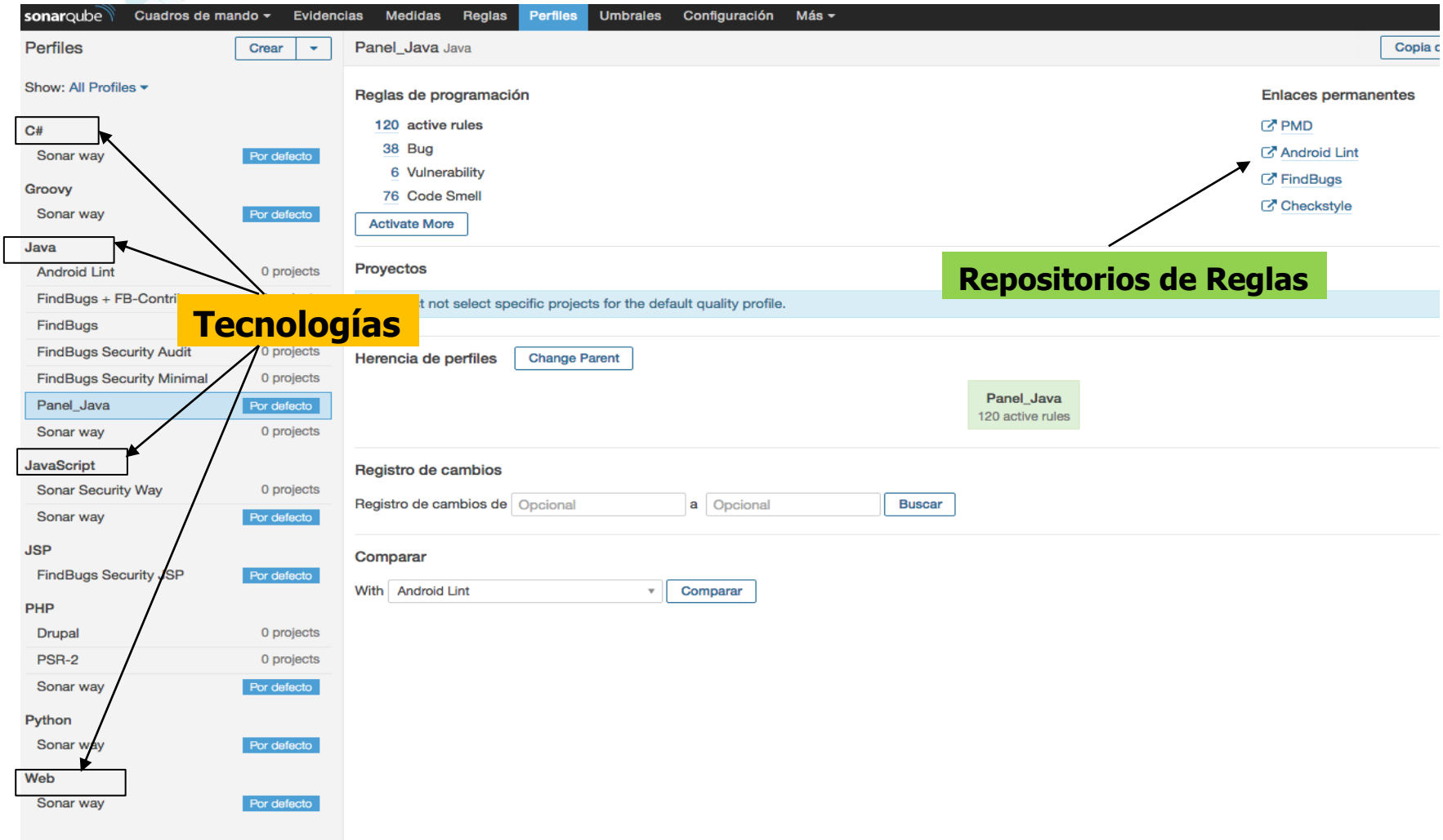
- En SonarQube, los plugins proporcionan reglas que se ejecutan sobre el código fuente y que generan evidencias - que se utilizan para calcular la deuda técnica. La página de reglas es el punto de entrada desde el que se pueden descubrir todas las reglas o se pueden crear nuevas basándose en plantillas.

- Estas reglas están vinculadas con la pila tecnológica y el perfil de Calidad del mismo.

- Ejemplo:
  - ✓ Pila Tecnológica Java.
  - ✓ Proporciona los siguientes repositorios de reglas vinculados con su pila tecnológica:
    - ➢ Android Lint, Findbugs con todos sus variantes (FB-Contrib / Security Audit/minimal) , Sonar way, PMD, Checkstyle.

- De los cuales se seleccionan las reglas llegando un acuerdo entre los distintos roles implicados en el proceso de desarrollo. Haciendo explícito un acuerdo de compromiso de cumplimiento conforme a las severidades.

Perfiles de Calidad

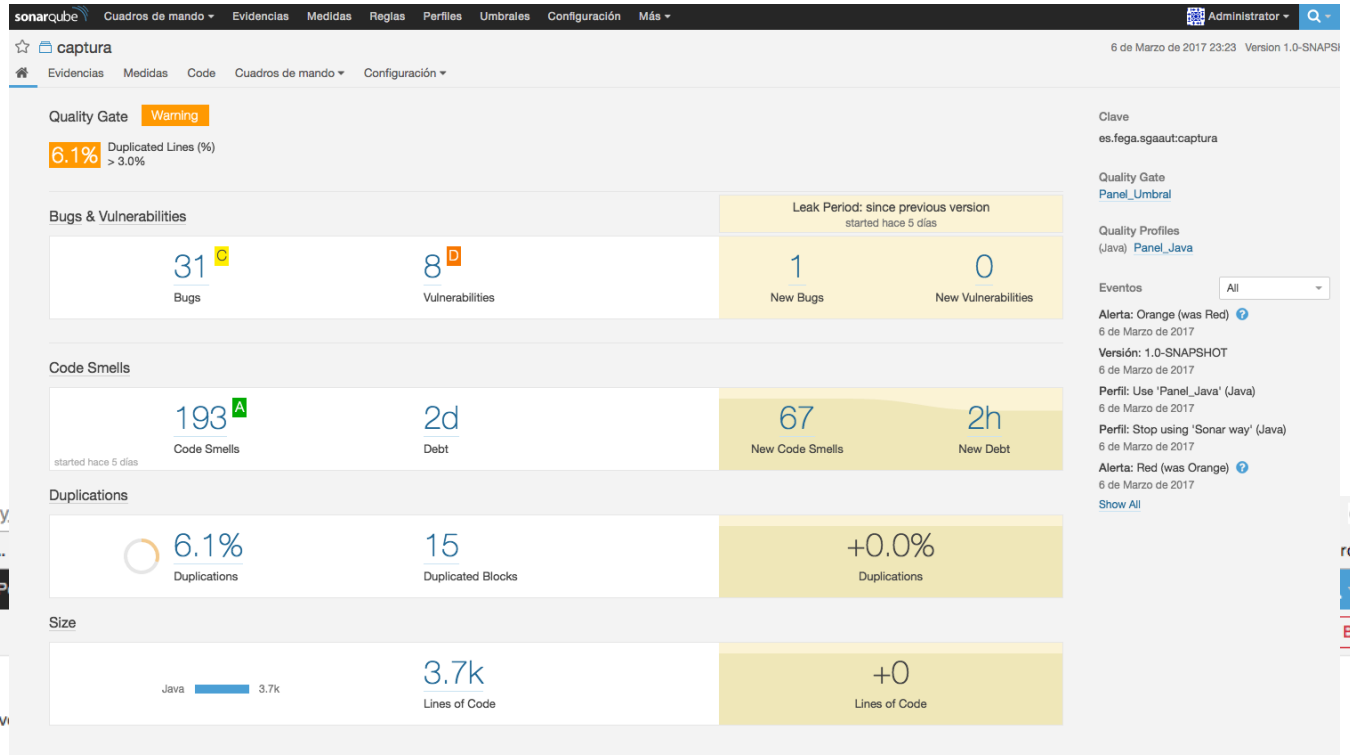


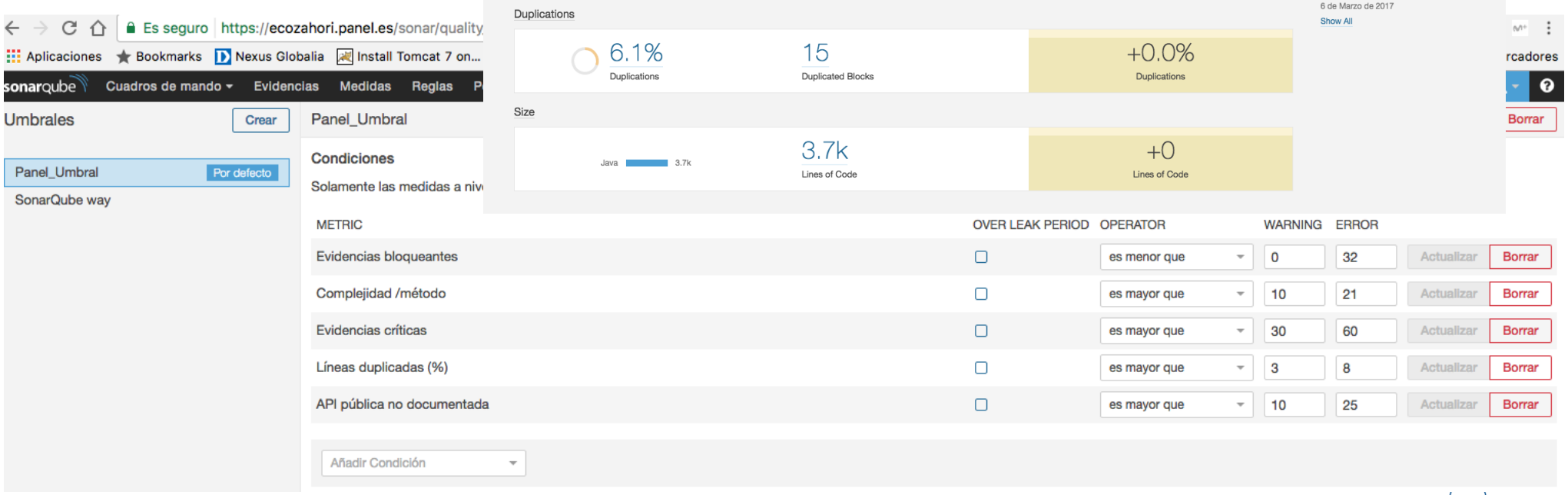
Umbrales de Calidad

- Es el lugar en el que definir los requisitos de los proyectos.
- Los requisitos se establecen contra las medidas, por ejemplo describir medidas descritas en la diapositiva:
  - ✓ Evidencias bloqueantes No es menor que 0 (Warning) 32 (Error)
  - ✓ Complejidad /método No es mayor que 10.0 (Warning) 21.0
  - ✓ Evidencias críticas No es mayor que 30 (Warning) 60
  - ✓ Líneas duplicadas (%) No es mayor que 3.0% (Warning) 8.0%
  - ✓ API pública no documentada No es mayor que 10 (Warning) 25 (Error)
- Lo ideal seria que todos los proyectos se verifiquen contra el mismo umbral de calidad, pero eso no es siempre práctico.
- Por ejemplo, es posible que:
  - ✓ La implementación tecnológica difiere de una aplicación a otra (tal vez no se requiera la misma cobertura de código en el código nuevo para Web o para aplicaciones Java).
  - ✓ Se quiere asegurar los requisitos más fuertemente en algunas de sus aplicaciones (frameworks internos por ejemplo).
  - ✓ Razón por la cual se pueden definir tantos umbrales de calidad como se desee.
- Las umbrales de Calidad se asocian a los proyectos que se deseen, en caso de que no tenga uno asociado se asocia el de por defecto.

Inspección Continua. Sonarqube

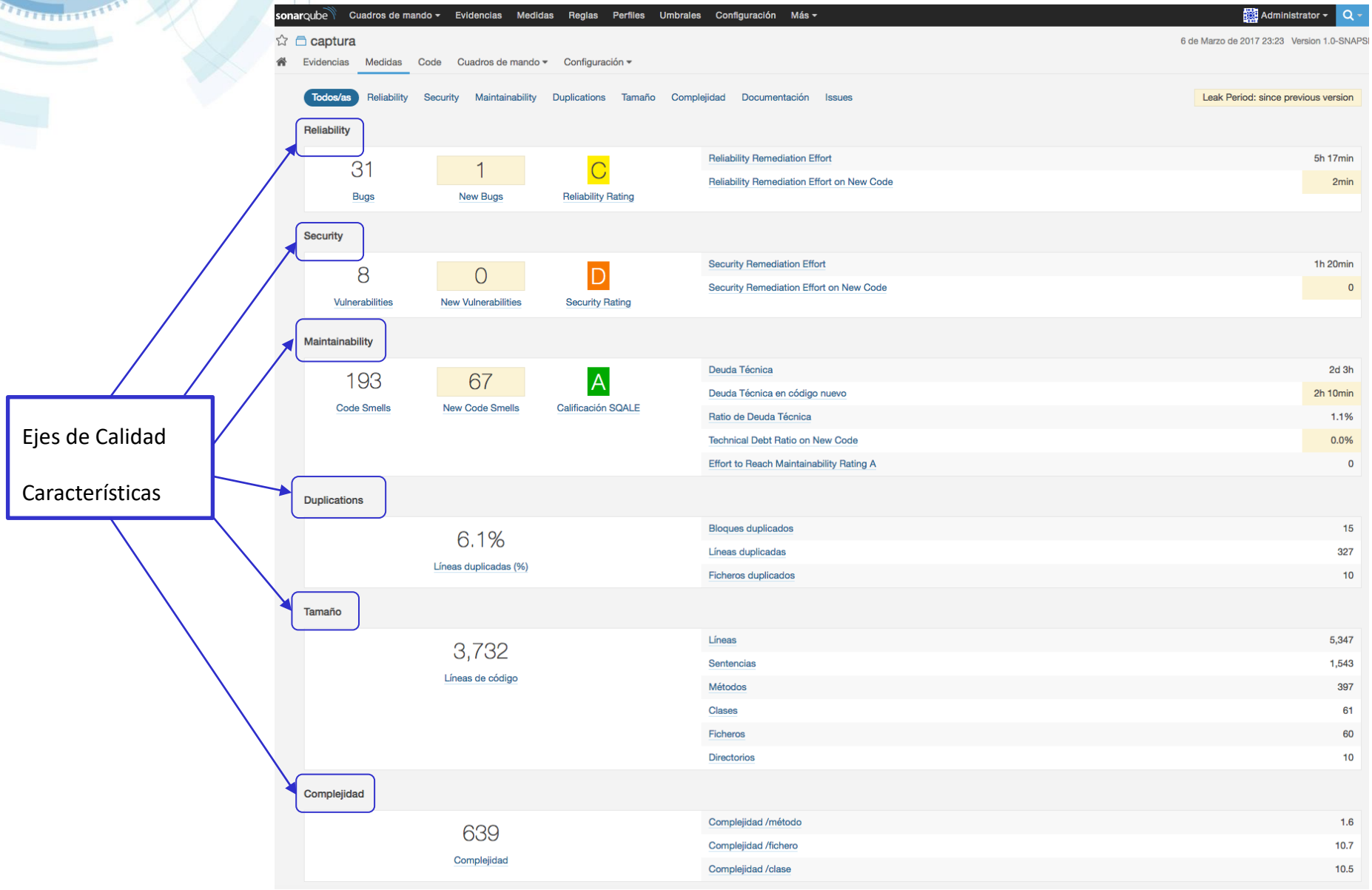
Umbrales de Calidad





Inspección Continua. Sonarqube

Métricas



Ejes de Calidad

Características

Detalle de Evidencias y vínculo con el código fuente

- Mientras se ejecuta un análisis, SonarQube presenta una evidencia cada vez que una parte del código incumple una regla de codificación. El conjunto de reglas se define mediante un perfil de calidad asociado al proyecto.

- Cada asunto tiene una de estas cinco severidades:

  - ✓ BLOQUEANTE:
    - Error con una alta probabilidad de impactar en el comportamiento de las aplicaciones en producción: pérdida de memoria, conexión JDBC no cerrada, ... El código DEBE ser inmediatamente corregido.
  - ✓ CRITICA:
    - Ya sea un error con una baja probabilidad de impacto en el comportamiento de la aplicación en producción o un tema que representa un fallo de seguridad: bloque catch vacío, inyección SQL, ... El código debe ser revisada de inmediato.
  - ✓ MAYOR:
    - Defecto de calidad que puede tener un alto impacto en la productividad de los desarrolladores: trozo de código no cubierto, bloques duplicados, parámetros no utilizados, ...

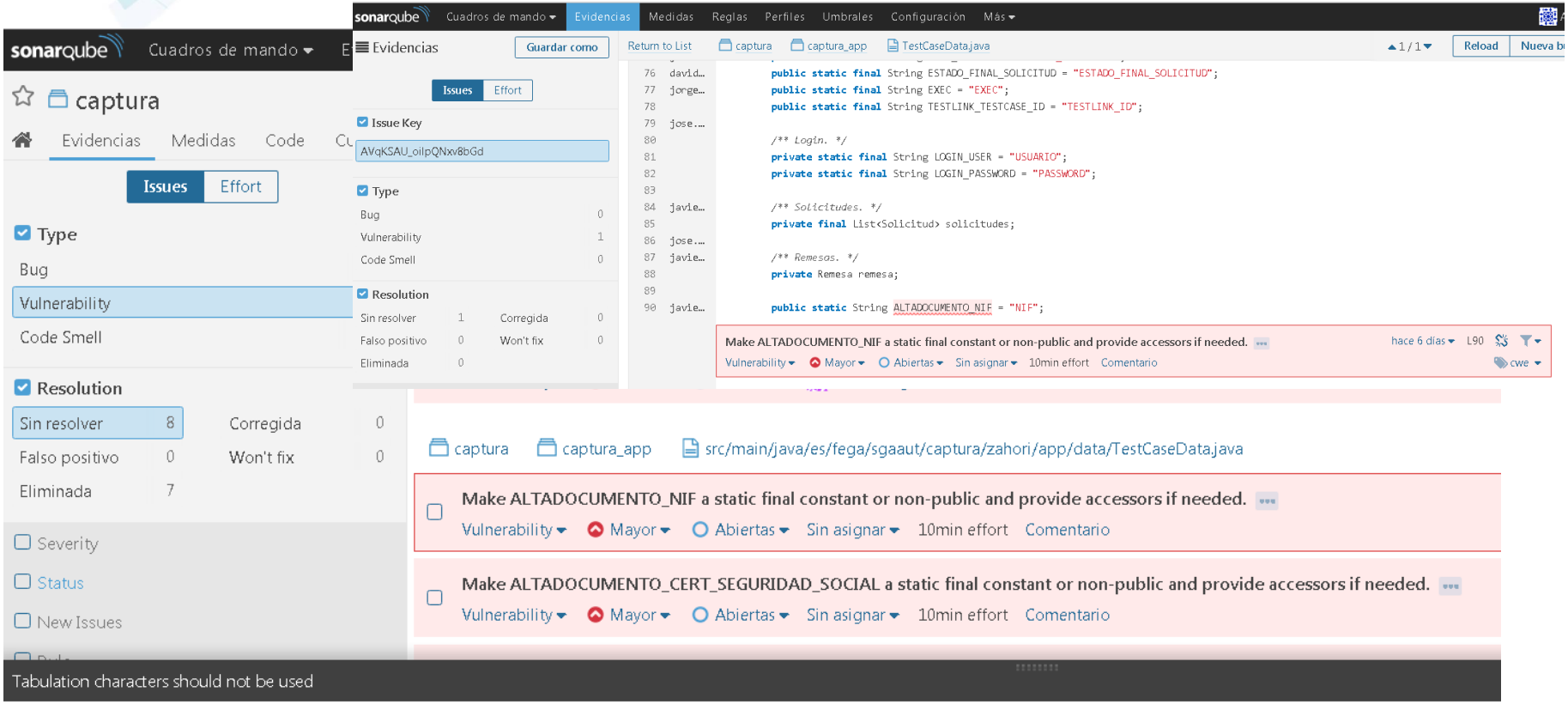
Detalle de Evidencias y vínculo con el código fuente

- Cada asunto tiene una de estas cinco severidades:

  - ✓ MENOR:
    - Defecto de calidad con un leve impacto en la productividad de los desarrolladores: líneas que no deberían ser tan largas, las sentencias "switch" deben tener al menos tres casos,...
  - ✓ INFO:
    - Ni un error, ni un defecto de calidad, sólo un hallazgo.

Inspección Continua. Sonarqube

Detalle de Evidencias



hands on

Jenkins

sonarqube

Más información en:

www.panel.es
panel@panel.es

panel.es
Panel Sistemas Informáticos, S.L.
Consultoría, servicios y soluciones TI.