■ □

# Tecnología

## Arquitectura

■ El sistema tiene una arquitectura definida.

□ Los componentes del sistema están separados entre frontend y backend.

■ El front y backend funcionan de forma separada (no están fusionados como en un sistema PHP “clásico”).

■ Toda validación realizada en el frontend también se realiza en el backend.

■ Se encuentran documentadas todos los puntos de entrada al backend.

## Base de datos

■ La comunicación a la base de datos solo se realiza desde el backend del sistema

■ La base de datos no es visible desde internet

■ El modelo construido implementa integridad referencial

■ Se analiza cada consulta SQL para verificar que se ejecuta de forma ´optima (usando todos los índices).

■ Se realizan pruebas de carga para simular el crecimiento esperado de la base de datos.

■ Se utiliza SQL parametrizado.

■ Se utiliza el dialecto SQL ANSI 1992 (o superior) para escribir las consultas.

■ La nomenclatura utilizada en los nombres de tablas, atributos, etc. es consistente en toda la base de datos.

■ Existe un diccionario de datos que describa cada elemento de la base de datos, para poder saber cuál es el propósito de cada tabla y atributo existente.

## Control de versiones

■ El código fuente del sistema se encuentra almacenado en un sistema de control de versiones (svn, github, etc)

□ Cada desarrollador sube sus cambios al sistema de control de versiones a lo menos una vez al dia

■ Se genera un “tag” (o release, o equivalente en el control de versiones) cada vez que se hace una entrega al cliente

## Logging

■ Se utiliza un framework establecido para realizar logging del funcionamiento del sistema.

■ El log queda registrado en algún archivo que permita su análisis.

□ El log se utiliza para resolver problemas.

## Restricciones

■ Se respetan las restricciones que la empresa pone en el proyecto

■ Se encuentran documentados los casos de uso/historias de usuario

■ Se tiene documentado el esquema de la base de datos

□ Existe un manual de usuario del sistema

□ Existe un manual de sistema

□ Existe un documento formal de entrega del sistema

# Convenciones de codificación

■ Existe una estándar definida para escribir el código fuente en el proyecto

■ Se aplica la misma identación en todo el proyecto

■ Se aplica la misma definición de uso del espacio en blanco en todo el proyecto

■ La posición de las llaves se aplica de forma consistente

□ Todo el código está escrito en el mismo idioma (español o ingles)

■ Todo el código usa el mismo estilo para escribir los nombres de los elementos (camel case, snake case, etc)

■ Todo el código usa la misma forma de escribir comentarios en el proyecto

# Metodología

Hay un Product Owner (PO) claramente definido

El PO está lo suficientemente empoderado para priorizar

El PO tienen el conocimiento para priorizar

El PO tiene contacto directo con el equipo

El PO tiene contacto directo con los stakeholders

El PO habla con una voz (´el/ella toma las decisiones)

El equipo tiene un backlog por sprint

Es altamente visible

Se actualiza todos los días

El equipo es el dueño exclusivo

Se realizan las daily scrum meetings

Todo el equipo participa

Salen a la superficie los problemas e impedimentos

Se realiza una demo después de cada sprint

Se muestra el software funcionando y testeado

Se recibe feedback de los stakeholders y el PO

Tienen una definición de “hecho” (DDH)

El DDH se alcanza en cada iteración

El equipo respeta el DDH

Se realiza la retrospectiva después de cada sprint

Se generan propuestas concretas de mejora

Algunas propuestas realmente se concretan

El equipo completo (+PO) participa

El PO tiene un backlog del producto (PBL)

Los ´ítems principales están priorizados de acuerdo con el valor que entregan al negocio

Los ´ítems principales están estimados

Las estimaciones las hizo el equipo

Los ´ítems principales caben en una iteración

El PO entiende el propósito de todos los ´ítems del backlog

Se realizan las reuniones de sprint planning

Participa el PO

El PO trae la PBL actualizada

El equipo completo participa

El resultado es el plan del sprint

El equipo completo cree que el plan se puede alcanzar

El PO está satisfecho con la priorización

Las iteraciones tienen duración predefinida

La duración es de 4 semanas o menos

Siempre terminan a tiempo

El equipo no es interrumpido ni controlado por externos

El equipo generalmente entrega lo comprometido

Todos los miembros del equipo se sientan juntos

9 personas como máximo por equipo

(Faltan algunas)

# Testing

Se realiza Testing Interno

Existen tests unitarios definidos para testear unidades de c´odigo

Estos tests unitarios se ejecutan “frecuentemente” (cada vez que se sube una versi´on al sistema de control de versiones)

Los tests unitarios cubren las unidades de c´odigo con algoritmos complicados

que requieran testing

Existen UATs definidos

Los UATs los ejecuta el equipo scrum antes de hacer la entrega al cliente

Los UATs los ejecuta el cliente para verificar si el software cumple sus objetivos

Los UATs fueron creados por el equipo en coordinaci´on con el Product Owner

El equipo es conciente de la calidad estructural del proyecto

El equipo realiza acciones para mejorar la calidad estructural del proyecto

El equipo es conciente de la calidad funcional del proyecto

El equipo realiza acciones para mejorar la calidad funcional del proyecto

# Las 10 Pautas

Escribir c´odigo corto

Las unidades de c´odigo no tienen m´as de 15 l´ıneas de longitud

Las unidades de c´odigo tienen nombres expresivos

Las unidades de c´odigo tienen nombres que reflejen su funcionamiento

Escribir c´odigo simple

Las unidades de c´odigo no tienen m´as de 4 puntos de ramificaci´on

Las unidades de c´odigo no contienen m´as de un ciclo cada una

Las unidades de c´odigo no contienen m´as de 5 variables cada una

Escribir el c´odigo solo una vez

No existe c´odigo evidentemente repetido

No existen valores literales (num´ericos o textuales) repetidos en el c´odigo Interfaces de unidades peque˜nas

Las unidades de c´odigo no tienen m´as de 4 par´ametros

Las variables relacionadas se consolidan en “grupos de datos”

Preocupaciones Separadas en M´odulos

Se nota que cada m´odulo tiene una sola responsabilidad

Los m´odulos no tienen m´as de 100 l´ıneas de longitud

Acopla Sueltamente los Componentes de la Arquitectura

Es posible revisar las dependencias entre los diferentes m´odulos

No existen m´odulos de los que depende gran parte del sistema

Mantener Balanceados los Componentes de la Arquitectura

El sistema tiene una cantidad apropiada de componentes (entre 6 y 12)

Los componentes del sistema tienen un tama˜no similar entre ellos

Mant´en peque˜na tu base de c´odigo

No existe c´odigo “muerto”

El c´odigo est´a dividido en “proyectos” que mantengan separadas las funcionalidades (o capas)

No hay c´odigo duplicado entre dichos “proyectos”

No existe funcionalidad “extra” (funcionalidad que no derive de los requisitos)

Cada “proyecto” tiene menos de 50.000 l´ıneas de c´odigo

Automatiza los Tests

Existen tests

Los tests repetitivos est´an automatizados

Los tests automatizados se ejecutan de forma frecuente (a lo menos una vez al dia)

Se mide el porcentaje de cobertura de los tests

Existen tests cualitativos (UATs)

Los tests cualitativos se ejecutan frecuentemente (a lo menos una vez por iteraci´on)

Escribe c´odigo limpio

Todo el c´odigo ha sido revisado por a lo menos dos personas (el que lo escribi ´o

y un revisor)

Se realizan sesiones semanales de aprendizaje donde se revisa el c´odigo de la

semana para destacar aprendizajes a partir de buen y mal c´odigo generado en

el proyecto