UNIVERSIDAD CATÓLICA DEL NORTE ESCUELA DE INGENIERÍA INGENIERÍA CIVIL EN COMPUTACIÓN E INFORMÁTICA



PROYECTO DE TÓPICOS AVANZADOS DE SOFTWARE CHECKLIST USA-SHOP

Nicolás Pino

Lucas Ramírez

Felipe Varas

Docente.: Eric Ross

Tabla de contenido

1.	Tecn	nología	3
1	1.1.	Checklist	3
	1.1.1	Arquitectura	3
	1.1.2	2. Base de datos	3
	1.1.3	3. Control de versiones	4
	1.1.4	Logging	4
	1.1.5	5. Restricciones	4
2.	Conv	venciones de codificación	5
2	2.1.	Checklist	5
3.	Meto	odología	5
3	3.1.	Checklist	5
4.	Testi	ing	9
4	l.1. (Checklist	9
5.	Las 1	10 pautas	10
5	5.1. (Checklist	10
	5.1.1	. Escribir código corto	10
	5.1.2	2. Escribir código simple	10
	5.1.4	Interfaces de unidades pequeñas	10
	5.1.5	5. Preocupaciones separadas en módulos	10
	5.1.6	6. Acopla sueltamente los componentes de la arquitectura	10
	5.1.7		
	5.1.8	B. Mantén pequeña tu base de código	11
	5.1.9	9. Automatiza los Tests	. 11
	5.1.1	0. Escribir código corto	11

1. Tecnología

1.1. Checklist

Marque cada — de acuerdo con si el equipo actualmente cumple el control:

1.1.1. 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 en se realiza en el backend.
- Se encuentran documentadas todos los puntos de entrada al backend (parámetros de entrada, respuestas posibles, consideraciones especiales, etc).

1.1.2. 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 óptima (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.

1.1.3. 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 día.
- Se genera un "tag" (o release, o equivalente en el control de versiones) cada vez que se hace una entrega al cliente.

1.1.4. 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.

1.1.5. 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

2. Convenciones de codificación

2.1. Checklist

Marque cada ☐ de acuerdo con si el equipo actualmente cumple el control:

- ✓ 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 inglés).
- ✓ 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.

3. Metodología.

3.1. Checklist

Marque cada ☐ de acuerdo con si el equipo actualmente cumple el control:

- ✓ 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 (él/ella toma las decisiones).
- El equipo tiene un backlog por sprint.
 - Es altamente visible.
 - Se actualiza todos los días.
 - ✓ El equipo es 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.
J	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 al 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 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.

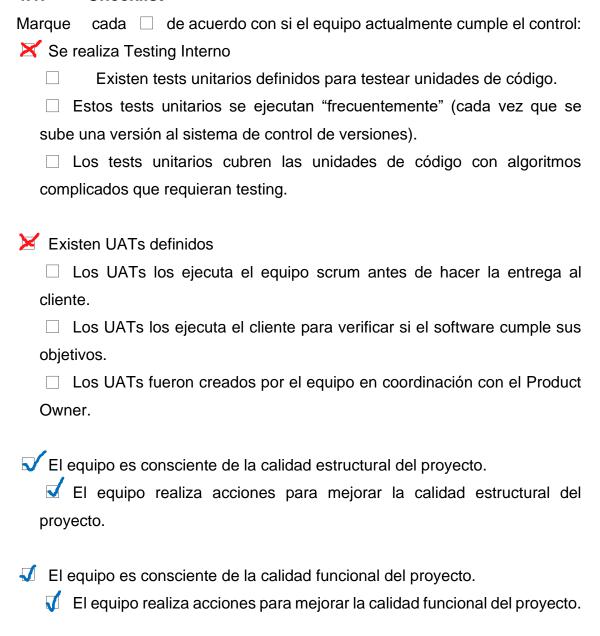
Los siguientes son características recomendadas, pero no siempre necesarias:

- ▼ El equipo tiene habilidades necesarias para llevar los ítems del backlog al estado "hecho".
- Los miembros del equipo no están encasillados en roles específicos.
- 🗶 Las iteraciones condenadas a fallar se terminan de forma anticipada.
- ☑ El PO tiene una visión del producto que está en sincronía con el PBL.
- ✓ PBL y la visión del producto son altamente visibles.
- ▼ Todos los miembros del equipo participan en las estimaciones.
- ✓ El PO está disponible cuando el equipo está estimando.
- ✓ Las estimaciones se realizan usando "tamaño relativo" en vez de "tiempo".

	El e	quipo tiene un Scrum Master (SM).
	V	El SM se sienta con el equipo.
J	Todo	o el equipo conoce 1 a 3 impedimentos.
	XE	El SM tiene una estrategia de como corregir el impedimento más
	impor	tante.
	X	El SM se enfoca en remover impedimentos.
	X	Los impedimentos se escalan a la administración cuando el equipo no
	los pu	ede resolver.
V	Los	ítems del PBL se rompen en tareas dentro de una iteración.
	J	Las tareas de las iteraciones se estiman.
	X	Las estimaciones de las tareas en las que se está trabajando se
	actua	lizan todos los días.
X	Se n	nide la velocidad del equipo.
X	Se n	nide la velocidad del equipo. Todos los ítems en la iteración tienen una estimación.
X	Se n	· ·
X	Se n	Todos los ítems en la iteración tienen una estimación.
		Todos los ítems en la iteración tienen una estimación. El PO usa la velocidad para planificación las "releases". La velocidad solo incluye ítems que están "hechos".
		Todos los ítems en la iteración tienen una estimación. El PO usa la velocidad para planificación las "releases". La velocidad solo incluye ítems que están "hechos". quipo tiene un "sprint burndown chart"
		Todos los ítems en la iteración tienen una estimación. El PO usa la velocidad para planificación las "releases". La velocidad solo incluye ítems que están "hechos".
		Todos los ítems en la iteración tienen una estimación. El PO usa la velocidad para planificación las "releases". La velocidad solo incluye ítems que están "hechos". quipo tiene un "sprint burndown chart"
	El ed	Todos los ítems en la iteración tienen una estimación. El PO usa la velocidad para planificación las "releases". La velocidad solo incluye ítems que están "hechos". quipo tiene un "sprint burndown chart" Es altamente visible. Se actualiza todos los días.
	El ed	Todos los ítems en la iteración tienen una estimación. El PO usa la velocidad para planificación las "releases". La velocidad solo incluye ítems que están "hechos". quipo tiene un "sprint burndown chart" Es altamente visible. Se actualiza todos los días. aily scrum se realiza todos los días, a la misma hora, en el mismo lugar.
√	El ed	Todos los ítems en la iteración tienen una estimación. El PO usa la velocidad para planificación las "releases". La velocidad solo incluye ítems que están "hechos". quipo tiene un "sprint burndown chart" Es altamente visible. Se actualiza todos los días.
√	El ed	Todos los ítems en la iteración tienen una estimación. El PO usa la velocidad para planificación las "releases". La velocidad solo incluye ítems que están "hechos". quipo tiene un "sprint burndown chart" Es altamente visible. Se actualiza todos los días. aily scrum se realiza todos los días, a la misma hora, en el mismo lugar.

4. Testing

4.1. Checklist



5. Las 10 pautas

5.1. Checklist

Marque cada

☐ de acuerdo con si el equipo actualmente cumple el control:

5.1.1. Escribir código corto

- ✓ Las unidades de código no tienen más de 15 líneas de longitud.
- Las unidades de código tienen nombres expresivos.
- Las unidades de código tienen nombres que reflejen su funcionamiento.

5.1.2. Escribir código simple

- √ Las unidades de código no tienen más de 4 puntos de ramificación.
- ✓ Las unidades de código no contienen más de un ciclo cada una.
- Las unidades de código no contienen más de 5 variables cada una.

5.1.3. Escribir código solo una vez

- ✓ No existe código evidentemente repetido.
- No existen valores literales (numéricos o textuales) repetidos en el código.

5.1.4. Interfaces de unidades pequeñas

- ✓ Las unidades de código no tienen más de 4 parámetros.
- ✓ Las variables relacionadas se consolidan en "grupos de datos".

5.1.5. Preocupaciones separadas en módulos

- Se nota que cada módulo tiene una sola responsabilidad.
- X Los módulos no tienen más de 100 líneas de longitud.

5.1.6. Acopla sueltamente los componentes de la arquitectura

- ✓ Es posible revisar las dependencias entre los diferentes módulos.
- ✓ No existen módulos de los que depende gran parte del sistema.

5.1.7. 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 tamaño similar entre ellos.

5.1.8. Mantén pequeña tu base de código

- ✓ No existe código "muerto".
- ✓ El código está dividido en "proyectos" que mantengan separadas las funcionalidades (o capas).
- No hay código duplicado entre dichos "proyectos".
- No existe funcionalidad "extra" (funcionalidad que no derive de los requisitos).
- Cada "proyecto" tiene menos de 50.000 líneas de código.

5.1.9. Automatiza los Tests

	Existen tests			
	Los tests repetitivos están automatizados.			
	Los tests automatizados se ejecutan de forma frecuente (a lo menos			
una vez al día).				
	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 iteración).				

5.1.10. Escribir código corto

✓ Todo el código ha sido revisado por a lo menos dos personas (el que lo escribió y un revisor).

Se realizan sesiones semanales de aprendizaje donde se revisa el código de la semana para destacar aprendizajes a partir de buen y mal código generado en el proyecto.