Breve resumen en base a: 2020-Scrum-Guide-Spanish-Latin-South-American.pdf

Esta guía define Scrum, un marco de trabajo liviano para generar valor mediante soluciones adaptativas para problemas complejos. Scrum se basa en el empirismo y el pensamiento Lean, empleando un enfoque iterativo e incremental.

El marco de trabajo Scrum es inmutable y solo existe en su totalidad.

Scrum requiere un Scrum Master que fomente un entorno donde:

1. Un Product Owner ordena el trabajo en un Product Backlog.
2. El Scrum Team convierte una selección del trabajo en un Increment durante un Sprint.
3. El Scrum Team y los interesados inspeccionan los resultados y se adaptan para el próximo Sprint.
4. Se repite el proceso.

Teoría de Scrum

Scrum se fundamenta en la *transparencia, inspección y adaptación.* La transparencia permite la inspección, que a su vez posibilita la adaptación.

Valores de Scrum

El éxito de Scrum depende de que las personas se vuelvan más competentes en cinco valores: Compromiso, Foco, Franqueza, Respeto y Coraje.

Scrum Team

El Scrum Team consta de un Scrum Master, un Product Owner y Developers. Es multifuncional, autogestionado y típicamente de 10 personas o menos.

Developers

Los Developers se comprometen a:

- Crear un plan para el Sprint (Sprint Backlog)
- Inculcar calidad adhiriéndose a la Definición de Terminado
- Adaptar su plan diariamente hacia el Objetivo del Sprint
- Responsabilizarse mutuamente como profesionales

Product Owner

El Product Owner maximiza el valor del producto y gestiona eficazmente el Product Backlog:

- Desarrolla y comunica el Objetivo del Producto
- Crea y comunica los elementos del Product Backlog
- Ordena los elementos del Product Backlog

- Asegura que el Product Backlog sea transparente y entendible

Scrum Master

El Scrum Master establece Scrum como se define en la Guía de Scrum:

- Guía al equipo en la autogestión y multifuncionalidad
- Ayuda a enfocarse en crear Increments de alto valor
- Elimina impedimentos al progreso del equipo
- Asegura que los eventos de Scrum ocurran y sean productivos

Eventos de Scrum

El Sprint

Los Sprints son el corazón de Scrum, donde las ideas se convierten en valor. Tienen una duración fija de un mes o menos.

Sprint Planning

La Sprint Planning inicia el Sprint estableciendo el trabajo a realizar. Aborda tres temas:

- Por qué este Sprint es valioso (Objetivo del Sprint)
- Qué se puede hacer en este Sprint
- Cómo se realizará el trabajo elegido

Daily Scrum

La Daily Scrum es un evento de 15 minutos para los Developers con el fin de inspeccionar el progreso hacia el Objetivo del Sprint y adaptar el Sprint Backlog. Ayer: qué hicimos, dificultades. Hoy: que vamos a hacer

Sprint Review

El propósito de la Sprint Review es inspeccionar el resultado del Sprint y determinar futuras adaptaciones. El Scrum Team presenta los resultados a los interesados clave y se discute el progreso hacia el Objetivo del Producto.

Básicamente le muestra el producto al PO.

Definition of Done: identificar si cada US que voy a agregar cumple con las condiciones que dijimos que tiene que cumplir.

Sprint Retrospective

La Sprint Retrospective planifica formas de aumentar la calidad y la efectividad. El Scrum Team inspecciona el último Sprint y identifica mejoras para implementar en el próximo Sprint.

Usar técnicas que despersonalicen la inspección. Poder elegir acciones puntuales para corregir aquello que se hizo mal.

Artefactos de Scrum

Product Backlog

El Product Backlog es una lista emergente y ordenada de lo que se necesita para mejorar el producto. Es la única fuente del trabajo para el Scrum Team.

Compromiso: Objetivo del Producto

El Objetivo del Producto describe un estado futuro del producto y sirve como meta para que el Scrum Team planifique.

Sprint Backlog

El Sprint Backlog se compone del Objetivo del Sprint, los elementos del Product Backlog seleccionados y un plan para entregar el Increment.

Compromiso: Objetivo del Sprint El Objetivo del Sprint es el único propósito del Sprint y proporciona flexibilidad en términos del trabajo exacto necesario para lograrlo.

Increment

Un Increment es un paso concreto hacia el Objetivo del Producto. Cada Increment se suma a todos los Increments anteriores y se verifica minuciosamente.

Compromiso: Definición de Terminado (Definition of Done) La Definición de Terminado es una descripción formal del estado del Increment cuando cumple con las medidas de calidad requeridas para el producto.

Definiciones importantes

Definición de Hecho ( )

Es una definición propia del equipo, que se valida con un **checklist** donde se especifican todas las características que debe tener una **User Story (US)** para considerarse **decentemente terminada** y presentársela al **Product Owner (PO)**.
Debe cumplir con los siguientes criterios:

- **Pruebas unitarias y de aceptación en verde**.
- **Pasar el ambiente de prueba**.

Definición de Listo ( )

Permite definir qué debe cumplir una **US** para asegurarnos de que está lista para ser implementada.
Es una definición **acordada por el equipo**, a partir de la cual se arma un **checklist** para determinar si la US cumple con los aspectos pactados.

- Si cumple con **todos los criterios**, puede ser incluida en un sprint.
- Si **no cumple** con alguno, se debe trabajar más en esa US antes de incluirla.

Spike

Tipo especial de **US** que sirve para **quitar riesgo o incertidumbre** de otro requerimiento, de otra US o de alguna faceta del proyecto que se quiere investigar.
Son creadas **específicamente para este fin**.

Spike Técnica

Utilizadas para **investigar enfoques técnicos** en el dominio de la solución.
Se aplican cuando el equipo necesita una **comprensión más fiable** sobre alguna tecnología antes de comprometerse a desarrollar una nueva funcionalidad en un tiempo fijo.

Spike Funcional

Utilizadas cuando hay **incertidumbre respecto a cómo el usuario interactuará** con el sistema.
Generalmente se evalúan con **prototipos** para obtener retroalimentación de los usuarios o involucrados.

Normalmente se trabajan **un sprint antes** del sprint donde se implementará la US que tiene incertidumbre.
Los **spikes** deben ser:

- **Demostrables**
- **Estimables**
- **Aceptables**

Cumpliendo el criterio de .

> **El spike es una excepción**, no siempre se debe aplicar; es la **última opción**.
> Antes de implementarlo, se gestiona todo dentro de la misma US.

Testing de Software

El testing está en el contexto del aseguramiento de calidad (QA). La prueba de software no incluye a control de calidad de proceso ni de calidad de producto.

El testing es un proceso **destructivo** cuyo **objetivo es el de encontrar defectos**. Se asume la presencia de defectos. Mundialmente, este proceso representa el 30-50% del costo de un software confiable.

El costo de corregir un error aumenta exponencialmente en el tiempo.

Es necesario por distintos motivos:

- La existencia de defectos en el software es inevitable.
- Se evitan demandas de clientes
- Se reducen riesgos
- Se construye confianza en el producto
- Las fallas son muy costosas
- Para verificar que el software se ajusta a los requerimientos y validar que las funciones se implementan correctamente.

Asegurar Calidad Controlar Calidad

Una vez definidos los requerimientos de calidad, se debe tener en cuenta que la calidad no se puede inyectar al final, y que la calidad depende de tareas realizadas durante todo el proceso. Detectar errores ahorra recursos. El testing **no puede asegurar ni calidad en el software ni software de calidad**.

Sigue los siguientes principios:

- El testing muestra presencia de defecto.
- El testing exhaustivo es imposible
- Testing temprano
- Agrupamiento de Defectos
- Paradoja del Pesticida
- El testing es dependiente del contexto
- Falacia de la ausencia de errores (se asume la presencia de errores)
- Un programador debería evitar probar su propio código.
- Una unidad de programación no debería probar sus propios desarrollos.
- Examinar el software para probar que no hace lo que se supone que debería hacer es la mitad de la batalla, la otra mitad es ver que hace lo que no se supone que debería hacer.
- No planificar el esfuerzo de testing sobre la suposición de que no se van a encontrar defectos.

ERROR vs DEFECTO

- El error se descubre a partir de técnicas específicas que me permiten encontrar los mismos dentro de la misma etapa en la que estoy trabajando.

- Los defectos nos demuestran un error no detectado que se trasladó a una etapa siguiente.

En el testing encontramos defectos, ya que justamente se encuentran cosas incorrectas que se realizaron en la etapa de implementación que es la etapa anterior. Más adelante en el contexto de revisiones técnicas e inspecciones vamos a ver más a fondo que son los errores y como encontrarlos.

Obviamente siempre es mejor encontrar errores antes que defectos.

SEVERIDAD y PRIORIDAD

Cuando hablamos de defectos encontrados durante el testing, debemos considerar dos aspectos:

SEVERIDAD

Tiene que ver con la gravedad del defecto que encontré.
Un defecto puede ser bloqueante y no permitirme seguir con el caso de prueba, crítico, o menor, hasta **cosmético** (sintaxis, ortografía, visualización, etc.).

PRIORIDAD

Urgencia que tenemos para resolver este defecto.
Aunque podría intuirse que según la severidad habrá tal o cual prioridad, esto no siempre es así.
Depende del contexto en el cual nos encontramos, por eso es importante tener los dos aspectos identificados.

Niveles de Testing

Existen 4 niveles principales: (aunque también existe el adHoc Testing)

- Testing unitario o pruebas unitarias
- Pruebas de integración
- Pruebas de sistema
- Pruebas de aceptación

Además, existen 4 ambientes de prueba:

- Desarrollo: testing unitario
- Prueba: testing unitario y de integración
- Pre - Producción: pruebas de sistema y a veces de aceptación
- Producción: pruebas de aceptación

Testing Unitario

- Se prueba cada componente tras su realización/construcción.
- Solo se prueban componentes individuales.
- Se encuentran errores en lugar de defectos.
- Cada componente es probado de forma independiente
- Se produce con acceso al código bajo pruebas y con el apoyo del entorno de desarrollo, tales como un framework de pruebas unitarias o herramientas de depuración.
- Los errores se suelen reparar tan pronto como se encuentran, sin constancia oficial de los incidentes.
- A diferencia de las demás pruebas, lo realizan los desarrolladores. Suelen estar automatizadas.

Ejemplo en Python:

```python
import unittest

def sum(a, b):
    return a + b

class TestSum(unittest.TestCase):
    def test_sum(self):
        self.assertEqual(sum(5, 3), 8)
        self.assertEqual(sum(-1, 1), 0)

if __name__ == '__main__':
    unittest.main()
```

Testing de Integración

- Test orientado a verificar que las partes de un sistema que funcionan bien aisladamente (o sea, pasaron por el testing unitario correctamente), también lo hacen en conjunto.
- Cualquier estrategia de prueba de versión o de integración debe ser incremental, para lo que existen dos esquemas principales:
- Integración de arriba hacia abajo (top-down)
- Integración de abajo hacia arriba (bottom-up).
- Lo ideal es una combinación de ambos esquemas.
- Tener en cuenta que los módulos críticos deben ser probados lo más tempranamente posible.
- Los puntos clave del test de integración son simples:
- Conectar de a poco las partes más complejas
- Minimizar la necesidad de programas auxiliares

Testing de Sistema

- Es la prueba realizada cuando una aplicación esta funcionando como un todo (Prueba de la construcción Final).
- Trata de determinar si el sistema en su globalidad opera satisfactoriamente (recuperación de fallas, seguridad y protección, stress, performance, etc.)
- El entorno de prueba debe corresponder al entorno de producción tanto como sea posible para reducir al mínimo el riesgo de incidentes debidos al ambiente específicamente y que no se encontraron en las pruebas.
- Deben investigar tanto requerimientos funcionales y no funcionales del sistema.

Testing de Aceptación

- Es la prueba realizada por el usuario para determinar si la aplicación se ajusta a sus necesidades.
- La meta en las pruebas de aceptación es el de establecer confianza en el sistema, las partes del sistema o las características específicas y no funcionales del sistema.
- Encontrar defectos no es el foco principal en las pruebas de aceptación.
- Comprende tanto la prueba realizada por el usuario en ambiente de laboratorio (pruebas alfa), como la prueba en ambientes de trabajo reales (pruebas beta).

AMBIENTES PARA CONSTRUCCIÓN DEL SOFTWARE

Los ambientes son los lugares donde se trabaja para el desarrollo de software. Cada ambiente cumple un propósito específico.

AMBIENTE DE DESARROLLO

El ambiente de desarrollo es donde los desarrolladores crean, prueban y depuran el software. Es un entorno local o en red utilizado por los desarrolladores para escribir y probar el código antes de integrarlo con el resto del sistema. Los programadores pueden utilizar herramientas de desarrollo, depuración y pruebas para garantizar que el software cumpla con los requisitos establecidos. Este ambiente suele ser flexible y permite a los desarrolladores experimentar y probar nuevas ideas sin afectar los entornos de producción. **Las pruebas unitarias se llevan a cabo en el ambiente de desarrollo**

AMBIENTE DE PRUEBA

El ambiente de pruebas es donde se llevan a cabo pruebas exhaustivas del software desarrollado. **Aquí se realizan pruebas de integración, pruebas funcionales, pruebas de rendimiento** y otras pruebas relevantes para verificar que el software cumpla con los requisitos establecidos y funcione correctamente.

El ambiente de prueba suele ser una réplica o tener características parecidas al entorno de producción, pero sin afectar a los usuarios finales y sin TODAS las características (ya que es caro tener ambientes iguales al de producción). Se utilizan conjuntos de datos y configuraciones similares a las del entorno de producción para garantizar una prueba más precisa del software. **Las pruebas de integración se realizan en el ambiente de pruebas**

AMBIENTE DE PRE-PRODUCCIÓN

El ambiente de preproducción, también conocido como entorno de puesta en escena o entorno de calidad, es donde se realiza una prueba final del software antes de su lanzamiento en producción. **Aquí se simulan las condiciones del entorno de producción y se realizan pruebas de último minuto, como pruebas de estrés,pruebas de carga y pruebas de seguridad**. El objetivo es validar y verificar que el software esté listo para ser implementado en el entorno de producción sin problemas significativos. **Las pruebas de sistema se llevan a cabo en el ambiente de preproducción**

AMBIENTE DE PRODUCCIÓN

El ambiente de producción es donde el software está en funcionamiento y es accesible a los usuarios finales. Es el entorno real en el que el software se utiliza para realizar las tareas y funciones para las que fue diseñado. Este ambiente suele ser altamente controlado y está configurado para ser escalable, seguro y confiable. Se implementan medidas de respaldo y recuperación ante desastres para garantizar la disponibilidad continua del software. Es el ambiente en el que el software está funcionando. **Las pruebas de aceptación se llevan a cabo en el ambiente de preproducción o en un entorno similar al de producción (en el ambiente de prueba)**

Casos de Prueba

El testing exhaustivo de TODAS las opciones del software no es posible, es inviable, es sumamente caro. Entre el 20% y el 50% del costo de software es en testing.

Un caso de prueba (CP) Set de condiciones o variables bajo las cuales un tester determinará si el software está funcionando correctamente o no. Incluye las entradas correspondientes y el resultado esperado.

El propósito de diseñar casos de prueba es encontrar el subconjunto de todos los casos que tienen mayor probabilidad de detectar el mayor numero posible de defectos.

Buena definición de casos de prueba nos ayuda a **reproducir** defectos. Utiliza datos específicos, como por ejemplo “ingresa el usuario ‘juan@gmail.com’ ” en

lugar de “ingresa el email del usuario”.

- Objetivo: maximizar errores encontrados
- Criterio: en forma completa
- Restricción: con el mínimo esfuerzo y tiempo

Un caso de prueba es la unidad de la actividad de la prueba. Consta de tres partes:

1. Objetivo: la característica del sistema a comprobar
2. Datos de entrada y de ambiente: datos a introducir al sistema que se encuentra en condiciones preestablecidas.
3. Comportamiento esperado: La salida o la acción esperada en el sistema de acuerdo a los requerimientos del mismo.

Generación de Casos de Prueba

- Ninguna técnica es completa, solucionan distintos problemas
- Lo mejor es combinar varias de estas técnicas para complementar las ventajas de cada una
- Depende del código a testear
- Sin requerimientos todo es mucho más difícil
- Tener en cuenta la conjetura de defectos
- Ser sistemático y documentar las suposiciones sobre el comportamiento o el modelo de fallas

Se pueden derivar casos de prueba desde:

- Documentos del cliente
- Información de relevamiento
- Requerimientos
- Especificaciones de programación
- Código

Condiciones de Prueba

- Esta es la reacción esperada de un sistema frente a un estímulo particular, este estímulo está constituido por las distintas entradas.
- Una condición de prueba debe ser probada por al menos un caso de prueba
- Se definen antes que los casos de prueba

Métodos

Como el tiempo y el presupuesto es limitado, se deben utilizar métodos que prueben la mayor cantidad de funcionalidades con la menor cantidad de pruebas.

- El documento usado para asegurarse que los requerimientos son cubiertos cuando se escriben los test cases es la **matriz de trazabilidad**.

CAJA BLANCA vs CAJA NEGRA

CAJA BLANCA	CAJA NEGRA
Testing basado en el analisis de la especificacion de una porcion del software sin referencia a su estructura interna. Definimos entradas y su resultado esperado, sin conocer el código.	Esta tecnica de testing examina la estructura basica de un programa y deriva los datos de testeo desde la logica del programa, asegurandose que todas las sentencias y condiciones se ejecutan al menos una vez. Se basan en el análisis de la estructura interna del software o un componente del software.
Se clasifican en: **Basado en especificaciones:** Partición de Equivalencias - Análisis de valores límites- etc. **Basados en la experiencia (no trabajamos esto en el práctico)**: Adivinanza de Defectos - Testing Exploratorio	Hay distintos métodos, algunos de ellos son: COBERTURA DE ENUNCIADOS O CAMINOS BASICOS - COBERTURA DE SENTENCIAS - COBERTURA DE DECISIÓN - COBERTURA DE CONDICIÓN - COBERTURA MÚLTIPLE