

Resumen de Ingeniería en Software

Cursada 2021

Por Mauro y Fede

Índice

Unidad 1 - Introducción	4
¿Qué es la Ingeniería de Software?	4
Body of Knowledge (BoK)	4
SWEBoK (Software Engineering Body of Knowledge)	4
The 15 SWEBOK KAs	4
Diagrama Causa-Efecto (Diag. De Ishikawa o Espina de Pescado)	5
Classic Mistake (paper)	6
¿Qué son los classic mistakes?	6
Clasificaciones	6
Unidad 2 - Calidad de Software	7
¿Qué es la Calidad?	7
Costos de no calidad	7
Pirámide de jerarquía de factores de calidad	8
Visiones de calidad	8
ISO 25010 – Calidad del producto de software	9
Modelos de Calidad de Proceso	10
CMMI (Capability Maturity Model Integrated)	10
¿Qué es?	10
Representación por niveles	10
SCAMPI (método de evaluación)	11
ITIL	11
¿Qué es?	11
ISO 15504 (SPICE)	11
¿Qué es?	11
The Applicability of ISO/IEC 25023 Measures in the Integration of Agents and Automation Systems (Paper)	11
THE THREE ASPECTS OF SOFTWARE QUALITY: FUNCTIONAL, STRUCTURAL, AND PROCESS (PAPER)	12
When good enough software is best (PAPER)	12
Cierta cantidad de fallas no críticas es aceptable (umbral de fallas no críticas por unidad de testing)	12
Unidad 3 - Software Engineering Approaches	13
¿Qué es un proyecto?	13
¿Qué es “Project Management”?	13
Dimensiones de un proyecto de software	13
Roles de un proyecto	14
CYNEFIN	14
¿Qué es?	14
Kanban	14
¿Qué es?	14

Conceptos básicos	15
Limitar el WIP	15
SCRUM	15
¿Qué es?	15
Artefactos	16
Roles	16
Ceremonias	16
LEAN	17
¿Qué es?	17
Principios del modelo LEAN	18
La Guía de Scrum (Paper)	18
A Leader's Framework for Decision Making (Paper)	19
Cynefin(alternativa) (Paper)	19
Unidad 4 - Estimaciones de Software	20
Proceso	20
¿Por qué fallan las estimaciones?	20
Nivel de incertidumbre	20
Ley de Parkinson	21
Ciclo de estimación	21
Métodos de Estimación	22
TIMEBOX Development (Paper)	24
Stop Promising Miracles (Paper)	24
What is Planning Poker in Agile? (Paper)	24
Fundamentals of Function Point Analysis (Paper)	25
Comparing Effort Estimates Based on Use Case Points with Expert Estimates (Paper)	25
Unidad 5: Software Configuration Management	26
Configuración	26
Ítem de Configuración	26
¿Qué es la gestión de la configuración de software (SCM)?	26
Pasos de SCM	26
Software Configuration Management de acuerdo a SWEBOK	28
Release Management	28
Tipos de Builds	28
Condiciones para deployar	29
Integración Continua	29
Continuous Delivery	29
Continuous Deployment	29
Herramientas para cada parte del proceso:	30
SOFTWARE CONFIGURATION MANAGEMENT	30
But I Only Changed One Line of Code!	30
Unidad 6: Software Testing	31
¿Qué es?	31
¿Qué es la crisis del Software?	31

Objetivo del testing	31
Motivo de las fallas	31
¿Qué es Software de Calidad?	32
Verificación vs Validación	32
Proceso de testing	32
Falla vs Incidente	33
¿Cuándo parar de probar?	34
Abaratamiento del Testing	34
Enfoque de Caja Negra	35
Enfoque de Caja Blanca	35
Complejidad Ciclomática	36
Tipos de prueba	36
Prueba Unitaria	37
Prueba de integración	37
Prueba de Aceptación de Usuario	37
Prueba de Stress	37
Pen Testing / Ethical Hacking	38
Prueba de Volumen	38
Prueba de regresión	38
Prueba Alfa y Beta	38
Clasificación de las pruebas	39
Pirámide del Testing	39
Modelo de ciclo de vida V	40
Cuadrantes, niveles y tipos de testing	40
Agile testing	40
Test driven development	41
WhenTwoEyesAreNotEnough(KarlWiegers) (Paper)	42
Using Heuristic Test Oracles (Paper)	43
Exploratory Testing Explained	44
Unidad 7: Software Metrics	45
Conceptos	45
GQM - Goal Question Metric	45
Clasificación	45
Métricas de Proceso	47
Métricas de Software (Producto)	47
Métricas de Recursos	47

Unidad 1 - Introducción

¿Qué es la Ingeniería de Software?

La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software.

Body of Knowledge (BoK)

Un “cuerpo de conocimiento” es un requisito para calificar a cualquier área de la ingeniería como una profesión y describe el conocimiento relevante para una disciplina.

SWEBoK (Software Engineering Body of Knowledge)

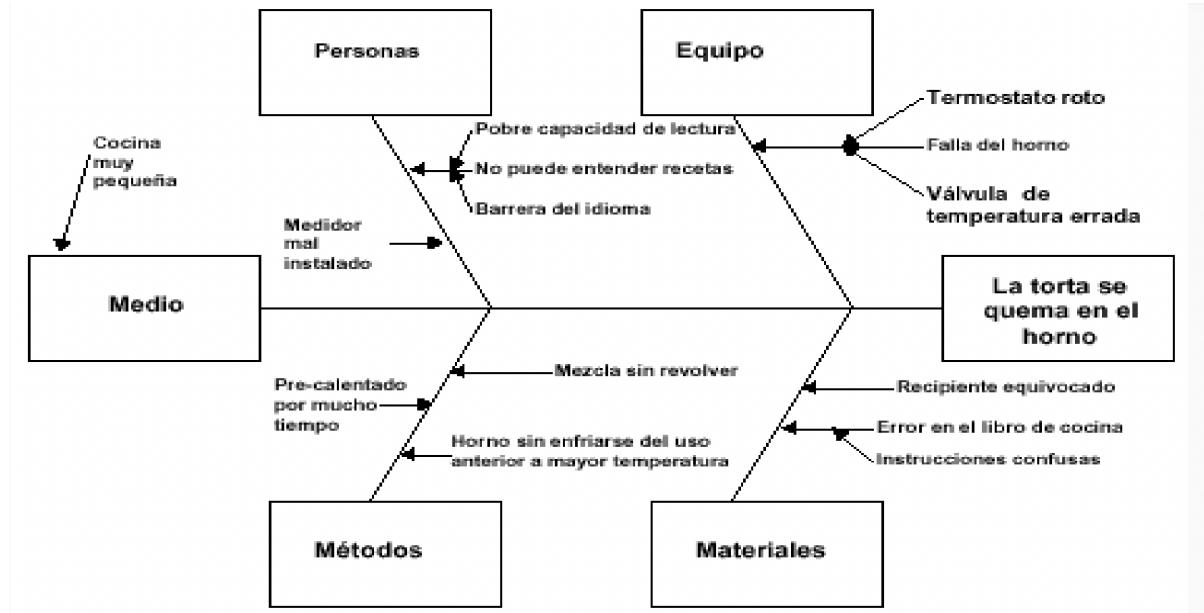
Está conformado por 15 Áreas de Conocimiento (Knowledge Areas – KA) que sintetizan conceptos básicos y referencian a información más detallada.

The 15 SWEBOK KAs

1. Software Requirements
2. Software Design
3. Software Construction
4. Software Testing
5. Software Maintenance
6. Software Configuration Management
7. Software Engineering Management
8. Software Engineering Process
9. Software Engineering Models and Methods
10. Software Quality
11. Software Engineering Professional Practice
12. Software Engineering Economics
13. Computing Foundations
14. Mathematical Foundations
15. Engineering Foundations

Diagrama Causa-Efecto (Diag. De Ishikawa o Espina de Pescado)

Es la representación de varios elementos (causas) de un sistema que pueden contribuir a un problema (efecto)



Classic Mistake (paper)

¿Qué son los classic mistakes?

Son errores que se cometen tan seguido, por tanta gente, que las consecuencias de cometer estos errores deberían ser predecibles y los mismos errores deberían ser evitables.

Clasificaciones

- Errores que ocurren por frecuencia (Most Frequent Classic Mistakes)
 - Overly optimistic schedules
 - Unrealistic expectations
 - Excessive multi-tasking
 - Shortchanged quality assurance
 - Noisy, crowded offices
- Errores que ocurren por impacto (Most Severe Classic Mistakes)
 - Unrealistic expectations
 - Weak personnel
 - Overly optimistic schedules
 - Wishful thinking
 - Shortchanged quality assurance
- Errores que ocurren por nivel de riesgo [Frequency * Severity = Mistake Exposure Index (MEI)] (Most Damaging Classic Mistakes Overall)
 - Unrealistic expectations
 - Overly optimistic schedules
 - Shortchanged quality assurance
 - Wishful thinking
 - Confusing estimates with targets

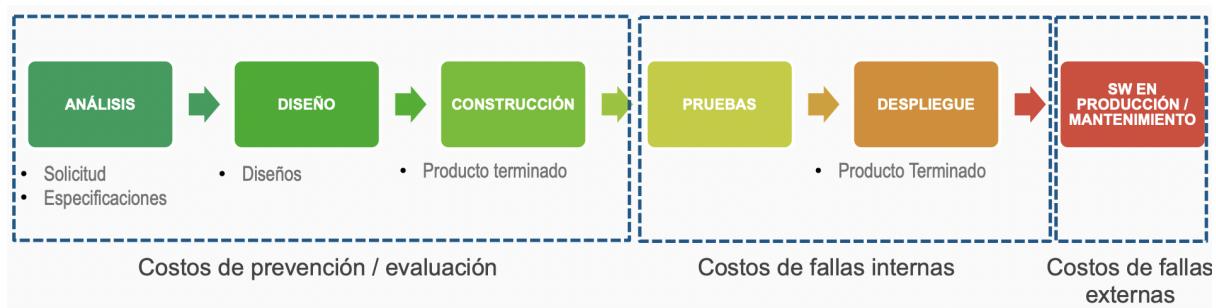
*Los ejemplos corresponden al top 5 en su medida

Unidad 2 - Calidad de Software

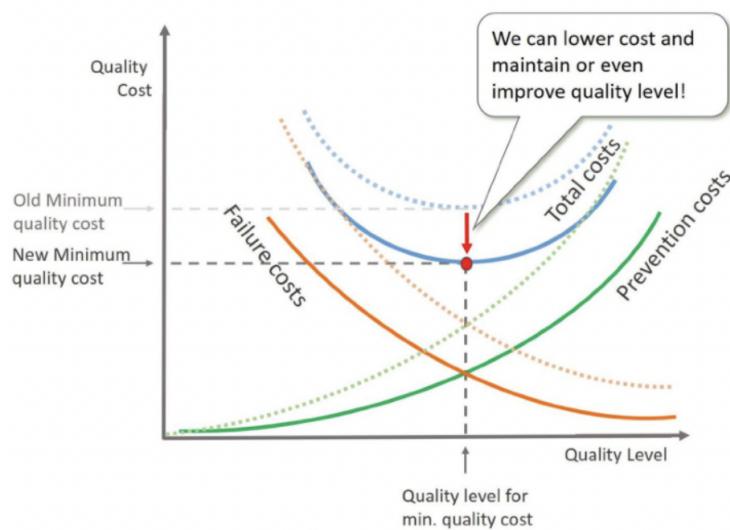
¿Qué es la Calidad?

- Cumplir con los requerimientos (requisitos)
- Cumplir con los requerimientos de alguna persona
 - Calidad es valor para alguna persona
 - Valor es aquello que está dispuesto a pagar para obtener sus requerimientos
- Adecuación al uso
 - Satisfacción de las necesidades del cliente
 - Ausencia de deficiencias
- ISO: La totalidad de aspectos y características de un producto o servicio que se sustentan en su capacidad de cumplir las necesidades especificadas o implícitas

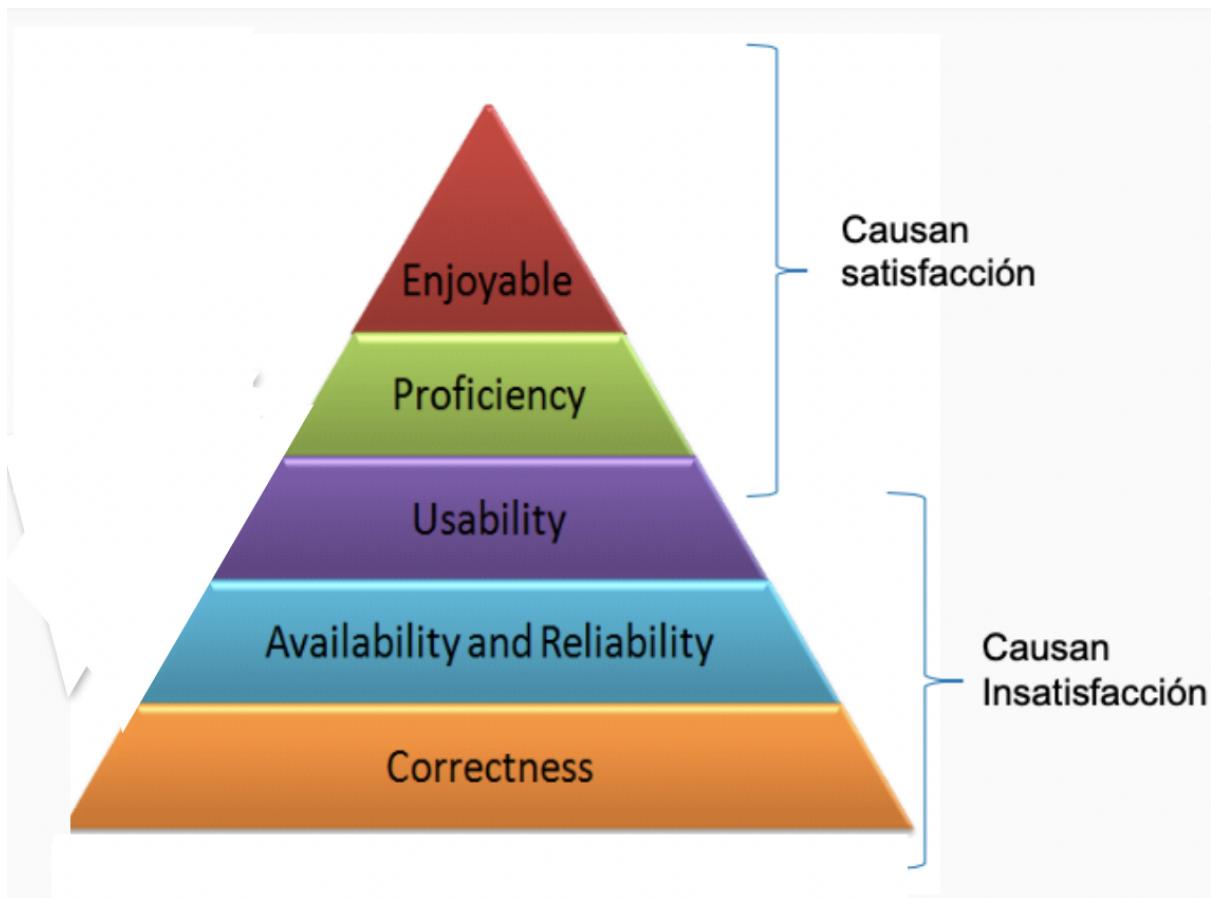
Costos de no calidad



- Baja motivación de los equipos de trabajo / Duplicación de esfuerzos
- Over-time constante / Re-trabajo constante (Mayor costo \$\$)
- Desgaste del equipo de trabajo
- Imagen negativa ante el cliente



Pirámide de jerarquía de factores de calidad



Desde la base hasta una parte de la usabilidad son parámetros que deberían estar cubiertos de mínima para un software productivo, la parte de satisfacción sería un adicional que tiene que ver más con la apreciación del usuario en la utilización del software

Visiones de calidad

- **VISIÓN TRASCENDENTAL:** La calidad es algo que se puede reconocer pero no se puede definir. Es una mirada plenamente subjetiva sobre uno o más aspectos de lo que estamos evaluando
- **VISIÓN DEL USUARIO:** La calidad es adecuación al propósito
- **VISIÓN DE LA MANUFACTURA:** La calidad es conformidad con la especificación
- **VISIÓN DEL PRODUCTO:** La calidad está vinculada a las características inherentes del producto
- **VISIÓN BASADA EN EL VALOR:** La calidad depende de la cantidad de dinero que el usuario está dispuesto a pagar por el producto

ISO 25010 – Calidad del producto de software



- **Adecuación Funcional:** Representa la capacidad del producto software para proporcionar funciones que satisfacen las necesidades declaradas e implícitas, cuando el producto se usa en las condiciones especificadas.
- **Eficiencia de Desempeño:** Esta característica representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones.
- **Compatibilidad:** Capacidad de dos o más sistemas o componentes para intercambiar información y/o llevar a cabo sus funciones requeridas cuando comparten el mismo entorno hardware o software.
- **Usabilidad:** Capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones.
- **Fiabilidad:** Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados.
- **Seguridad:** Capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no puedan leerlos o modificarlos
- **Mantenibilidad:** Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas.
- **Portabilidad:** Capacidad del producto o componente de ser transferido de forma efectiva y eficiente de un entorno hardware, software, operacional o de utilización a otro.

Modelos de Calidad de Proceso

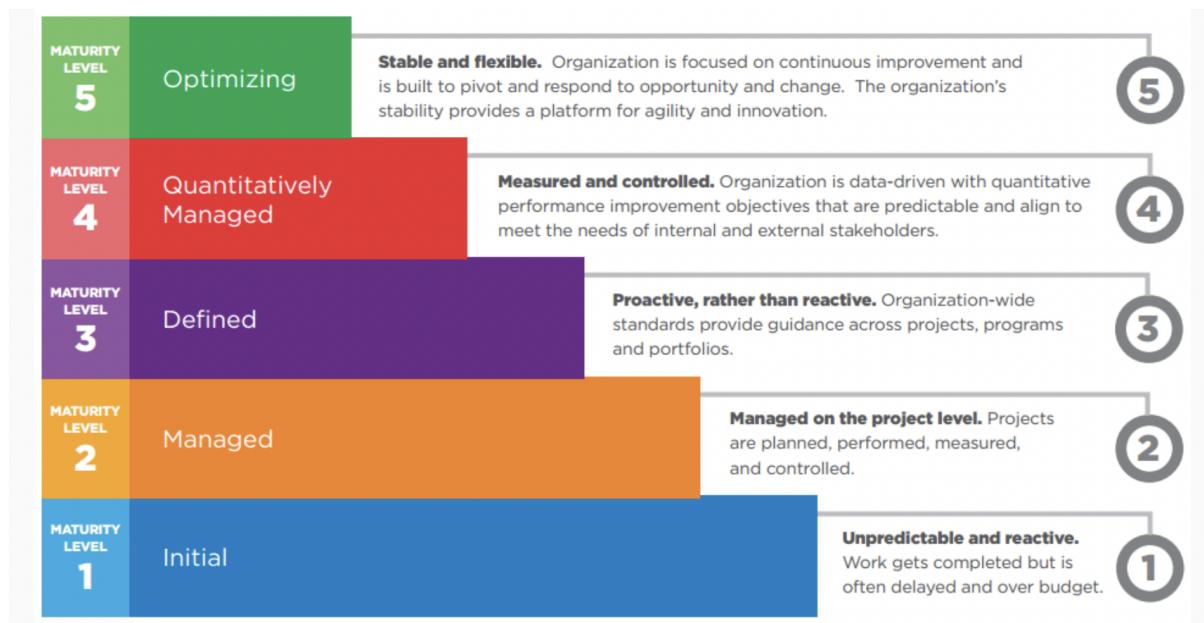
CMMI (Capability Maturity Model Integrated)

¿Qué es?

Modelo para la mejora y evaluación de los procesos de desarrollo, mantenimiento y operación de SW. Determina la madurez de un proceso y organiza el esfuerzo para mejorarlo describiendo un camino incremental de mejora

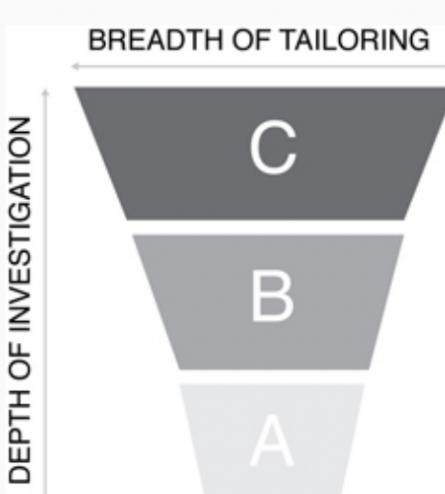


Representación por niveles



SCAMPI (método de evaluación)

Characteristic	Class A	Class B	Class C
Amount of objective evidence	High	Medium	Low
Ratings generated	Yes	No	No
Resource needs	High	Medium	Low
Team size	Large	Medium	Small



ITIL

¿Qué es?

Conjunto de buenas prácticas utilizadas para la gestión de servicios de tecnologías de información

ISO 15504 (SPICE)

¿Qué es?

Estándar internacional de evaluación y determinación de la capacidad y mejora continua de procesos de ingeniería de SW

The Applicability of ISO/IEC 25023 Measures in the Integration of Agents and Automation Systems (Paper)

No resumido, no parece aportar nada significativo

THE THREE ASPECTS OF SOFTWARE QUALITY: FUNCTIONAL, STRUCTURAL, AND PROCESS (PAPER)

No resumido, no parece aportar nada significativo

When good enough software is best (PAPER)

El Software Manager debe saber todos los parámetros cruciales (costo, funcionalidad, el staff, la calidad, etc.) pero el cliente es el que determina el balance apropiado entre ellos.

Es más importante entregar el producto y sacrificar calidad que entregarlo tarde con la calidad prometida

El movimiento de las variables de compromiso (tiempo calidad MO) no debe ser considerado lineal

Los tradeoffs que hay que realizar y como hacer entender al cliente sobre estos.

El intentar poder debatir sobre 5 ejes fundamentales con el cliente: costos, planif, errores, tiempos y equipo

Que le de tanta importancia a la velocidad de entrega a pesar de que pueda llegar a ser con muchos errores

Cierta cantidad de fallas no críticas es aceptable (umbral de fallas no críticas por unidad de testing)

Unidad 3 - Software Engineering Approaches

¿Qué es un proyecto?

Es un esfuerzo temporal emprendido para crear un producto o servicio único para lograr un objetivo.

- Tiene un objetivo o beneficio a obtener que guía el proyecto
- Temporal: porque tiene principio y fin
- Único: No es recurrente, cada proyecto es diferente al otro
- Posee Recursos limitados
- Consta de una sucesión de actividades o fases en las que se coordinan los distintos recursos

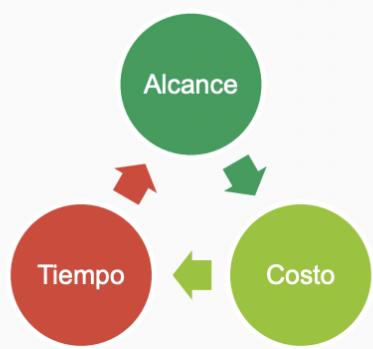
¿Qué es “Project Management”?

La administración de proyectos es una disciplina que consiste en planificar, organizar, obtener y controlar recursos, utilizando herramientas y técnicas para lograr que el proyecto logre sus objetivos en tiempo y forma.

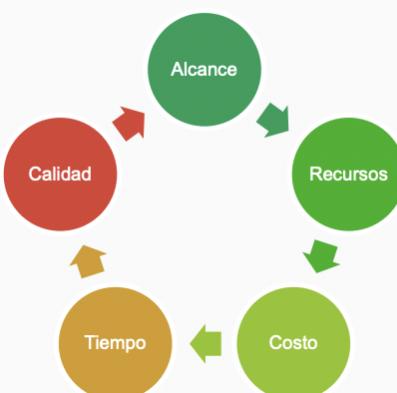
Dimensiones de un proyecto de software

- No se puede cumplir con todas a la vez !!
- Cada dimensión puede ser:
 - Driver: objetivo vital a lograr, tiene poco/nada de flexibilidad.
 - Restricción: no está bajo nuestro control directo y también tiene poco/nada de flexibilidad.
 - Grado de Libertad: Libertad para fijar objetivos. Existe Flexibilidad para manejar esta variable

Enfoque de las 3 dimensiones
(triple constraint)



Enfoque de las 5 dimensiones



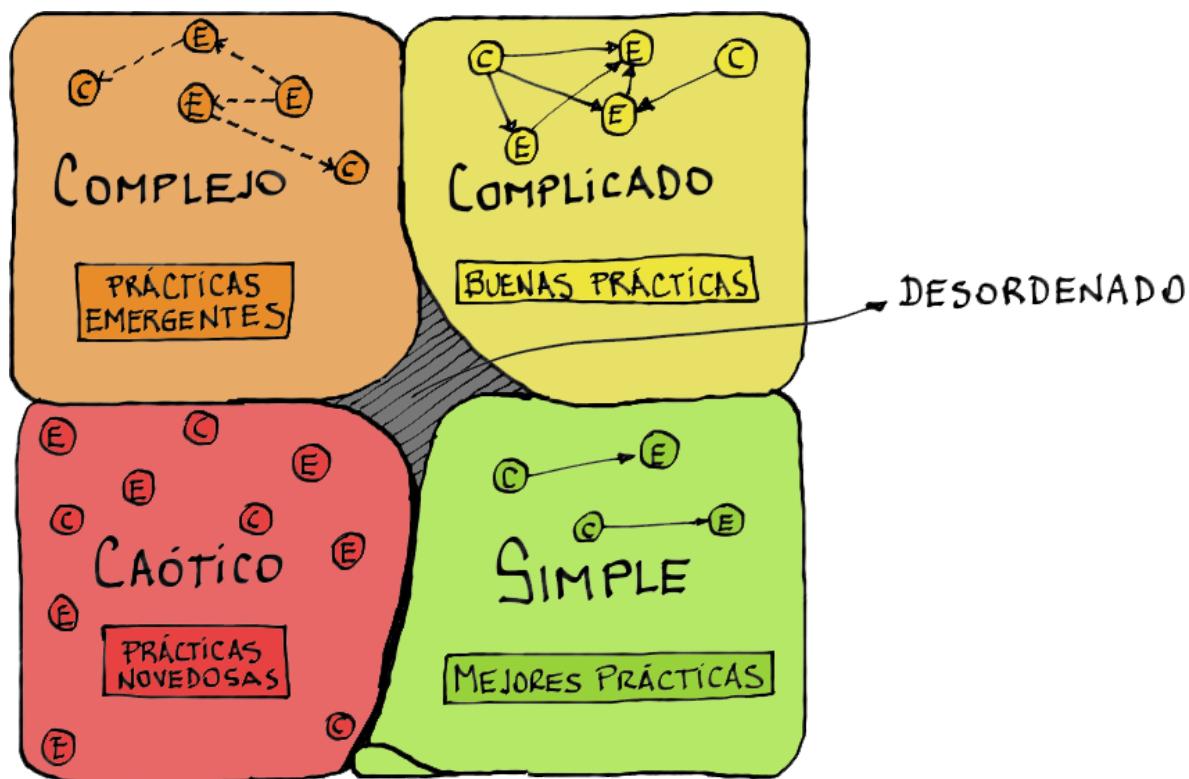
Roles de un proyecto

- Stakeholders
- Sponsor
- Usuario Campeón
- Usuarios directos
- Usuarios indirectos

CYNEFIN

¿Qué es?

Modelo que compara características de cinco dominios de complejidad diferentes: simple, complicado, complejo, caótico y desordenado.



Scrum es mejor en un ambiente complejo.

Kanban

¿Qué es?

Kanban es una Administración Visual de Flujo de Trabajo que ayuda a realizar más con menos estrés.

Conceptos básicos

- ✓ Visualizar el flujo de trabajo
- ✓ Limitar el trabajo en progreso, es decir, cuánto trabajo podes hacer en forma simultánea
- ✓ Realizar mediciones y optimizar el flujo



Limitar el WIP

Consiste en acordar la cantidad de ítems que pueden trabajarse en paralelo por cada etapa del proceso.

El principal objetivo es detectar cuellos de botella que representan estancamiento/bloqueos para avanzar.

La práctica de limitar el WIP es lo que marca la diferencia entre tener una “lista visual de to-do” y el tener un sistema Kanban

¿Cuándo uso Kanban?

- Los proyectos sean inestables, tengamos muchos cambios
- Los requerimientos estén cambiando constantemente de prioridad
- En entornos de resolución de incidencias

SCRUM

¿Qué es?

Marco de trabajo para la administración de proyectos que enfatiza en el trabajo en equipo, en el proceso iterativo hacia un objetivo bien definido. Comienza con una simple premisa: comenzar con lo que puede ser visto o conocido. Luego, seguir el avance y modificar lo necesario. Scrum es un marco metodológico pensado para construir productos de forma incremental, en una serie de periodos de tiempo llamados Sprints.

Un Sprint es un período fijo de tiempo (1 a 4 semanas). En cada Sprint el equipo Scrum construirá y entregará un Incremento de Producto. Cada incremento es una versión mejorada del producto que alcanza criterios de aceptación y el nivel de calidad requerido.

Artefactos

Product Backlog: Backlog del proyecto, contiene todas las tareas que hay que hacer para finalizar el producto.

Sprint backlog: Backlog del sprint que se está ejecutando, contiene todas las tareas que se seleccionaron para ser realizadas en ese Sprint

Roles

Scrum Master:

- Dueño del proceso
- Elimina trabas que tenga el team
- Vela que se aplique bien scrum

Product Owner:

- Es el cliente
- Owner del backlog, define que se prioriza
- Gestiona el plan de entregas

Equipo:

- El team de desarrollo
- No deben ser equipos grandes
- Transforma el backlog en funcionalidades incrementales
- Son interdisciplinarios e autoorganizados

Ceremonias

Daily meeting:

- Duran 15 minutos max
- Se dice que se hizo, que se va a hacer y que trabas tenemos
- Se busca hacer bien temprano
- Luego de la reunión, se busca resolver las trabas mencionadas

Sprint Planning

- Se realiza al inicio de cada sprint
- Se arma el sprint backlog
- Realizar y revisar estimaciones (Planning Poker)
- Asignación de user stories
- Toma aprox. un día entero, dependiendo el tamaño del sprint puede ser menos

Sprint Review

- Se realiza al finalizar cada sprint
- Se muestra lo desarrollado (funcionando) en el sprint
- Se obtiene feedback del PO
- Toma aprox. medio día, dependiendo el tamaño del sprint puede ser menos

Retrospectiva

- Se realiza al final de cada sprint
- Se nombra lo que funcionó bien y lo que no
- Se toman compromisos para mejorar lo que no funcionó, para implementar en futuras iteraciones
- Se toma lo que funcionó bien para seguir haciéndolo de esa manera

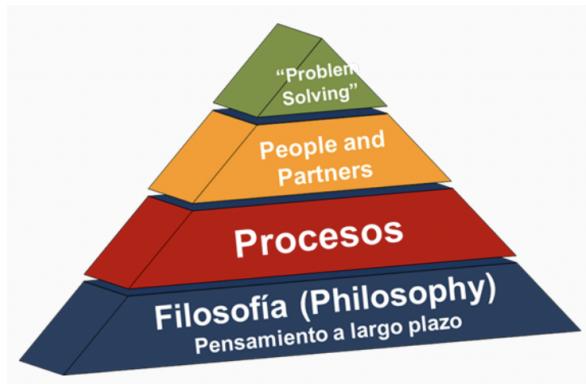
LEAN

¿Qué es?

Lean es una filosofía y un enfoque que hace hincapié en la eliminación de residuos o de no valor añadido al trabajo, a través de un enfoque en la mejora continua para agilizar las operaciones. Lean se centra en ofrecer una mayor calidad, reducir el tiempo de ciclo y reducir los costos.

El Modelo de gestión LEAN busca guiar y reforzar los principios LEAN de manera efectiva en toda la organización a través del uso de distintas herramientas y sistemas con sus rutinas de trabajo generando los comportamientos necesarios que conduzcan a los resultados deseados creando así una cultura de MEJORA CONTINUA de creación de VALOR al CLIENTE y desarrollo de las PERSONAS.

Principios del modelo LEAN



1. Gestionar basado en una filosofía (propósito) de largo plazo
2. Crear flujos de procesos continuos para evidenciar problemas
3. Usar sistemas de procesos "PULL" para evitar la sobreproducción
4. Nivelar la carga de trabajo: HEIJUNKA
5. Generar una práctica y cultura de "parar" para resolver problemas y asegurar la calidad
6. Estandarizar; base fundacional de la mejora continua y el empoderamiento de las personas
7. Utilizar controles visuales para evidenciar los problemas
8. Recurrir únicamente a tecnología confiable y correctamente testeada para servir a las personas y a los procesos
9. Desarrollar líderes que entiendan el trabajo, viven la filosofía y se la enseñen a otros
10. Desarrollar personas y equipos excepcionales que sigan la filosofía de su organización.
11. Respetar a tu red de contratistas y proveedores desafiándolos y ayudándolos a mejorar
12. Ve y observa por ti mismo para comprender profundamente la situación (GENCHI GENBUTSU)
13. Tomar decisiones por consenso, teniendo en cuenta todas las opciones (NEMAWASHI), e implementar ágilmente
14. Propiciar el proceso de convertirse en una organización que aprende a través de la reflexión (HANSEI) y la mejora continua (KAIZEN)

La Guía de Scrum (Paper)

No lo lei, imagino que explica qué es SCRUM

A Leader's Framework for Decision Making (Paper)

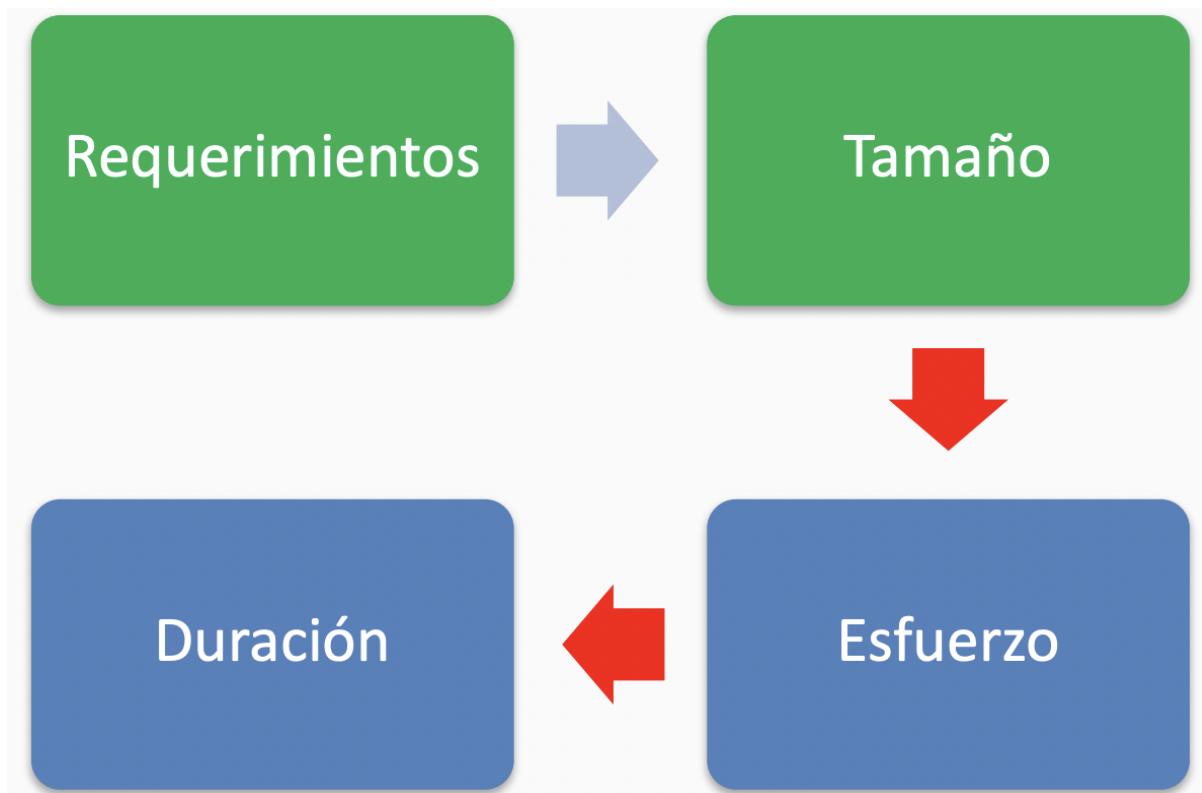
Es CYNEFIN, no aporta mucho más a lo visto antes.

Cynefin(alternativa) (Paper)

Es CYNEFIN, no aporta mucho más a lo visto antes.

Unidad 4 - Estimaciones de Software

Proceso



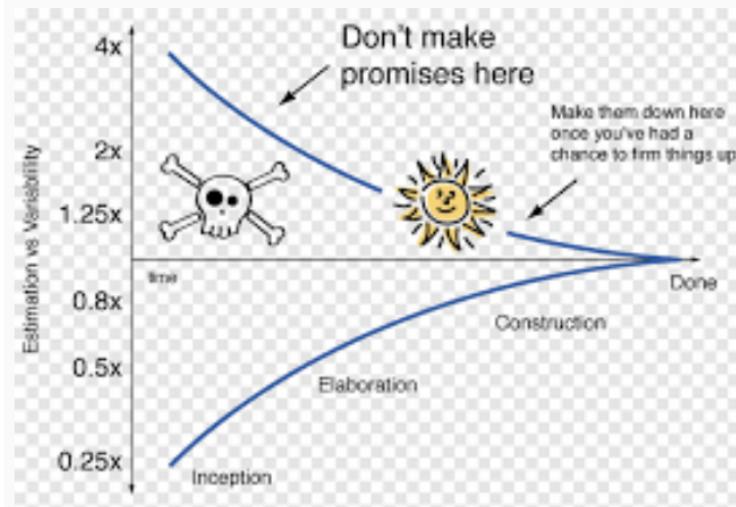
¿Por qué fallan las estimaciones?

- Optimismo
- Estimaciones informales (“estomacales”)
- No hay historia
- Mala definición del alcance
- Novedad / Falta de Experiencia
- Complejidad del problema a resolver
- No se estima el tamaño
- Porque la estimación fue buena pero cuando empieza el proyecto:
 - Mala administración de los requerimientos
 - No hay seguimiento y control
 - Se confunde progreso con esfuerzo

Nivel de incertidumbre

El “cono de la incertidumbre” define niveles estadísticos predecibles de la incertidumbre de las estimaciones en cada etapa del proyecto

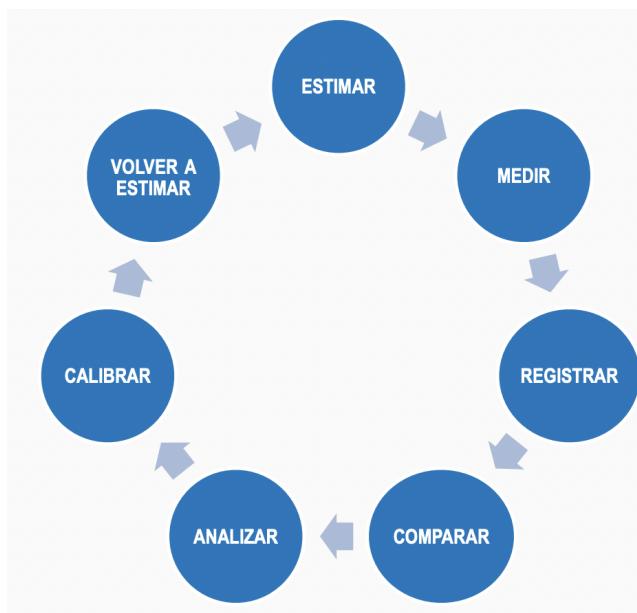
Cuento más refinada la definición, mas exacta será la estimación



Ley de Parkinson

“Toda tarea se expande hasta ocupar el tiempo que tiene asignado”

Ciclo de estimación



Métodos de Estimación

- Métodos no paramétricos
 - Juicio Experto
 - Depende de la persona que haga la estimación
 - Se basa en la experiencia personal
 - Por lo general se recurre a analogías
 - Pert (Program Evaluation and Review Technique)
 - Se basa en el juicio experto
 - Incluye los factores optimistas y pesimistas en su estimación
 - Estimación = $(\text{Optimista} + 4 \times \text{Medio} + \text{Pesimista}) / 6$
 - Se puede usar para estimar tamaño y esfuerzo
 - También se conoce como el método Clark
 - Wideband Delphi
 - Es el juicio experto en grupos
 - Da lugar a la participación de todos los involucrados
 - Ventajas: fácil de implementar y da sentido de propiedad sobre la estimación
 - Se puede utilizar en etapas tempranas del proyecto
 - Se recomienda utilizarlo en proyectos poco conocidos en donde no se cuenta con historia
 - Planning Poker
 - Podría no ser bueno para etapas tempranas por falta de historia sobre lo que hay que estimar pero es debatible. Sin duda es mejor en iteraciones posteriores, a medida que se conozca más al equipo y al producto sobre el que trata el proyecto más exacta es.
 - Se utiliza en etapas tempranas del proyecto

- Se estiman tamaños de tareas relativos entre sí.
- Se define una tarea pivote, y se le asigna un tamaño en story points que sigan una escala (ejemplo, potencias de 2, fibonacci, etc.)
- Teniendo en cuenta esa tarea, se le asigna un tamaño a las demás
- Ante cada estimación, presentan sus motivos los extremos y todos vuelven a estimar por si cambian de opinión

- Métodos paramétricos
 - Function Points
 - Es el método de mayor sensibilidad
 - Miden el tamaño del SW en base a la funcionalidad capturada en los requerimientos
 - Usa el modelo lógico o conceptual para trabajar
 - En general son aceptados en la industria a pesar de su complejidad y antigüedad
 - Hay muchas heurísticas en el mercado basadas en PFs
 - Los PFs son independientes de la tecnología
 - Requieren un análisis de requerimientos avanzado
 - Elementos del sistema:
 - Archivos lógicos internos (ALI o ILF)
 - Archivo de interface externa (AIE o EIF)
 - Entradas externas (EE o EI)
 - Salidas externas (SE o EO)
 - Consultas Externas (CE o EQ)
 - Hay 14 factores de complejidad técnica. Cada uno se evalua de acuerdo al grado de influencia (0-5).
 - Use Case Points
 - Tiene en cuenta factores externos de entorno
 - El número de Use Case Points depende de :
 - Cantidad y complejidad de los casos de uso
 - Cantidad y complejidad de los actores intervenientes en el sistema
 - Factores técnicos y ambientales
 - El método requiere que sea posible contar el número de transacciones en cada caso de uso. Una transacción es un evento que ocurre entre el actor y el sistema.
 - Basado en el método de Function Points
 - Elementos del sistema
 - Casos de Uso
 - Simple: 3 o menos transacciones.
 - Medio: 4 a 7 transacciones
 - Complejo: Más de 7 transacciones
 - Transacciones de Casos de Uso
 - Actores
 - Actores Simples: Son sistemas externos, son muy predecibles, todo sistema con interfaz de aplicación bien definida.

- Actores Promedio: Dispositivos de Hardware.
Requieren más esfuerzo controlarlos, son más propensos a errores. Personas interactuando a través de una interfaz basada en texto
- Actores Complejos: Son humanos, son impredecibles y difíciles de controlar. Personas que interactúan a través de una GUI
- Objects Points
 - Los “objetos” contemplan pantallas, reportes y módulos
 - No está relacionado necesariamente con objetos de OOP
 - El método original contempla solo estos tres tipos de objetos
 - Implementaciones ad-hoc del método consideran otros tipos de componentes como (stored procedures, clases, scripts SQL, etc ...)
 - Se asigna a cada componente un peso de acuerdo a su clasificación por complejidad
 - Considera como factor de ajuste el porcentaje de reuso de código
 - Cada objeto es clasificado de acuerdo a su nivel de complejidad en:
 - Simple
 - Medio
 - Difícil
 - Luego se le brinda un peso a cada nivel de complejidad (teniendo directa proporción con el esfuerzo que requiere la implementación de c/u)
 - Finalmente sumando la cantidad de objetos de cada complejidad con sus respectivos pesos nos da como resultado los OPs

TIMEBOX Development (Paper)

Este paper trata de un framework ó metodología para llevar adelante proyectos, sólo es aplicable por períodos cortos de tiempo de 1 a 3 meses. Se enfoca en mantener al equipo motivado y con el sentido de urgencia adecuado para poder cerrar un MVP o realizar la última parte de un proyecto más largo. En general una vez concluído con el TBD se premia al equipo por la dedicación, se otorga algún tipo de recompensa

Stop Promising Miracles (Paper)

No resumido

What is Planning Poker in Agile? (Paper)

No resumido, un poco ya está explicado arriba

Fundamentals of Function Point Analysis (Paper)

No resumido, un poco esta explicado arriba FP

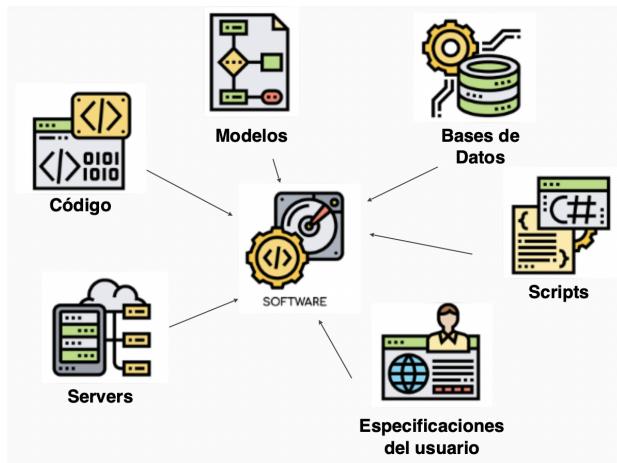
Comparing Effort Estimates Based on Use Case Points with Expert Estimates (Paper)

No resumido

Unidad 5: Software Configuration Management

Configuración

Es el conjunto de todos los componentes fuentes que son compilados en un ejecutable consistente. Todos los componentes, documentos e información de su estructura que definen una versión determinada del producto a entregar



Ítem de Configuración

Es cualquier elemento involucrado en el desarrollo del producto y que está bajo el control de la gestión de configuración. Para cada ítem se conoce información sobre su configuración (nombre, versión, fecha de creación, autor, entre otros)

¿Qué es la gestión de la configuración de software (SCM)?

Es una disciplina orientada a administrar la evolución de:

- Productos
- Procesos
- Ambientes

Su propósito es establecer y mantener la integridad de los productos del proyecto de software a lo largo del ciclo de vida del mismo

Involucra para una configuración:

- Identificarla en un momento dado
- Controlar cambios sistemáticamente
- Mantener su integridad y origen

Acompaña la actividad de cambio con actividades de control

Pasos de SCM

1. Identificación de la configuración

- Definir qué elementos (ICs) estarán controlados por la gestión de configuración
 - No todo necesita estar bajo SCM
 - Es necesario contar o definir con guías para qué elementos son parte o no.
- Define momentos o condiciones para establecer una línea de base o bien para liberarla (es decir llevarla a otro ambiente, también conocido como “release”)

2. Control de la configuración

- Asegurar que los ítems de configuración mantienen su integridad ante los cambios a través de:
 - La identificación del propósito del cambio
 - La evaluación del impacto y aprobación del cambio
 - La planificación de la incorporación del cambio
 - El control de la implementación del cambio y su verificación

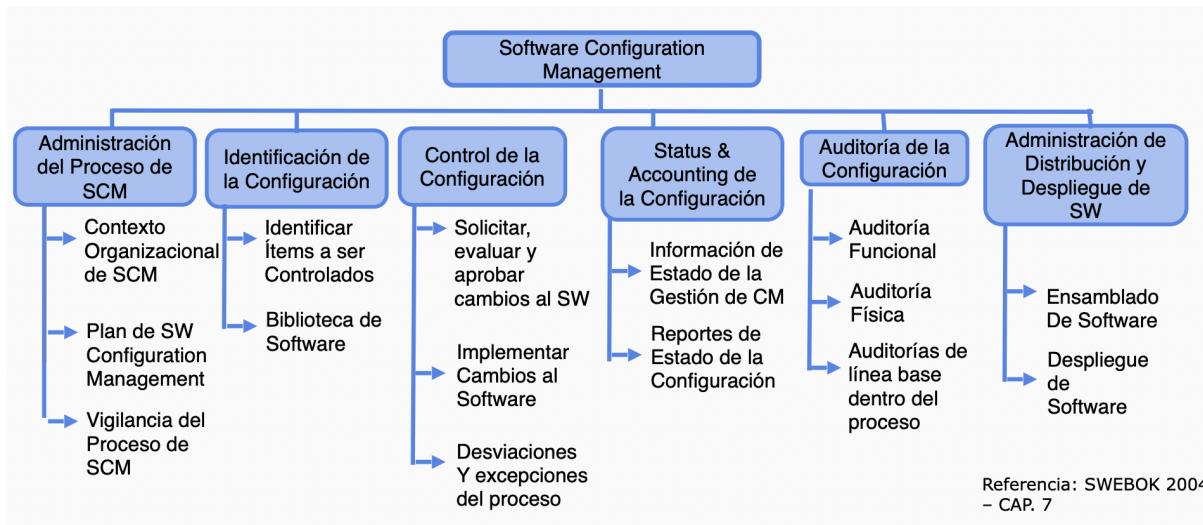
3. Status & Accounting de la configuración

- Registrar y reportar la información necesaria para administrar la configuración de manera efectiva
 - Listar los ICs aprobados
 - Mostrar el estado de los cambios que fueron aprobados
 - Reportar la trazabilidad de todos los cambios efectuados al baseline
- Debe poder contestar “¿Qué cambios se realizaron al sistema?”
 - Cuándo cambió?
 - Quién lo cambió?
 - Qué cambió?
 - Alcance del cambio
 - Quién aprobó el cambio?
 - Quién solicitó el cambio?
- Se debe informar periódicamente con reportes a todos los grupos afectados

4. Administración de Distribución y Despliegue de SW

- Asegurar la construcción exitosa del paquete de software, basada en los ICs requeridos para la funcionalidad a entregar, para luego liberarlo en forma controlada a otros entornos ya sea de pruebas, producción, usuario final, etc.
- Se divide en 2 partes:
 - Software Building
 - Release Management
- Además del ejecutable o paquete de software, comprende la administración, identificación y distribución de un producto.

Software Configuration Management de acuerdo a SWEBOk



Administración del proceso de SCM

- Es la etapa previa al arranque del proyecto, donde se debe comprender el contexto organizacional.
- Incluye las actividades de realización del Plan de SCM con sus distintas etapas:
 - Identificación de la configuración
 - Procesos de Control de cambios
 - Auditorías de software
 - Release management, entre otras
 - Políticas de manejo de ramificaciones o branches para que no crezcan

Tipos de Auditoría

- Tiene por objetivo realizar una verificación del estado de la configuración a fin de determinar si se están cumpliendo los requerimientos especificados
- Pueden ser:
 - Funcional, que verifica el cumplimiento de requerimientos
 - Física, que verifica la configuración del producto en cuanto a la estructura especificada
 - De Proceso, que verifica se haya cumplido el proceso de SCM

Release Management

Asegurar la construcción exitosa del paquete de software, basada en los ICs requeridos para la funcionalidad a entregar, para luego liberarlo en forma controlada a otros entornos ya sea de pruebas, producción, usuario final, etc.

Tipos de Builds

- Local builds: El developer lo realiza localmente en su entorno de desarrollo, corre Pruebas Unitarias (UT)

- Integration builds: su objetivo es generar el entorno completo para pruebas de integración
- Nightly builds: su objetivo es ejecutar la construcción en forma diaria y generar reportes con información sobre estabilidad, tiempo de build, etc.
- Release builds: Se disparan cuando bien un administrador decide crear una nueva versión a ser liberada, o por el mismo sistema de integración si se utiliza el modo de deployment contínuo

Condiciones para deployar

- Qué usuarios deberán recibir los cambios (geográfico, premium, por segmento)
- En qué forma se deberá hacer el despliegue
 - Entornos por los que deberá pasar (testing, staging, producción, otros)
 - Procesos de Roll Forward
 - Procesos de Rollback
 - Validación de despliegue correcto / incorrecto
- Riesgos del despliegue y cómo se minimizan
- Aprobación por el negocio y/o área de QA
- Quienes realizarán el deployment de la versión definida

Integración Continua

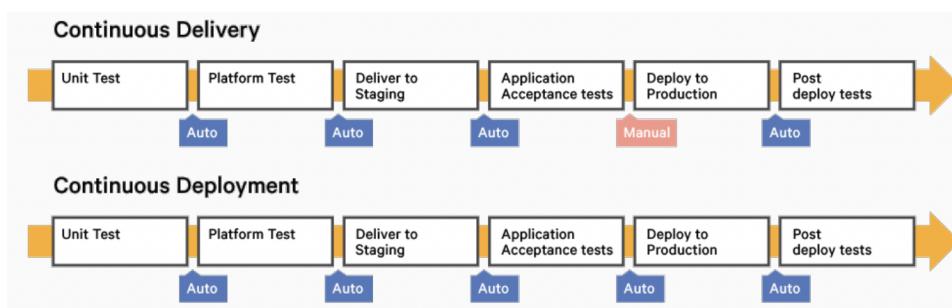
Proceso en el cual hay un repositorio único de código, donde se automatiza el proceso de buildeo y el build es testeable automáticamente. Cubre todo el proceso hasta la generación de los builds.

Continuous Delivery

Los builds llegan a ser deployados en ambientes de staging/test donde son probados, para finalmente dejar una versión disponible la cual puede ser deployada en producción de manera manual.

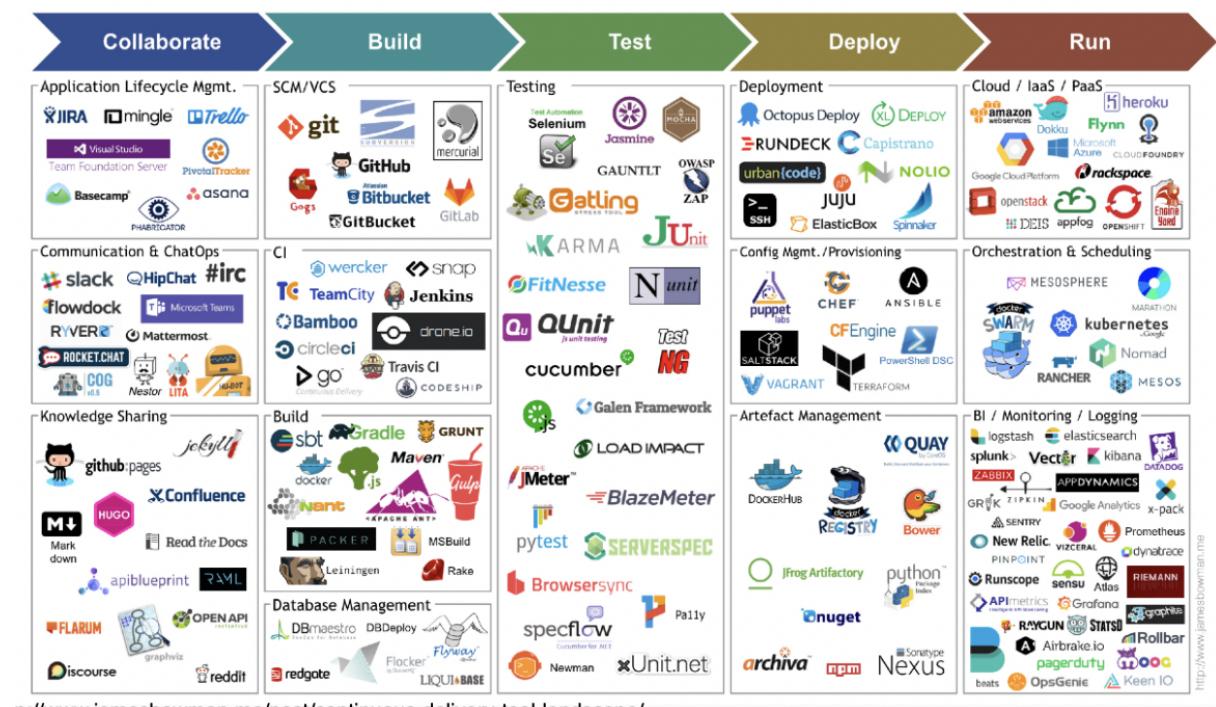
Continuous Deployment

Al proceso anterior, se le suma el deployado automático en producción.



Herramientas para cada parte del proceso:

Para usar de ejemplo:



SOFTWARE CONFIGURATION MANAGEMENT

No resumido

But I Only Changed One Line of Code!

Básicamente resume las etapas de SCM, y concluye que seguir este proceso reduce la posibilidad de que ocurran pequeños cambios con grandes impactos en el software.

Unidad 6: Software Testing

¿Qué es?

- Según la IEEE: Una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, los resultados de dicha ejecución son observados o registrados y, a partir de los mismos, se realiza una evaluación de algún aspecto del sistema o componente
- Software Testing es usualmente definida como una actividad para evaluar si los resultados obtenidos en la ejecución del software coinciden con los resultados esperados, con el propósito de determinar si el sistema está libre de defectos. Involucra la ejecución de un componente de software o de un sistema para evaluar una o varias propiedades de interés
- Las pruebas por sí solas encuentran fallas, NO defectos (a lo cual solo pueden demostrar su presencia pero nunca su ausencia)
- Partimos de la suposición que el SW a probar contiene defectos (lo cual casi siempre es válido) y se trata de encontrar tantos como sea posible
- Por el contrario, si nuestro objetivo es mostrar que el SW no contiene errores, estaremos inconscientemente orientados a ese fin
- El Testing es importante porque los bugs pueden ser costosos o incluso peligrosos. Los bugs de software pueden causar potencialmente problemas económicos o de pérdidas humanas.

¿Qué es la crisis del Software?

Concepto de los años 70, surgido por estas preguntas:

- ¿Por qué lleva tanto tiempo terminar los programas?
- ¿Por qué es tan elevado el coste?
- ¿Por qué no podemos encontrar todos los errores antes de entregar el software a nuestros clientes?
- ¿Por qué es tan difícil constatar el progreso durante el desarrollo?
- ¿Por qué es tan difícil calcular cuánto tiempo va a costar?

Por esto se creó el concepto de ingeniería de software y la Ingeniería de Requerimientos

Objetivo del testing

Básicamente, encontrar fallas en el producto

- De forma eficiente
 - Hacerlo lo más rápido posible
 - Hacerlo lo más barato posible
- De forma eficaz
 - Encontrar la mayor cantidad de fallas
 - No detectar fallas que en realidad no son
 - Encontrar las más importantes

Motivo de las fallas

- Requerimientos poco claros
- Requerimientos incorrectos
- Requerimientos cambiantes
- Diseño Complejo
- Desarrolladores sin experiencia
- Lógica Complicada
- Falta de tiempo
- Falta de conocimiento del producto
- Testers sin experiencia
- Problemas del entorno o de sistemas

¿Qué es Software de Calidad?

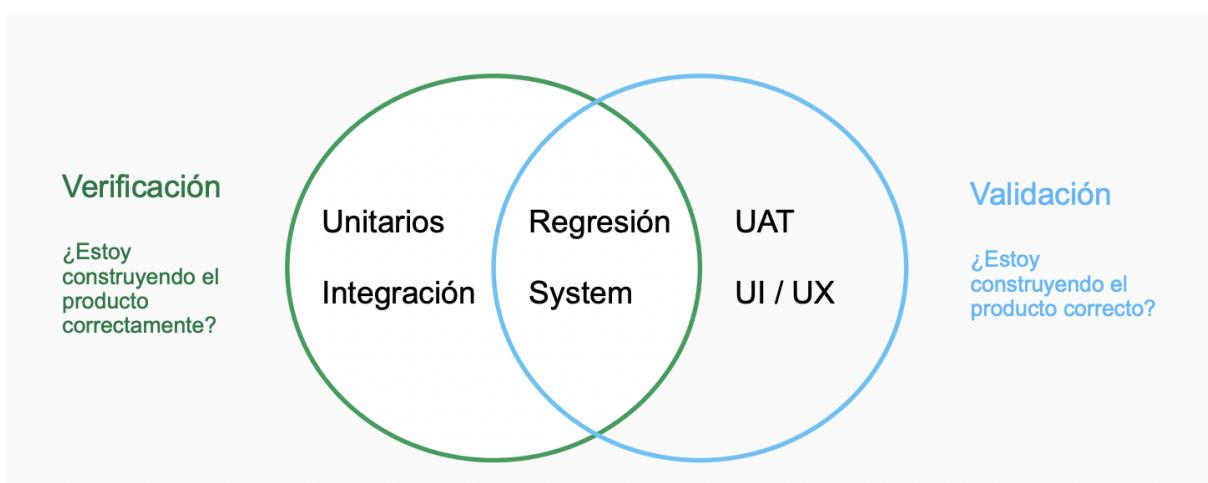
- Aquel que cumpla con los requisitos
- El que al ser usado NO tenga fallas

QA vs QC

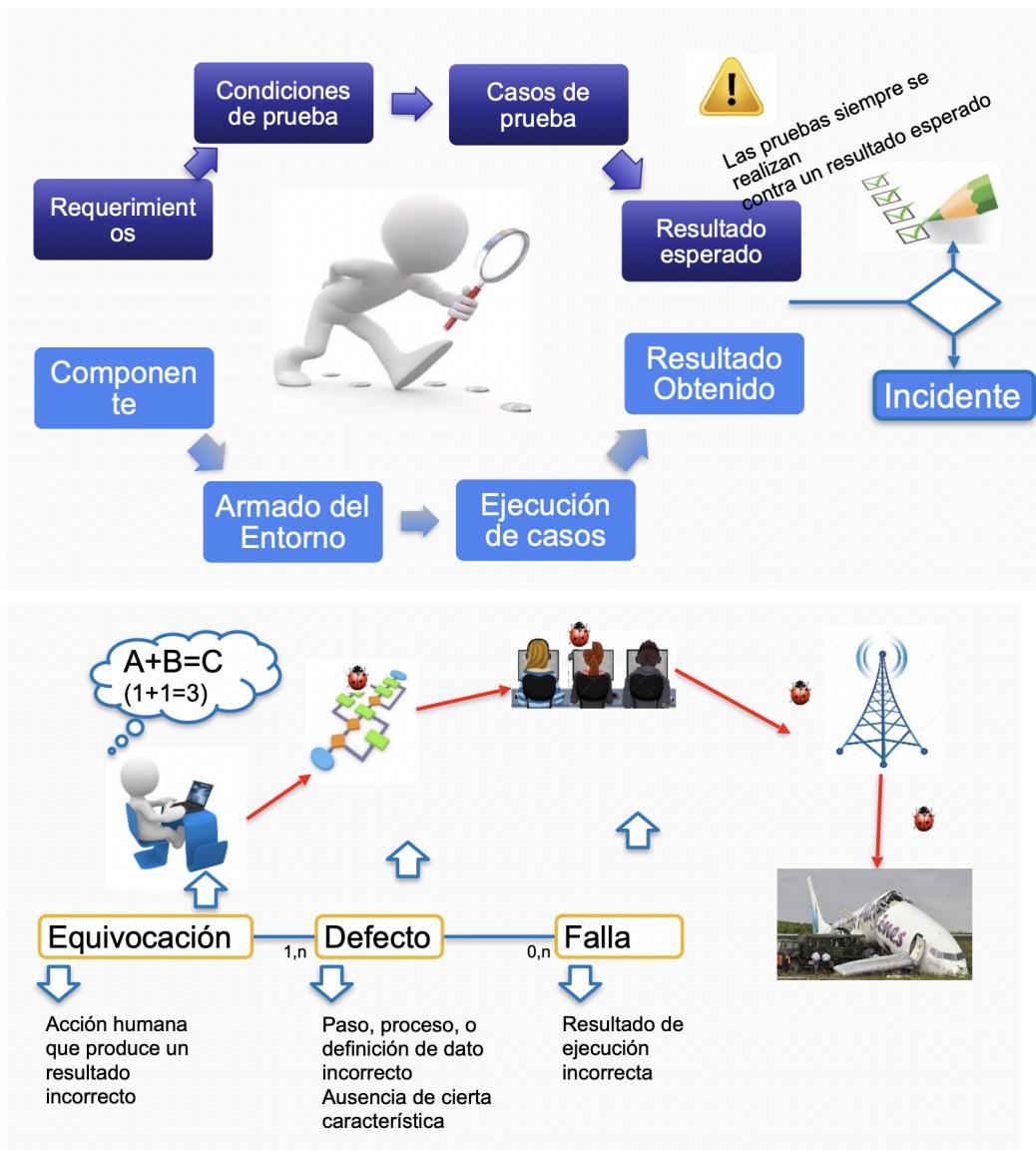
Quality Assurance refiere a todos los procesos que se llevan a cabo desde la gestación del proyecto hasta su finalización, contempla la planificación de test y builds automáticos, las consideraciones necesarias para realizar componentes más testeables en la etapa de diseño, las herramientas que se van a utilizar, etc.

Quality Control es la validación del producto final, una vez que el software se encuentra terminado los usuarios validan que se cumpla con todos los requerimientos y que el funcionamiento esté libre de fallas.

Verificación vs Validación



Proceso de testing



Falla vs Incidente

Toda ocurrencia de un evento que ocurre durante la ejecución de una prueba de software que requiere investigación

- Un incidente puede deberse a:
 - Un defecto en el SW
 - Un defecto en los casos de prueba
 - Un defecto en el ambiente
 - entre otros eventos...
- Entonces no todo incidente es una falla, por ejemplo
 - Defectos en los casos
 - Equivocaciones al ejecutar las pruebas
 - Interpretaciones erróneas
 - Dudas

Proceso de Depuración

- Depurar es eliminar un defecto que posee el SW
- La depuración NO es una tarea de prueba aunque es consecuencia de ella
- La prueba detecta el falla (efecto) de un defecto (causa)

En la “Depuración” debemos:

- DETECTAR
 - Dada la falla debemos hallar el defecto (dado el efecto debemos encontrar la causa)
- DEPURAR
 - Encontrado el defecto debemos eliminarlo
 - Debemos encontrar la razón del defecto
 - Debemos encontrar una solución
 - Debemos aplicarla
- VOLVER A PROBAR
 - Asegurar que sacamos el defecto
 - Asegurar que no hemos introducido otros (regresión)
- APRENDER PARA EL FUTURO
 - Lecciones Aprendidas

¿Cuándo parar de probar?

- Nuestro Problema
 - Imposibilidad en la práctica de estimar la intensidad del testing para alcanzar determinada densidad de defectos
- Estrategias
 - Pasa exitosamente el conjunto de pruebas diseñado
 - “Good Enough”: cierta cantidad de fallas no críticas es aceptable (umbral de fallas no críticas por unidad de testing)
 - Cantidad de fallas detectadas es similar a la cantidad de fallas estimadas

Abaratamiento del Testing

- Hacer pruebas es caro y trabajoso
- La forma de abaratarlas y acelerarlas –sin degradar su utilidad- es:
 - DISEÑANDO EL SW PARA SER TESTEADO
- Algunas herramientas son:
 - Diseño Modular
 - Ocultamiento de Información
 - Uso de Puntos de Control
 - Programación NO egoísta

Enfoque de Caja Negra

Es un método de Software Testing en el cual la estructura interna, el diseño y la implementación del software es desconocido por el Tester.

- Prueba funcional, producida por los datos, o producida por la entrada/salida
- Prueba lo que el software debería hacer
 - Se basa en la definición del módulo a probar (definición necesaria para construir el módulo)
 - Nos desentendemos completamente del comportamiento y estructura interna del componente
- ¿Qué hacemos?
 - Seleccionamos subconjuntos de los datos de entrada posibles, esperando que cubran un conjunto extenso de otros casos de prueba posibles
 - Podemos suponer que la prueba de un valor representativo de cada clase es equivalente a la prueba de cualquier otro valor
 - Llamamos a c/subconjunto “Clase de Equivalencia”
 - Estas pruebas son llamadas “Pruebas por Partición de Equivalencia” o “Pruebas basadas en subdominios”

Condiciones de Prueba: Son descripciones de situaciones que quieren probarse ante las que el sistema debe responder

Casos de Prueba: Son lotes de datos necesarios para que se dé una determinada condición de prueba

- Criterio de selección:
 - De todas las combinaciones posibles sólo seleccionaremos algunas: La menor cantidad de aquellas que tengan mayor probabilidad de encontrar un defecto no encontrado por otra prueba.
 - También tomamos las condiciones de borde (ej. req es mayores de 18 años, 18 y 19 son condiciones de borde).
 - También sumar casos de ingreso de tipo de dato inválido.
 - También sumar condiciones cruzadas: si se indica que no se tiene hijos, no debería aceptar la entrada de nombre de los hijos
 - Conjetura de errores: Cosas que sabemos que pueden fallar o que no sabemos si están del todo bien
- Partición de clases:
 - dividimos los posibles casos en clases de equivalencia, y testeamos un caso por cada clase.
 - Dividimos las clases en clases válidas y clases inválidas
 - Por cada condición, tomamos un caso válido y uno inválido

Enfoque de Caja Blanca

Es un método de Software Testing en el cual la estructura interna, el diseño y la implementación del software es conocido por el Tester.

- El Tester elige qué valores de entrada utilizar para ejecutar ciertos caminos en el código, conociendo la potencial salida de ese camino.
- En este tipo de pruebas, es esencial que el Tester conozca el lenguaje de programación con el que el Software fue construido, debido a que se basa en cómo está estructurado el componente internamente y su definición

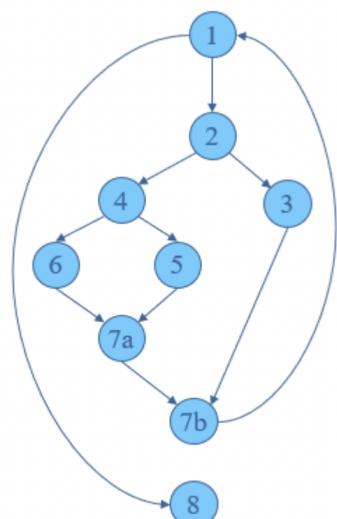
Grados de cobertura:

- Cobertura de Sentencias
 - Prueba c/instrucción
- Cobertura de Decisiones
 - Prueba c/salida de un “IF” o “WHILE”
- Cobertura de Condiciones
 - Prueba cada expresión lógica (A AND B) de los IF, WHILE
- Prueba del Camino Básico
 - Prueba todos los caminos independientes

Complejidad Ciclomática

- Métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa
 - cantidad de caminos independientes
 - camino independiente: agrega un nuevo conjunto de sentencias de procesamiento o una nueva condición
- Formas de calcularla
 - número de regiones del grafo
 - $V(g) = A - N + 2$ (A = aristas, N = nodos)

Ej:



```
procedimiento ordenar
1: do while no queden registros
   leer registro;
2:   if campo1 del registro =0
3:     then procesar registro
       guardar en buffer
       incrementar contador
4:   elsif campo2 del registro =0
5:     then reinicializar contador
6:   else procesar registro
       guardar en archivo
7a: endif
    endif
7b: enddo
8: end
```

Tipos de prueba

Prueba Unitaria

- Se realiza sobre una unidad de código claramente definida
- Generalmente lo realiza el área que construyó el módulo
- Se basa en el diseño detallado
- Comienza una vez codificado, compilado y revisado el módulo
- Los módulos altamente cohesivos son los más sencillos de probar

Prueba de integración

- Orientada a verificar que las partes de un sistema que funcionan bien aisladamente también lo hacen en su conjunto.
- TIPOS:
 - No Incrementales
 - BIG BANG
 - Incrementales
 - BOTTOM-UP
 - TOP-DOWN
 - “Sandwich”
- Los puntos clave son
 - Conectar de a poco las partes mas complejas
 - Minimizar la necesidad de programas auxiliares

Prueba de Aceptación de Usuario

- Prueba realizada por los usuarios para verificar que el sistema se ajusta a sus requerimientos
- Las condiciones de pruebas están basadas en el documento de requerimientos
- Es una prueba de “caja negra”

Prueba de Stress

- Orientada a someter al sistema excediendo los límites de su capacidad de procesamiento y almacenamiento.
- Teniendo en cuenta situaciones NO previstas originalmente

Pen Testing / Ethical Hacking

- Es la práctica de testear un software, red o aplicación web con el fin de encontrar vulnerabilidades de seguridad que un hacker pudiera explotar.
- El test en sí puede automatizarse utilizando aplicaciones software o ser realizado en forma manual. De cualquier manera, el proceso involucra obtener información acerca del software o servicio a probar, identificar potenciales puntos de entrada y reportar lo encontrado

Prueba de Volumen

- Orientada a verificar que el sistema soporta los volúmenes máximos definidos en la cuantificación de reqs.
 - Capacidad de Almacenamiento
 - Capacidad de Procesamiento

Prueba de regresión

- Orientada a verificar que, luego de introducido un cambio en el código, la funcionalidad original no ha sido alterada.
 - Hay que probar “lo viejo”
 - Reuso de condiciones & casos

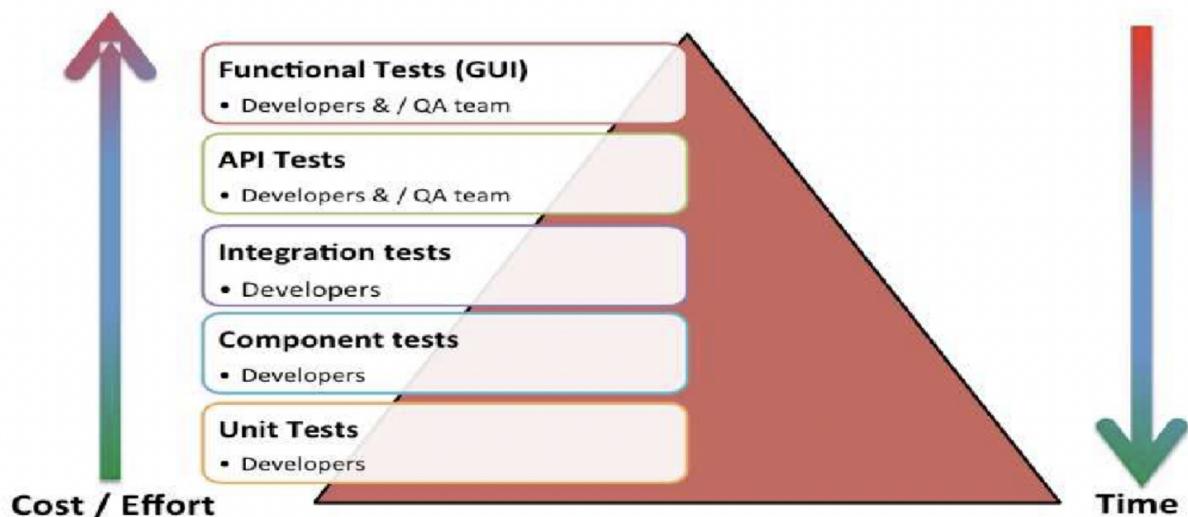
Prueba Alfa y Beta

- Se entrega una primera versión al usuario que se considera está lista para ser probada por ellos
 - Normalmente plagada de defectos
 - Una forma económica de identificarlos (ya que el trabajo lo hace otro)
 - En muchos casos no puede hacerse
 - Alternativa válida: ejecutar en “paralelo”
 - Alfa: la hace el usuario en mis instalaciones
 - Beta: la hace el usuario en sus instalaciones

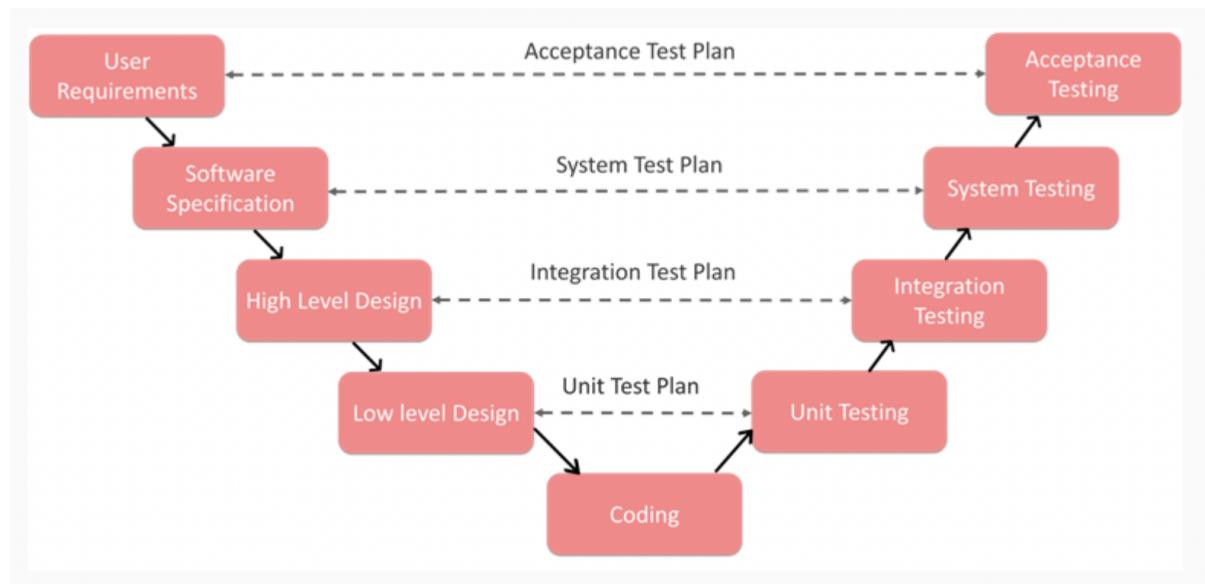
Clasificación de las pruebas

- Por el conocimiento
 - Caja blanca
 - Caja negra
 - Caja gris
- Por el tipo de ejecución
 - Automatizadas
 - Manuales
 - Semi-automatizadas
- Por el tipo de prueba
 - Funcionales
 - No Funcionales
- Por el alcance o nivel
 - Unitarias
 - De integración
 - De sistemas
 - De aceptación de usuario

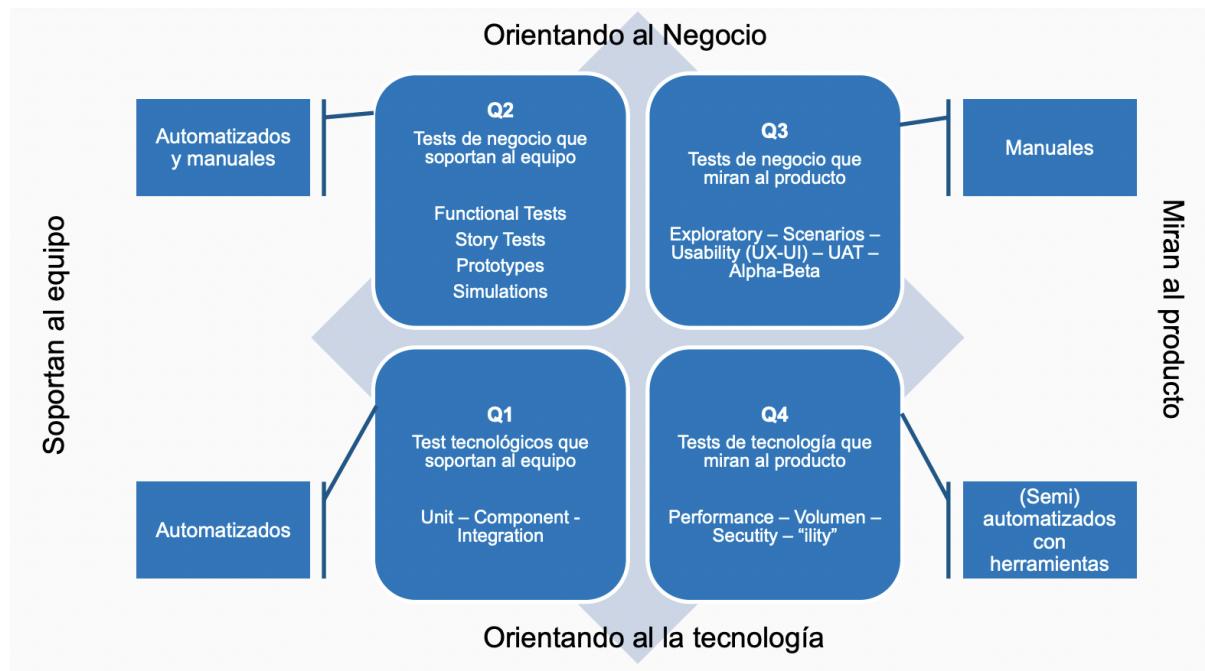
Pirámide del Testing



Modelo de ciclo de vida V



Cuadrantes, niveles y tipos de testing



Agile testing

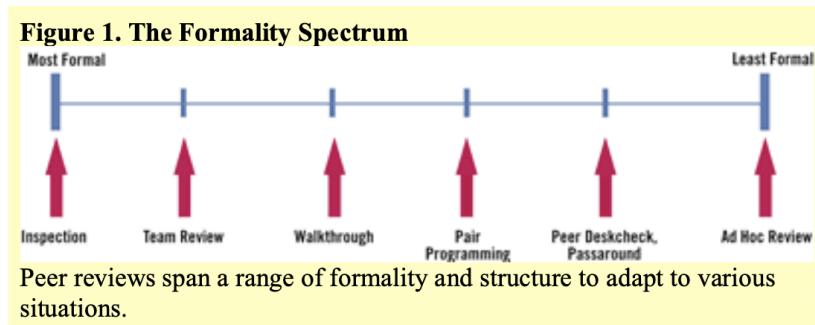
Test driven development

Se generan los test y luego las funcionalidades

- Pros
 - Aplicaciones más modulares, flexibles, y escalables
 - Código menos acoplado y más mantenible
 - El codificador puede hacer refactoring en cualquier momento con bajo riesgo
 - Soporta la regresión de bugs
 - Mejor cobertura de código
- Cons
 - Similares errores en el código y en el test case
 - Algunas funcionalidades son difíciles de probar con estos tests
 - Mantenimiento de los tests
 - Puede llevar a un falso nivel de confianza sobre la calidad del código

WhenTwoEyesAreNotEnough(KarlWiegers) (Paper)

En este paper formalizan distintos tipos de revisión que pueden realizarse sobre los componentes de software. Se detallan desde las revisiones más formales a las más informales.



Inspección: Es el método más formal, consta de un proceso detallado con roles particulares para los participantes. Comúnmente son 7 etapas, planificación, resúmenes, preparación, reunión, re trabajo, seguimiento y análisis de causas. El proceso consiste en que alguno de los participantes que no sea quien realizó el trabajo dirija la reunión como Moderador (moderator), otro miembro sea el que presenta el material al equipo de inspección Lector (reader) y por último el equipo es quien registrará los problemas (recorder). Al final del proceso se realiza una evaluación del trabajo y se establece como se verificarán los cambios que serán realizados en el rework.

Team Review: Los participantes reciben el material a revisar con cierta antelación a la reunión de revisión. Durante la reunión el equipo reúne la información de la revisión y los defectos encontrados. A diferencia de la inspección los pasos de seguimiento y resúmenes no se realizan, hay roles que se omiten (reader) y otro se combinan (quien dirige la reunión es quien construyó el producto). Una diferencia sustancial es que en lugar de tener a alguien describiendo una porción del código (reader) quien dirige la reunión consulta por comentarios.

Walkthrough: Revisión informal donde el autor de un trabajo lo describe a un grupo de compañeros y pide sus comentarios. No suelen seguir ningún procedimiento, no requiere reportes ni genera métricas. Suele usarse para hacer zoom sobre un cambio durante el mantenimiento, puede introducir un riesgo ya que tal vez sólo se presenta lo que se cambio y podrían haber otros cambios que no se están considerando.

En general se presenta como se construyó el componente y que hace, los flujos, entradas y salidas.

Pair Programming: Codear en conjunto para minimizar las fallas y elevar la calidad del producto resultante. No es una técnica específica de revisión más bien una estrategia de desarrollo

Peer Deskcheck: Sólo una persona a parte del autor verifican el trabajo, depende enteramente de quien sea el revisor. Es el método más barato y se puede aplicar para trabajos no riesgosos

Passaround: Es un deskcheck con múltiples participantes, en general se envía una copia del trabajo realizado a otros colaboradores para obtener su feedback. Este método mitiga dos riesgos del deskcheck normal: no obtener feedback a tiempo y la posibilidad de que el revisor haga un trabajo malo. Involucrando más personas por lo que se obtendrá más feedback. Si se hace de manera online se obtiene cierta sinergía ya que los revisores pueden ver los comentarios realizados por otros participantes, evitando redundancias y mostrando distintas interpretaciones.

Ad hoc review: El desarrollador le solicita a otro compañero que lo ayude a revisar una pieza de trabajo.

Using Heuristic Test Oracles (Paper)

En el paper explican una manera de poder justificar porque algó que detectaste probando está mal aunque no tengas documentación que indique nada al respecto.

ej: si estás usando un programa que tiene un campo para ingresar texto libre y al ingresar 5k de caracteres falla, posiblemente no haya un caso escrito sobre esa prueba y a la hora de levantar el punto podría suceder que el equipo de desarrollo diga que no estaba considerado en los requisitos

La técnica se base en apalancarse en distintas miradas para poder justificar porqué el hallazgo puede considerarse como un bug a corregir

History: Inconsistencia con su historia, el producto debe ser consistente con sus versiones pasadas. Si algo cambió y nadie informó que iba a cambiar posiblemente sea un error.

Image: Inconsistencia con su imagen, considerando la relevancia de tener una “buena imagen en el mercado” cualquier situación que se encuentre dentro del software que de la sensación de que el producto no es lo suficientemente profesional o que no se adecua a los estándares mínimos es un problema.

Comparable: Inconsistencia con productos comparables, tomando como base un producto comparable si encontramos algo que no estamos ofreciendo o estamos haciendo distinto que el elegido para la comparación podemos determinar que hay algo a resolver.

Claims: Inconsistencia con afirmaciones, tiene que ver con tener inconsistencias en las afirmaciones que hacemos sobre el producto, un ejemplo podría ser tener TyC desactualizados donde no se retrate el comportamiento real del producto.

User Expectations: Inconsistencia con la expectativas de usuario, el producto no hacer algo que un usuario esperaría que haga, para poder sopesar este punto es necesario tener alguna idea de quién es el usuario y que planea hacer con el producto.

Product: Inconsistencia con el producto, comportamientos diferentes en distintas partes del producto, por ejemplo muestro el cálculo de un precio de un servicio que estoy ofreciendo en una vista pero en otra vista tengo un precio distinto para el mismo servicio.

Purpose: Inconsistente con el propósito, se detecta un comportamiento distinto al que el usuario espera tener con el software. Un ejemplo podría ser que tengo un software que me permite dar descuentos sobre distintos artículos y al utilizarlo el precio resultante termina siendo superior al original.

Exploratory Testing Explained

No llegué a verlo, explica un poco que es exploratory testing y porque es bueno hacerlo.

Unidad 7: Software Metrics

Conceptos

Medida: proporciona una indicación cuantitativa de la extensión, cantidad, dimensiones, capacidad o tamaño de algunos atributos de un proceso o producto. Ej. un programa tiene 10.000 LDC (líneas de código)

Métrica: es una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado. Ej. la productividad de este proyecto fue de 500 (LDC/persona-mes)

Indicador: es una métrica o combinación de métricas que proporcionan una visión más profunda de un proceso, un proyecto de software o de un producto en sí

KPI: Key Performance Indicator,

GQM - Goal Question Metric

- 1. Conceptual (Goal)
 - Metas (goals): aquello que la organización intenta alcanzar (SMART)
- 2. Operacional (Question)
 - Preguntas (questions): son aquellas preguntas cuyas respuestas permiten definir el cumplimiento de las metas
- 3. Cuantitativo (Metrics)
 - Métricas (metrics): las mediciones necesarias para ayudar a responder a las preguntas y confirmar si las mejoras del proceso cumplieron su objetivo

Clasificación



Métricas de Proceso

- Duración promedio de proyectos.
- Cantidad de proyectos segregados por tipo, tamaño, etc.
- Esfuerzo promedio segregado por tipo, duración.
- Defectos introducidos en una fase del ciclo de vida.
- Defectos detectados en una fase del ciclo de vida.
- % de tiempo/esfuerzo/costo dedicado a una fase del ciclo de vida.
- % promedio de desvío en proyectos (En costo o duración).

Métricas de Software (Producto)

- Cantidad de líneas de código (LOC)
- Funcionalidad (Puntos por Función / Use Case Points)
- Complejidad Ciclomática (Mc Cabe)
- Cohesión
- Acoplamiento
- Calidad de Producto (ISO 25000)
- Cantidad de fallas de un producto
- Confiabilidad (MTBF, MTTR)
 - MTBF: Mean Time Between Failures
 - MTTR: Mean Time To Recovery

Métricas de Recursos

- Cantidad de fallas detectadas por tester
- Cantidad de LOC producidas por un desarrollador
- Esfuerzo dedicado a codificar por un desarrollador
- Edad promedio del equipo
- Costo promedio de cada rol
- Años de experiencia promedio de cada rol