UNIVERSIDAD DE LIMA | 60 años

PRUEBAS UNITARIAS

UNIDAD 4: PRUEBAS DE SOFTWARE

Temario

- Concepto de pruebas unitarias.
- Pruebas unitarias estáticas y dinámicas.
- Herramientas para pruebas.
- Pruebas unitarias en eXtreme Programming

UNIVERSIDAD DE LIMA | 60 años

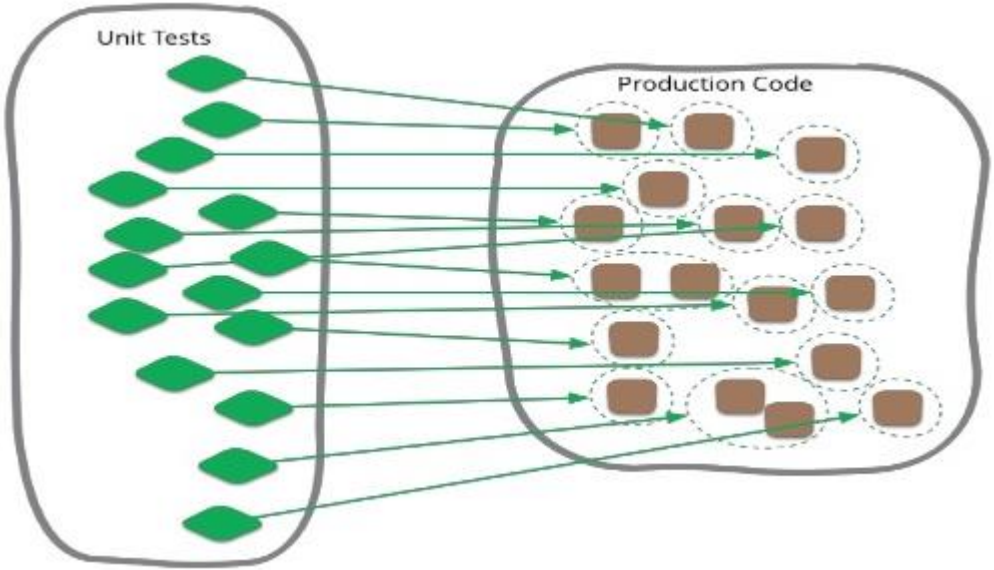
Pruebas de Unidad (Unitarias)

Pruebas Unitarias

- Cada pieza de código funcional (clase o método) será susceptible de ser probada con pruebas unitarias.
- Estas son pruebas automáticas (código) que se encargan de simular la interacción de entidades externas con la pieza de código (desde ahora componente).

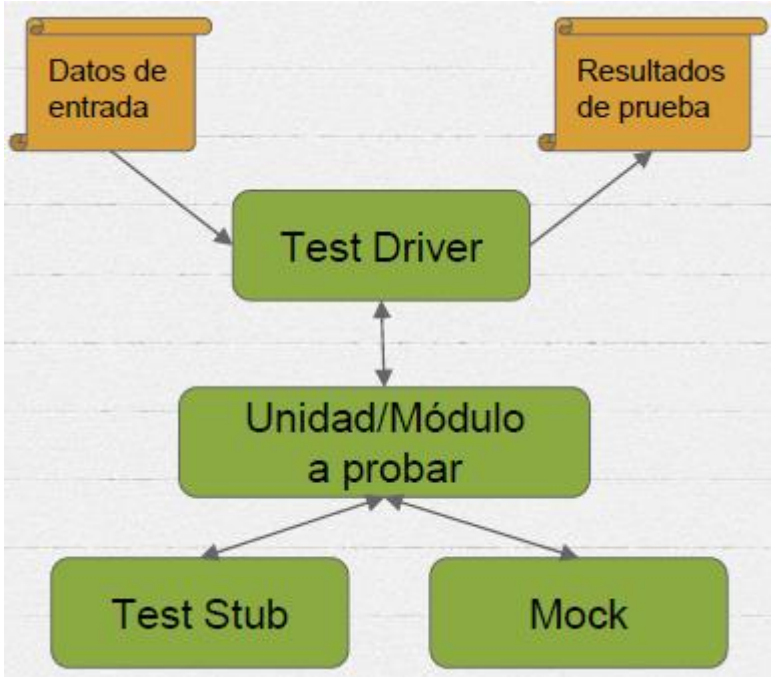
TIPOS DE PRUEBAS

- **Pruebas de Unidad (Unitarias)**
  - Se debe probar el objeto por completo.
    - Todos los métodos
    - Todos las salidas
    - Todas las entradas
    - Todos los estados
    - Todas las herencias
  - Se pueden usar las técnicas de **clases equivalentes**.



TIPOS DE PRUEBAS

- **Pruebas de Unidad (Unitarias)**
  - **Test Driver (Manejador de prueba)**: Objeto que llama al módulo a probar.
  - **Test Stub**: Objeto (que el módulo a probar llama) que siempre devuelve un resultado estático.
  - **Mock Object:** Objeto (que el módulo a probar llama) que devuelve resultados, sin que esté implementado.

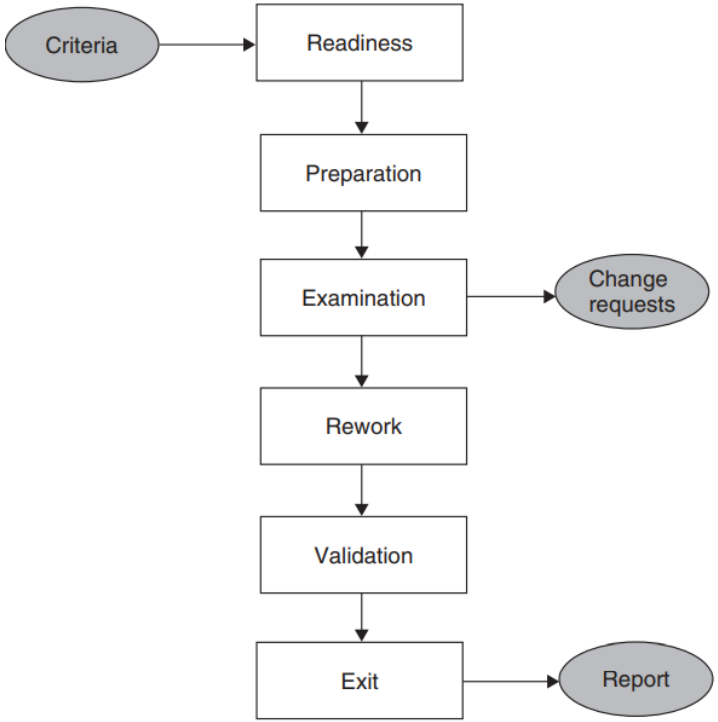


UNIVERSIDAD DE LIMA | 60 años

Pruebas unitarias estáticas

Prueba unitaria estática

- Se llevan a cabo en una fase de inspección y corrección donde el producto no necesita estar en su forma final. Por ejemplo, la finalización de la codificación es un hito, pero no necesariamente representa el producto finalizado. La idea detrás de la revisión es encontrar los defectos lo más cerca posible de sus puntos de origen para que esos defectos se eliminen con menos esfuerzo y el producto provisional contenga menos defectos antes de emprender la siguiente tarea.
- Técnicas
  - Inspection
  - Walkthrough



Code review

Code Review Metrics

- Mah. Michael define las métricas de software como “La aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo del software y sus productos para suministrar información relevante a tiempo, así el administrador junto con el empleo de estás técnicas mejorará el proceso y sus productos”
- Estas métricas de software:
  - Lines of Code (LOC)
  - Complejidad ciclomática (Cyclomatic complexity)
  - ABC Score

Line of Code - LOC (cuantitativa)

- Es la métrica más común y tradicional de poder medir que tan complejo (feo) puede estar cierto código.
- Se basa en medir las líneas de código que un programa puede tener.
- Beneficios
  - Simplicidad.
  - Bajo costo (en tiempos)
- Inconvenientes
  - No refleja la realidad. Dos código que realizan lo mismo pueden tener diferentes líneas de código (según el estilo de codificación del programador).
  - No se toman en cuenta comentarios y sentencias que no aportan a la complejidad.

Tipos de LoC

- Hay dos métricas de LoC y difieren según el método de medición:
  - LoC lógico.
  - LoC físico

Complejidad Ciclomática

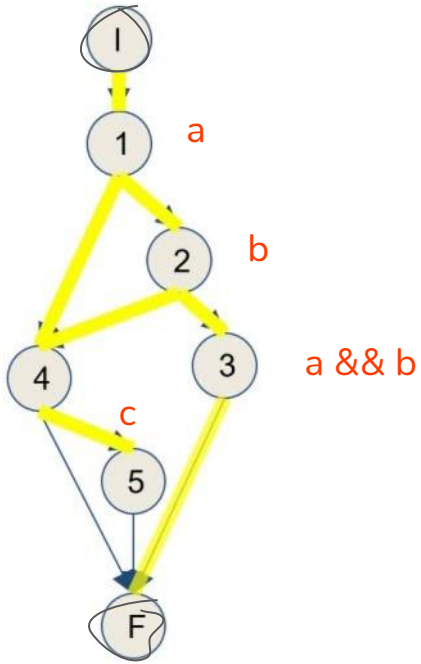
- La Complejidad Ciclomática es una métrica del software ampliamente difundida y aceptada, que permite obtener una medición cuantitativa de la complejidad lógica de un programa o pieza de software. Tiene la particularidad de haber sido desarrollada de manera que pueda aplicarse con independencia del lenguaje en que se ha programado.
- Originalmente esta métrica fue propuesta por Thomas McCabe en 1976 y se basa en el diagrama de flujo determinado por las estructuras de control del código correspondiente a un programa.

Complejidad Ciclomática - Ejemplo

```
public void aMethod() {

      if (a && b) {
            doSomething();
      } else if (c) {
            doSthElse();
      }
}
```

else



V(G) = Aristas-Nodos+2

V(G) = 9 - 7 + 2 = 4

Nodo -> Entrada y salida del Método

Nodo -> Condiciones

Aristas -> Flujos de ejecución

```
int x,y,r;
 if (x<0 || y<0) {
     system.out.println(“X o Y son negativos”);
 } else {
     r=(x+y)/2;
     system.out.println(“La media de X e Y es:” + r );
 }
```

UNIVERSIDAD DE LIMA

ABC Score

- No mide complejidad, pero provee una manera de medir tamaños de código fuente sin prestar atención a distintos estilos de programación.
- Método de cálculo
  - Se calculará el vector ABC conformado por 3 componentes:
    - Assigments: Transferencia explícita de datos a una variable
    - Branch : Una llamada explicita a código fuera del alcance de la función analizada.
    - Conditionals: Un test lógico que devuelve una variable booleana.
  - Se contarán cada una de las ocurrencias de estos componentes en una porción de código y se le asignará a un vector V.
- Beneficios
  - Nos permite tener una métrica que compara distintos códigos sin tomar en cuenta el estilo de programación del desarrollador.
  - Nos da un análisis sencillo y rápido de porciones de código.

UNIVERSIDAD DE LIMA | 60 años

Pruebas unitarias dinámicas

Prueba unitaria dinámicas

Las pruebas unitarias basadas en la ejecución se conocen como pruebas unitarias dinámicas. En esta prueba, una unidad de programa se ejecuta realmente de forma aislada.

Se diferencia de la ejecución ordinaria en lo siguiente:

1. Una unidad bajo prueba se saca de su entorno de ejecución real.
2. El entorno de ejecución real se emula escribiendo más código para que la unidad y el entorno emulado se puedan compilar juntos.
3. La unidad bajo prueba y el código adicional se ejecuta con entradas seleccionadas previamente. El resultado de una ejecución de este tipo se recopila de diversas formas y se compara con el resultado esperado. Cualquier diferencia entre el resultado real y el esperado implica una falla y la falla está en el código.

UNIVERSIDAD DE LIMA | 60 años

Gestión de Pruebas

Las pruebas en la organización

- Consideraciones
  - “¿Si existe un equipo independiente, para qué hacemos pruebas los desarrolladores?”
  - El equipo de pruebas puede ser más directo al dar sus resultados sin temer repercusiones al criticar el trabajo de sus compañeros.
  - Un equipo especializado puede obtener más respeto de la gerencia.
  - Habilidades blandas para flexibilizar su relación con el área de desarrollo. La inflexibilidad puede ser fatal.

El equipo de pruebas

- Cuando se define un equipo especializado para las pruebas, podemos encontrar los siguientes roles.
  - Líder de testers: Planifica, monitorea y controla la ejecución de las pruebas. Son gestores del proceso de pruebas. Pueden ser llamados coordinadores o administradores de pruebas.
  - Tester: Puede ayudar en la planificación y/o diseño de las pruebas. Ejecutan, documentan e informan los resultados de las pruebas.

Habilidades requeridas (sugeridas) para un tester

- Las habilidades básicas requeridas son de comunicación oral y escrita. Además:
  - Dominio del tema o aplicación: Debe saber para qué sirve lo que está probando. Conocer las reglas del negocio .
  - Dominio de tecnología: Debe saber las limitaciones y prestaciones de la tecnología utilizada para la implementación.
  - Dominio de Pruebas: Todos los temas que hemos visto en estas clases así como experiencia.

Habilidades requeridas (sugeridas) para un tester Ágil

- Actitud positiva y orientada a resultados
- Hacer los resultados de pruebas visibles, para tomar decisiones colectivas.
- Claridad de expresión
- Capacidad de análisis para entender los resultados de las pruebas.
- Responden positivamente al cambio
- Autodidactas y apasionados del testing
- Entienden sobre criterios de aceptación
- Orientados a la victoria del equipo

Estimaciones para las pruebas

- Estimar el tiempo, costo y recursos necesarios para las pruebas puede tomar dos enfoques:
  - Estimación experta: Usar el conocimiento del equipo sobre las tareas a realizar
  - Estimación paramétrica: Usar métricas de datos históricos. Por ejemplo: pruebas x tester x día, defectos x tester x día, clasificar por complejidad.

Estrategias o enfoque al planificar pruebas

Se puede elegir entre distintas estrategias para definir la estrategia a utilizar para planificar y/o estimar las pruebas:

- Analítico: Se pueden analizar los riesgos, requerimientos u otros elementos obtenidos durante el diseño del SW.
- Basado en modelos: Creación o selección de modelos (matemáticos) para probar comportamientos críticos del sistema.

Estrategias o enfoque al planificar pruebas

- Metodológico: Usando un procedimiento de pruebas definido por experiencia, normalmente hecho in-house.
- Por Cumplimiento de Estándares o Procesos: Usando el estándar IEEE 829, o XP (ágil) o cualquier procedimiento desarrollado por terceros.

Estrategias o enfoque al planificar pruebas

- Dinámico: Se enfoca en adaptarse a los errores encontrados en las ejecuciones del sistema. Por ejemplo: las pruebas exploratorias o las pruebas basadas en ataques.
- Dirigido: Se centran en lo que los usuarios quieren probar (o que ellos mismos lo prueben).
- Regresionista: Buscando que lo probado una vez, pueda volver a probarse cada vez que haya un cambio. Para ello, la automatización es fundamental.

Elementos para definir la estrategia

¿Cómo elegir la(s) estrategia(s) a usar (o combinar)?

Debemos considerar:

- Riesgos
- Habilidades
- Objetivos
- Regulaciones
- Producto
- Negocio

Gestión de la Configuración & Gestión de Pruebas

- Gestión de la Configuración: Determinación de los elementos (y sus versiones) que arman un SW o sistema. Estos elementos incluyen: código fuente, scripts de pruebas, SW de terceros, hardware, datos, documentación, etc.

¿Por qué se relaciona con las pruebas?

Gestión de la Configuración & Gestión de Pruebas

- Asegurar que se entrega al equipo de pruebas la versión correcta de todos los elementos.
- Especificar la configuración de dichos elementos para poder realizar las pruebas.
- Identificar qué defectos se encuentran en qué versión del elemento.

Aseguramiento de la calidad (QA) *VS* Control de calidad (QC) / Pruebas

	QA	QC
Definición	Actividades que aseguran la calidad en los procesos que desarrollan el producto.	Actividades que aseguran la calidad del producto.
Enfoque	Prevenir defectos, se centran en el proceso. Es proactivo.	Identificar defectos en el producto. Es reactivo.
Objetivo	Mejorar el desarrollo y pruebas para que no surjan defectos cuando el producto se desarrolla.	Identificar defectos después (o durante) el desarrollo pero antes de su entrega.

Aseguramiento de la calidad (QA) *VS* Control de calidad (QC) / Pruebas

	QA	QC
Cómo	Establecer un sistema de gestión de calidad y revisar que sea adecuado. Auditorías periódicas.	Encontrar y eliminar problemas de calidad para asegurara que siempre se cumplan los requerimientos.
Responsable	Todo el equipo encargado del desarrollo.	Es equipo encargado de las pruebas.
Tipo de Herramienta	Es una herramienta de gestión	Es una herramienta correctiva.

UNIVERSIDAD DE LIMA | 60 años

Herramientas para pruebas

Herramientas para pruebas

- Herramientas para la gestión de pruebas
- Herramientas de soporte del testing estático
- Herramientas de especificación de pruebas
- Herramientas para ejecución de pruebas
- Herramientas para soporte del rendimiento y monitoreo

Beneficios del uso de Herramientas

- Reducción de trabajo repetitivo
- Consistencia
- Repetibilidad
- Más casos de prueba realizados
- Recursos orientados a los objetivos
- Reducción de riesgos
- Retroalimentación rápida para los desarrolladores
- Acceso sencillo a la información sobre las pruebas
- Se tiene una plantilla para definir el proceso de pruebas

Riesgos del uso de Herramientas

- Expectativas irreales
- Subestimar el tiempo, costo y esfuerzo de la introducción de la herramienta
- Subestimar el tiempo y esfuerzo para conseguir beneficios continuos y significativos
- Dependencia a la herramienta
- No contar con las habilidades crear las pruebas
- No contar con las habilidades para usar la herramienta

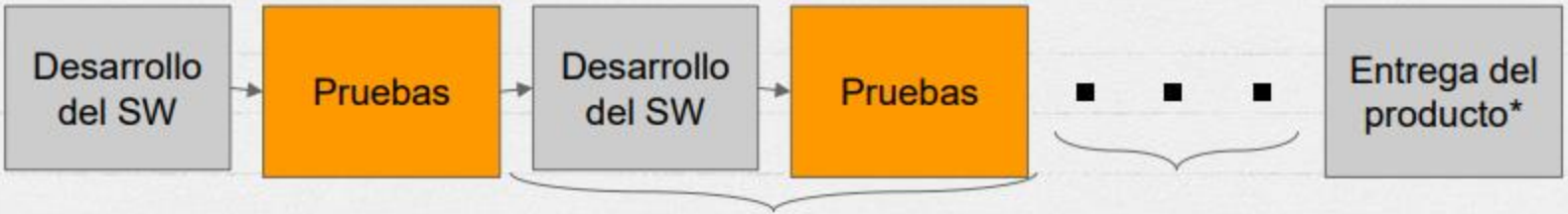
UNIVERSIDAD DE LIMA | 60 años

Pruebas unitarias en eXtreme Programming

Conceptos de Ciclo de Vida del SW

Ciclo de Vida Cascada

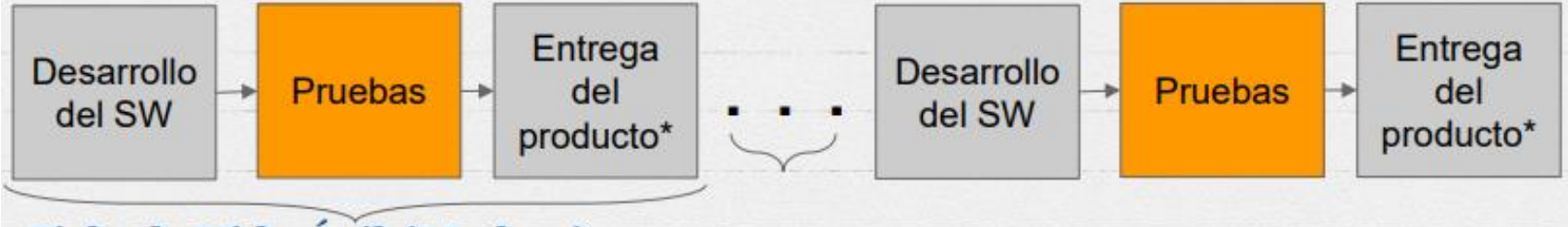
Ciclo de Vida Incremental



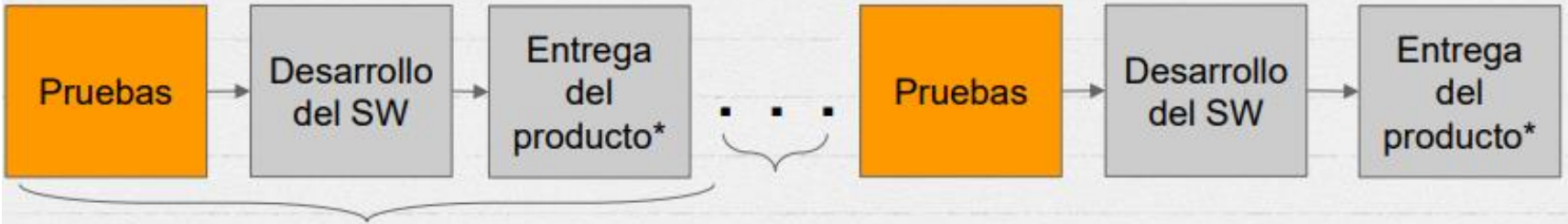
* Incluyendo pruebas de aceptación/validación

Conceptos de Ciclo de Vida del SW

Ciclo de Vida Ágil (Modo 1) / Evolutivo



Ciclo de Vida Ágil (Modo 2)



* Incluyendo pruebas de aceptación/validación

Algunos Conceptos Ágiles

- Pequeños Releases: Se entrega un SW funcional después de cada Sprint.
- Pruebas de Aceptación: Cada historia de usuario (requisito) debe tener una (o más) pruebas de aceptación.
- Propiedad Colectiva: No existe dueño del código. Todos pueden modificarlo.
- Integración Continua: Sistema integrado contínuamente. Se compila/integra varias veces al día.

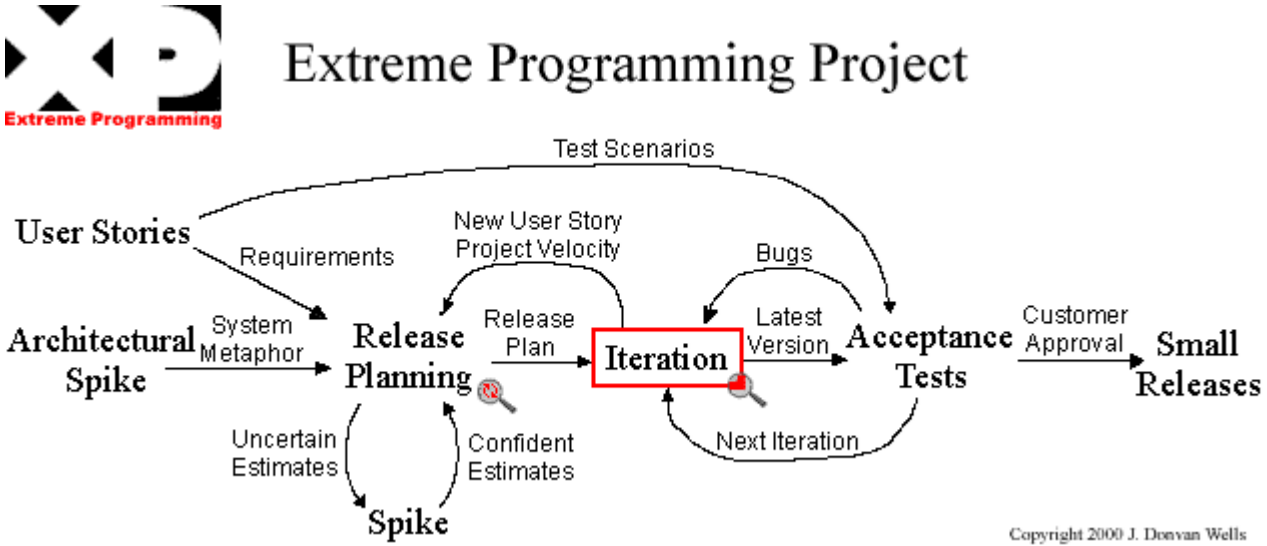
Algunos Conceptos Ágiles

- Desarrollo (Diseño) dirigido por Pruebas:
  - Técnica de diseño: basados en ejemplos
  - Sólo se desarrollan las funcionalidades requeridas.
  - Minimizar el número de errores.
  - Desarrollo modular y reutilizable.
  - TDD: TFD + Refactorización
  - TFD: Test-First Development

Algunos Conceptos Ágiles

- Refactorización:
  - Siempre se busca entregar valor de negocio
  - Desarrollo iterativo -> diseño simple
  - Mejora Contínua del Diseño
  - Procesos se enfocan en:
    - Remover duplicaciones
    - Incrementar Cohesión (modularidad)
    - Disminuir acoplamiento
- Diseño Emergente: El diseño va surgiendo de la revisión y refactorización del código. No hay un gran diseño anticipado.

Extreme Programming



UNIVERSIDAD DE LIMA

Referencias

- Proceso de Construcción de Software 2, Mag. Natalí Flores Lafosse, Maestría en Informática de la PUCP, 2018.
- Sommerville, I. (2005). Ingeniería del software. Pearson educación.
- Bourque, P., & Fairley, R. E. (2014). Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0. IEEE Computer Society Press.