UNIVERSIDAD DE LIMA | 60 años

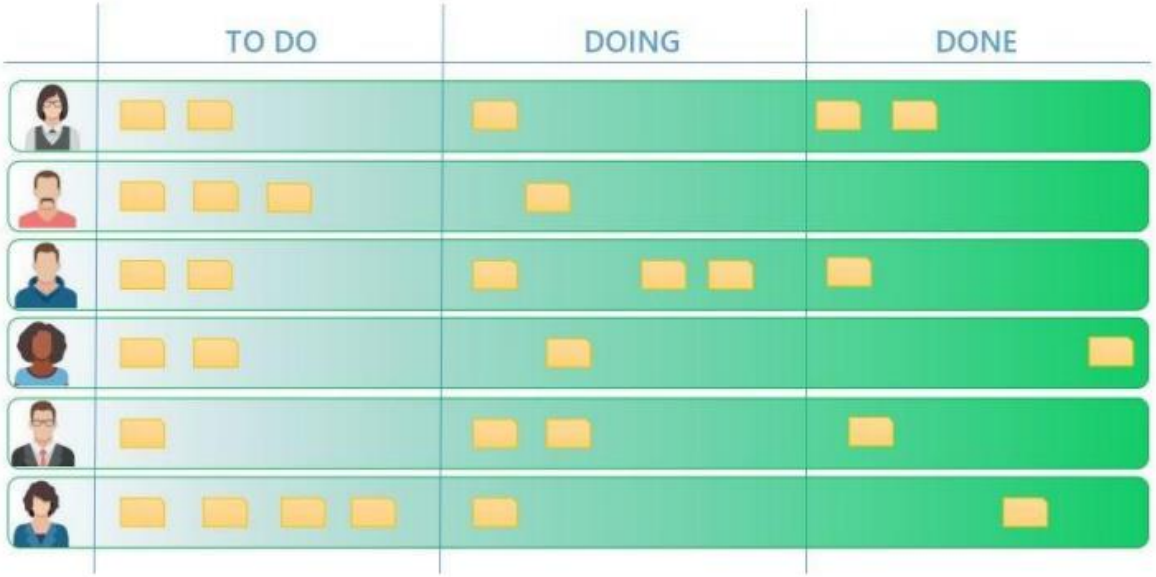
DISEÑO ÁGIL

UNIDAD 3: PRÁCTICAS DE DISEÑO ÁGIL

Temario

- Conceptos de diseño ágil
- Principios de diseño de paquetes
- Arquitectura en proyectos ágiles.

Motivación — Tablero Kanban



Trello

http://www.youtube.com/watch?v=I-H-WXAX_oM

UNIVERSIDAD DE LIMA

UNIVERSIDAD DE LIMA | 60 años

Métodos ágiles

Los métodos ágiles nacieron en la década de 1990 a partir de la necesidad de reducir los gastos generales aparentemente grandes asociados con los métodos pesados basados en planes utilizados en proyectos de desarrollo de software a gran escala.

Métodos ágiles

- Los métodos ágiles se consideran métodos ligeros
  - Ciclos de desarrollo breves e iterativos
  - Equipos autoorganizados
  - Diseños más simples
  - Refactorización de código
  - Desarrollo basado en pruebas
  - Participación frecuente del cliente
  - Énfasis en la creación de un entorno de trabajo transparente.

Manifiesto Ágil

- El manifiesto Ágil surge el 17 de febrero del 2001, cuando se reunieron diecisiete críticos del desarrollo de software, y acuñaron el término "metodología Ágil" para definir los métodos que estaban surgiendo como alternativa a las metodologías formales.
- El manifiesto Ágil está conformado por 12 principios asociados a 4 aspectos o pilares.

Aspectos o Pilares del Manifiesto

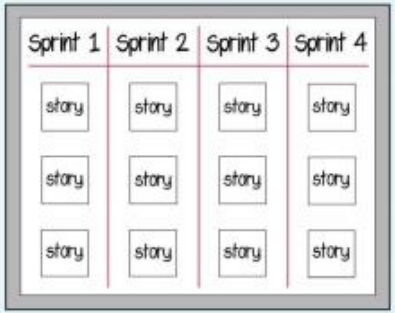
1 Satisfacción del **cliente**

Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor

2 Bienvenidos los **cambios**

Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.

3 Entregas **frecuentes**



Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.

4 Trabajamos **juntos**

Los responsables de negocio y los desarrolladores trabajamos juntos diariamente durante todo el proyecto.

Fuente: http://www.knowledgetrain.co.uk/resources/practice/agile-principles

UNIVERSIDAD DE LIMA

Aspectos o Pilares del Manifiesto

5 Confianza y
apoyo

Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.

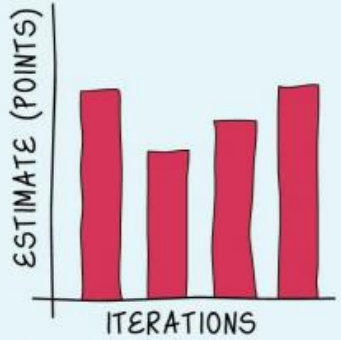
6 Conversaciones
cara a cara

El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.

7 **Software**
funcionando

El software funcionando es la medida principal de progreso

8 **Desarrollo**
sostenible

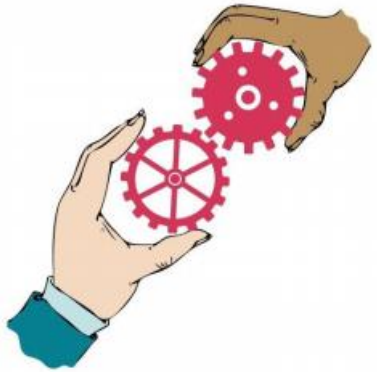


Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.

UNIVERSIDAD DE LIMA

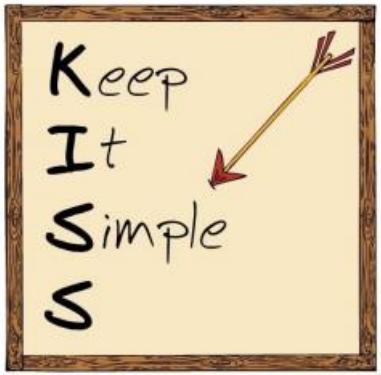
Aspectos o Pilares del Manifiesto

9 Atención continua



La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.

10 La simplicidad



El arte de maximizar la cantidad de trabajo no realizado, es esencial.

11 Equipos autoorganizados

Las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados

12 Reflexionar y ajustar

A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

UNIVERSIDAD DE LIMA

Declaración de Interdependencia

- La Declaración de Interdependencia en la gestión de proyectos fue escrita a principios del 2005 por un grupo de 15 líderes de proyectos como un suplemento al "Manifiesto Ágil". Enumera seis valores de gestión necesarios para reforzar una mentalidad de desarrollo ágil, particularmente en la gestión de proyectos **complejos e inciertos**.
- Los 6 Valores de la Declaración de Interdependencia
  1. Aumentamos el retorno de inversión, al enfocarnos en el flujo continuo de valor.
  2. Ofrecemos resultados fiables mediante la participación del cliente en las iteraciones frecuentes, donde también son responsables por el trabajo.
  3. Asumimos que habrá incertidumbre y las superamos a través de iteraciones, anticipación y adaptación.
  4. Damos rienda suelta a la creatividad y la innovación al reconocer que las personas son la fuente máxima de valor y creamos un entorno en el que puedan tener un impacto positivo.
  5. Aumentamos el rendimiento a través de la rendición de cuentas por parte del grupo en cuestión de resultados y eficacia del equipo, responsabilidades que todos comparten.
  6. Mejoramos la eficacia y la fiabilidad a través de estrategias situacionalmente específicas, procesos y prácticas.

UNIVERSIDAD DE LIMA

¿Qué es Agilidad?

- "Agilidad es la capacidad de crear y responder al cambio con el fin de obtener ganancias en un entorno empresarial turbulento".

- "La agilidad es la capacidad de equilibrar la flexibilidad y estabilidad"

UNIVERSIDAD DE LIMA

¿Qué es Agilidad?

¿Cómo debemos ver a la Agilidad?

- En cualquier tipo de disciplina de gestión, ser ágil es una cualidad, por lo tanto esto debe ser una meta que se debe tratar de alcanzar.
- La gestión de proyectos Agile especialmente, implica la adaptabilidad durante la creación de un producto, servicio o cualquier otro resultado.

¿Por qué Metodologías Ágiles?

- El 80% de todos los proyectos emplearán Métodos Ágiles en los próximos años (Gartner).
- Casi tres cuartas partes (71%) de las organizaciones informan que utilizan enfoques ágiles a veces, a menudo o siempre

Gestión de Proyectos Tradicional

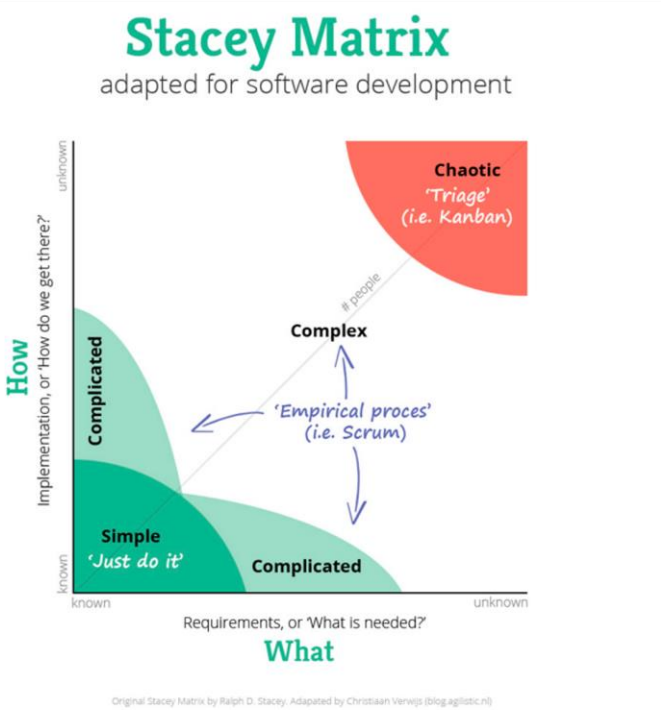
- Ventajas: Orden lógico.
- Desventaja: Asume predictibilidad.

UNIVERSIDAD DE LIMA

UNIVERSIDAD DE LIMA

Agilidad y Complejidad

Professor of Management, Ralph D. Stacey



https://lnds.net/blog/lnds/2018/07/08/agilidad-y-complejidad/

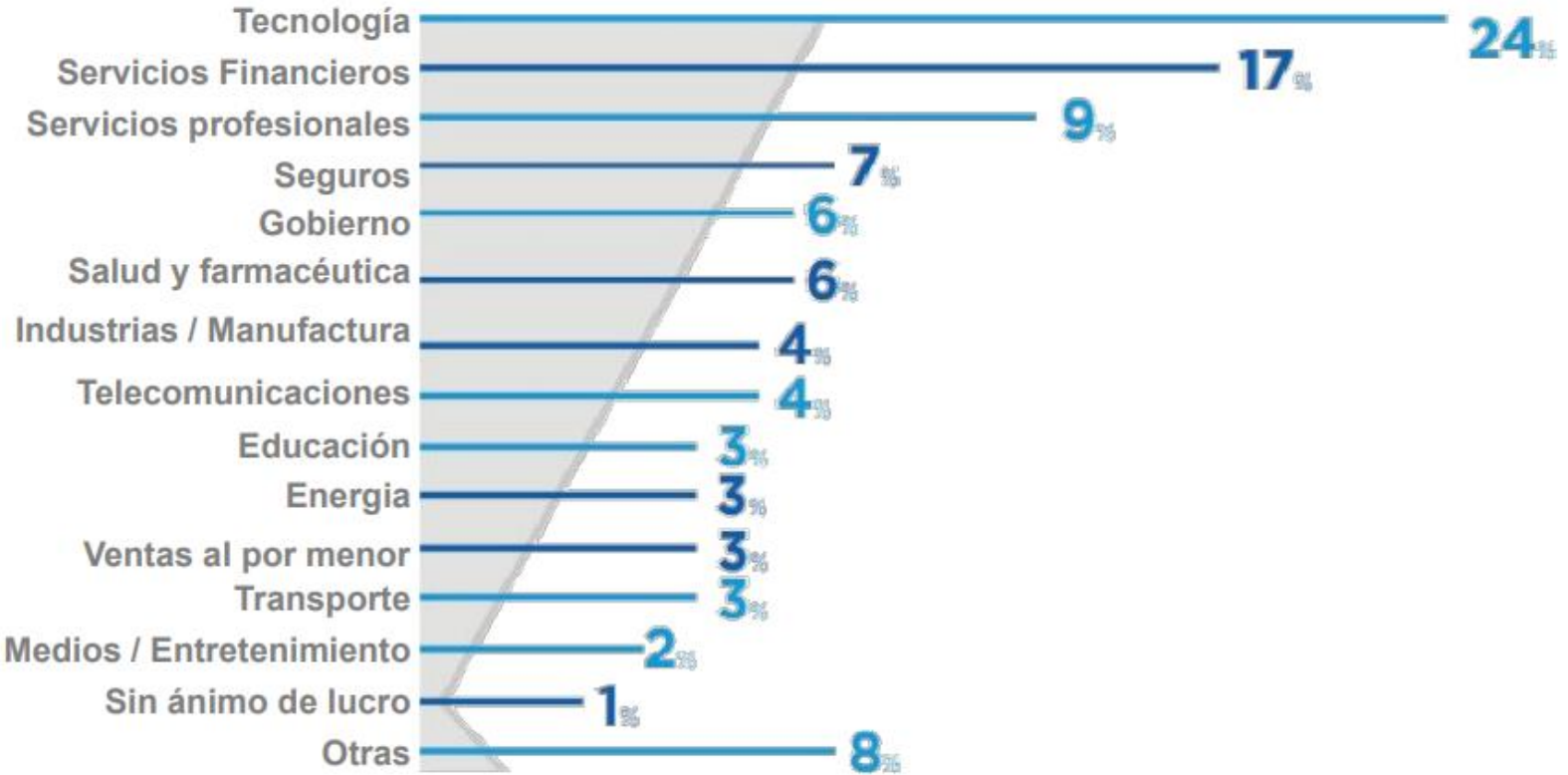
UNIVERSIDAD DE LIMA

Razones para adopción ágil

Fuente: https://explore.versionone.com/state-of-agile

UNIVERSIDAD DE LIMA

Aplicación en la industria



Fuente: https://explore.versionone.com/state-of-agile

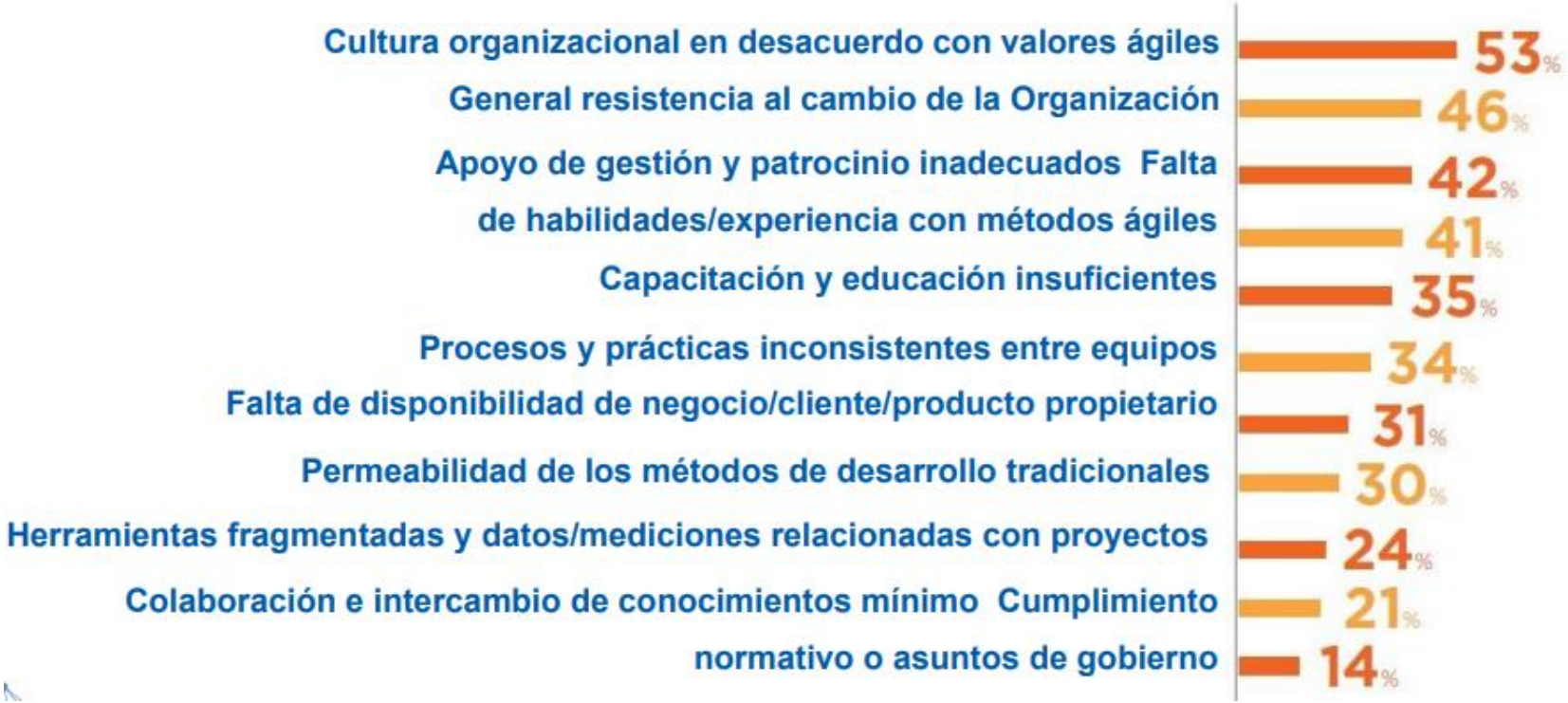
UNIVERSIDAD DE LIMA

¿Cómo medir el éxito de las iniciativas ágiles?

Fuente: https://explore.versionone.com/state-of-agile

UNIVERSIDAD DE LIMA

Principales barreras al adoptar ágil



Fuente: https://explore.versionone.com/state-of-agile

UNIVERSIDAD DE LIMA

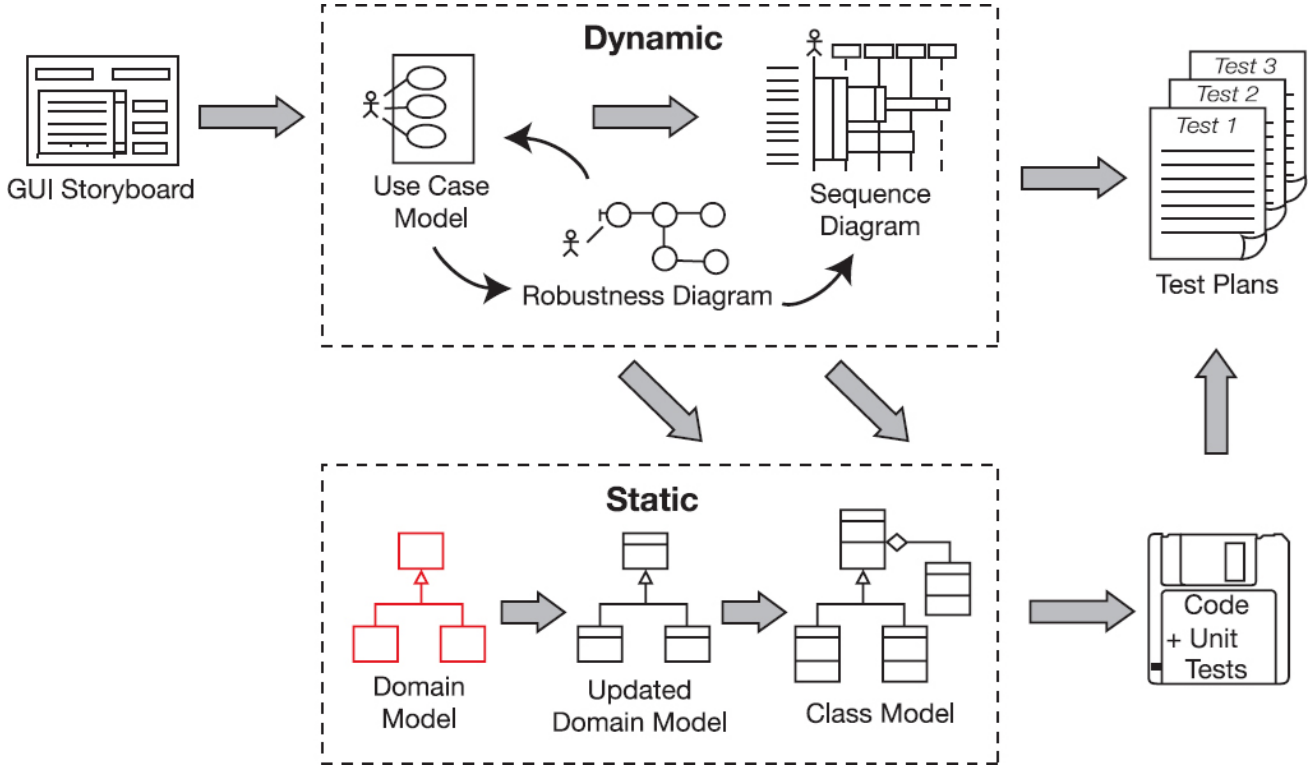
¿Qué es Scrum?

- Scrum es un marco de trabajo de adaptación iterativa e incremental, rápido, flexible y eficaz diseñado para ofrecer un valor significativo de forma rápida en todo el proyecto.
- Scrum es:
  - Ligero.
  - Fácil de entender.
  - Extremadamente difícil de llegar a dominar.
- Usos de Scrum
  - Scrum fue desarrollado inicialmente para gestionar y desarrollar productos. Desde principios de los años 90.
  - Scrum se ha usado para desarrollar software, hardware, software embebido, redes de funciones interactivas, vehículos autónomos, escuelas, gobiernos, mercadeo, también para gestionar la operación de organizaciones y casi todo lo que usamos en nuestra vida diaria, como individuo y como sociedad.
  - Scrum demostró ser especialmente efectivo en la transferencia iterativa e incremental de conocimiento.
  - Scrum se usa ahora ampliamente para productos, servicios y gestión de la organización matriz.

XP eXtreme Programming

- La metodología ágil más conocida y más utilizada es la programación extrema (XP)
- Es la única metodología ágil que se centra principalmente en el lado de la programación del desarrollo de software.
- Complemento de SCRUM ya que nos permite definir prácticas y técnicas:
  - Pruebas automáticas
  - Programación en pares
  - Refactoring

ICONIX un ejemplo de XP



UNIVERSIDAD DE LIMA

Extreme Programming versus Scrum

Ventajas y Desventajas

- Extreme Programming como su nombre indica se podría considerar, pues ... extrema
- Scrum, por otra parte, es más fácil de digerir
- Scrum es menos técnico, menos detallada y menos estricta
  - Puede causar que los ingenieros se pierdan ó
  - Puede hacer libres a los ingenieros (menos es más!)
- Las prácticas de Extreme Programming son altamente dependientes
  - Que hace más difícil para la adopción gradual
- Las prácticas de Extreme Programming dependen de excelentes profesionales

Mejor Juntas XP y Scrum

- XP y Scrum no compiten, se complementan…
- Primero aplican prácticas XP y luego Scrum
  - Algunas de las prácticas de XP podrían utilizarse de manera aislada (incluso en el modelo de cascada!)
  - Después de ganar la confianza de TDD y refactoring la organización está más abierto a escuchar acerca de las alternativas de planificación (Scrum)
- Scrum primero y luego completar con XP
  - Las organizaciones que ha instalado correctamente scrum podrían apoyarse en las prácticas de ingeniería (XP)

UNIVERSIDAD DE LIMA | 60 años

Conceptos de diseño ágil

Diseño ágil

- Si la agilidad se trata de crear software en pequeños incrementos. Entonces:
  - ¿Cómo se puede diseñar el software?
  - ¿Cómo puede tomarse el tiempo para asegurarse de que el software tenga una buena estructura que sea flexible, mantenible y reutilizable?
  - Si construye en pequeños incrementos:
    - ¿no está realmente preparando el escenario para muchos desechos y reelaboración en nombre de la refactorización?
    - ¿No te vas a perder el panorama general?
- En un equipo ágil, el panorama general evoluciona junto con el software. Con cada iteración, el equipo mejora el diseño del sistema para que sea lo mejor posible para el sistema tal como es ahora.
- El equipo no pasa mucho tiempo pensando en los requisitos y necesidades futuras.
- Tampoco intentan construir hoy la infraestructura para respaldar las características que creen que necesitarán mañana. Más bien, se enfocan en la estructura actual del sistema, haciéndolo lo mejor posible.

Síntomas de un diseño deficiente

1. Rigidez—El sistema es difícil de cambiar porque cada cambio obliga a muchos otros cambios a otras partes del sistema.
2. Fragilidad: los cambios hacen que el sistema se rompa en lugares que no tienen una relación conceptual con la parte que se cambió.
3. Inmovilidad: es difícil separar el sistema en componentes que puedan reutilizarse en otros sistemas.
4. Viscosidad—Hacer las cosas bien es más difícil que hacer las cosas mal.
5. Complejidad innecesaria: el diseño contiene infraestructura que no agrega ningún beneficio directo.
6. Repetición innecesaria: el diseño contiene estructuras repetitivas que podrían unificarse bajo una sola abstracción.
7. Opacidad: es difícil de leer y comprender. No expresa bien su intención.

Smells y Principios

- Code smells: (Código que huele o apesta) es cualquier síntoma en el código fuente de un programa que posiblemente indica un problema más profundo.
- Un diseño smells es un síntoma, es algo que se puede medir objetivamente. A menudo, el smells es causado por la violación de uno o más de los principios SOLID.
- “What is Software Design?” — **Reeves** argumenta que el diseño de un sistema de software está documentado principalmente por su código fuente. Resulta que el artículo de Jack fue un presagio del **desarrollo ágil.**
- Un conjunto de diagramas UML puede representar partes de un diseño, pero no es el diseño. El diseño de un proyecto de software es un concepto abstracto.

¿Qué estimula el software a pudrirse?

- En entornos no ágiles, los diseños se degradan porque los requisitos cambian de formas que el diseño inicial no anticipó.
- A menudo, estos cambios deben realizarse rápidamente y pueden ser realizados por desarrolladores que no están familiarizados con la filosofía de diseño original.
- Entonces, aunque el cambio en el diseño funciona, de alguna manera viola el diseño original.
- Poco a poco, a medida que continúan los cambios, estas violaciones se acumulan y el diseño comienza a oler mal.
- De alguna manera debemos encontrar una manera de hacer que nuestros diseños sean resistentes a tales cambios y emplear prácticas que los protejan de la descomposición.

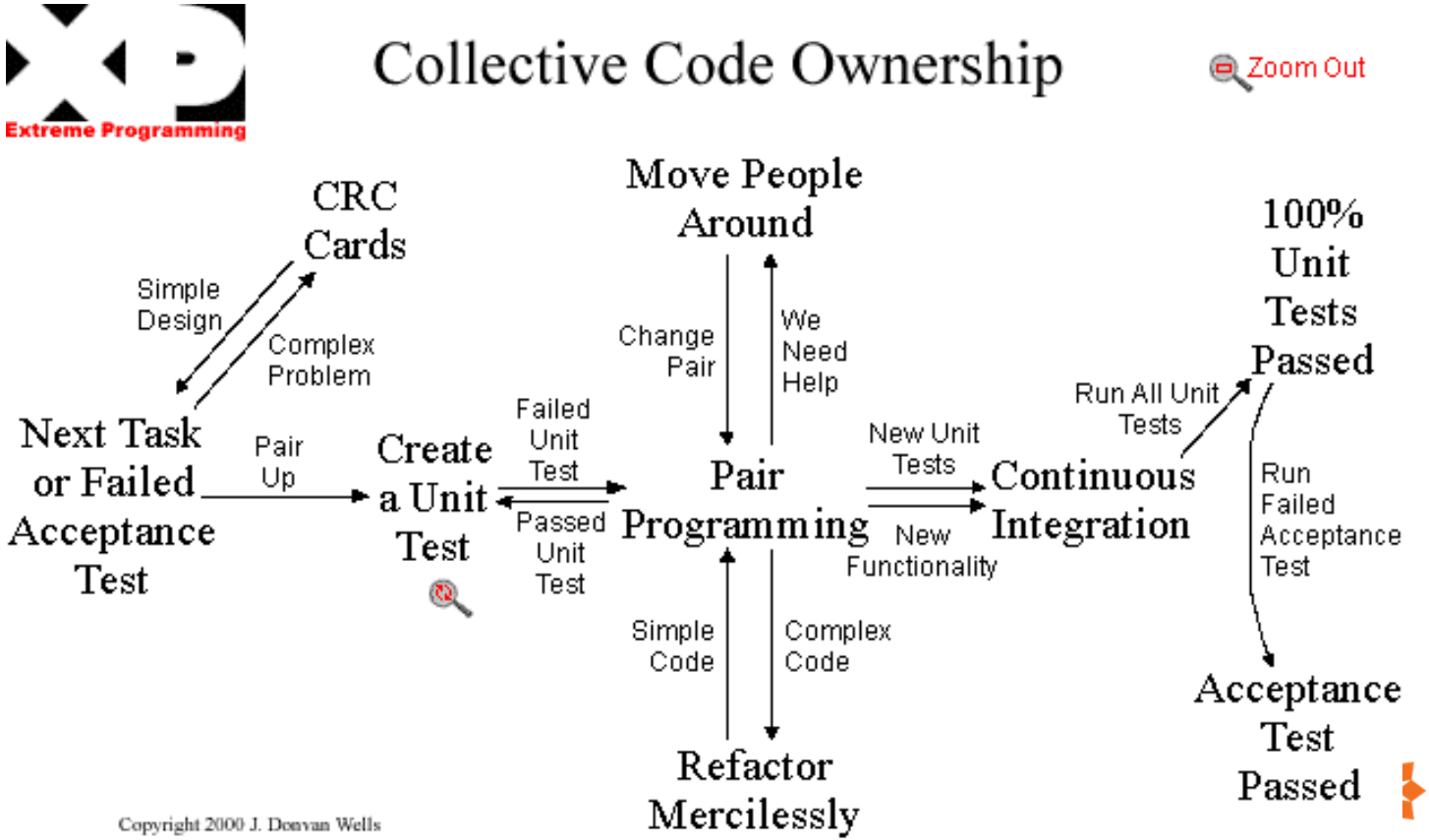
Los equipos ágiles no permiten que el software se pudra

- Un equipo ágil prospera con el cambio.
- El equipo invierte poco por adelantado; por lo tanto, no pertenece a un diseño inicial envejecido.
- Mantienen el diseño del sistema lo más limpio y simple posible, y lo respaldan con muchas pruebas unitarias y pruebas de aceptación.
- Esto mantiene el diseño flexible y fácil de cambiar.
- El equipo aprovecha esa flexibilidad para mejorar continuamente el diseño de modo que cada iteración finalice con un sistema cuyo diseño sea lo más apropiado posible para los requisitos de esa iteración.

XP – Design

- Sigue el principio manténgalo simple (KIS - keep it simple)
- Fomenta el uso de **CRC (class-responsibility-collaborator) cards** como un mecanismo eficaz para pensar en el software en un contexto orientado a objetos.
- Para problemas de diseño difíciles, XP sugiere la creación de "**spike solutions**", un diseño por prototipos.
- Fomenta la "refactoring", un refinamiento iterativo del diseño interno del programa.
- El diseño ocurre tanto antes como después de que comience la codificación.

De los requisitos al diseño - CRC Card

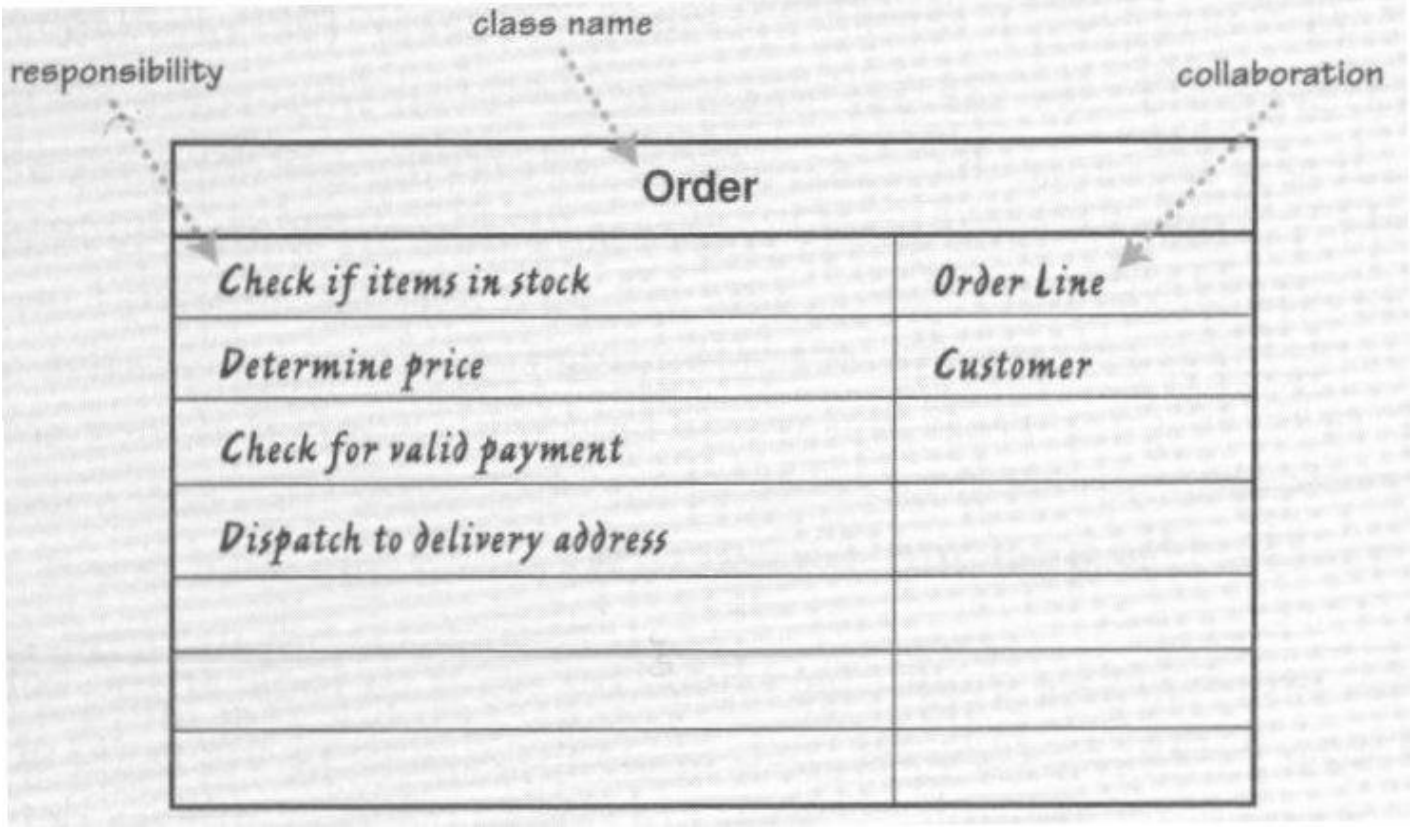


UNIVERSIDAD DE LIMA

Introducción CRC Cards

- Class Responsibility Collaboration
- Desarrollado en los 80’s
- Usado para:
  - Analizar el dominio del problema
  - Descubrir diseño orientado a objetos
- Diseño orientado a objetos
  - Objetos tienen estado y comportamiento
  - Objetos delegan responsabilidades
  - ”Think objects”

CRC Card Template



Proceso - CRC Card

- Básico: Simular la ejecución de escenarios de casos de uso / historias de usuarios
- Pasos:
  - Lluvia de ideas - class/object/component
  - Asignar class/object/component a personas (grupo hasta 6 personas)
  - Ejecuta los escenarios uno por uno
  - Añadir nuevas class/object/component según sea necesario
  - Añadir nuevas responsabilidades
  - delega a otras clases o personas

Ejemplo Biblioteca: User story

Préstamo de Libros:

Cómo usuario **necesito** prestarme un libro **para** complementar los estudios.

Criterios de aceptación

- El prestado de un libro por un usuario es registrado en la Biblioteca
- El usuario no debe prestarse más de 10 libros
- El usuario no puede prestarse un libro si tiene otro libro con la fecha vendida de préstamo.

Ejemplo Biblioteca - Registrar el préstamo de un Libro a un usuario

- Conjunto de tarjetas CRC iniciales: Aplicación Biblioteca, Usuario, Libro

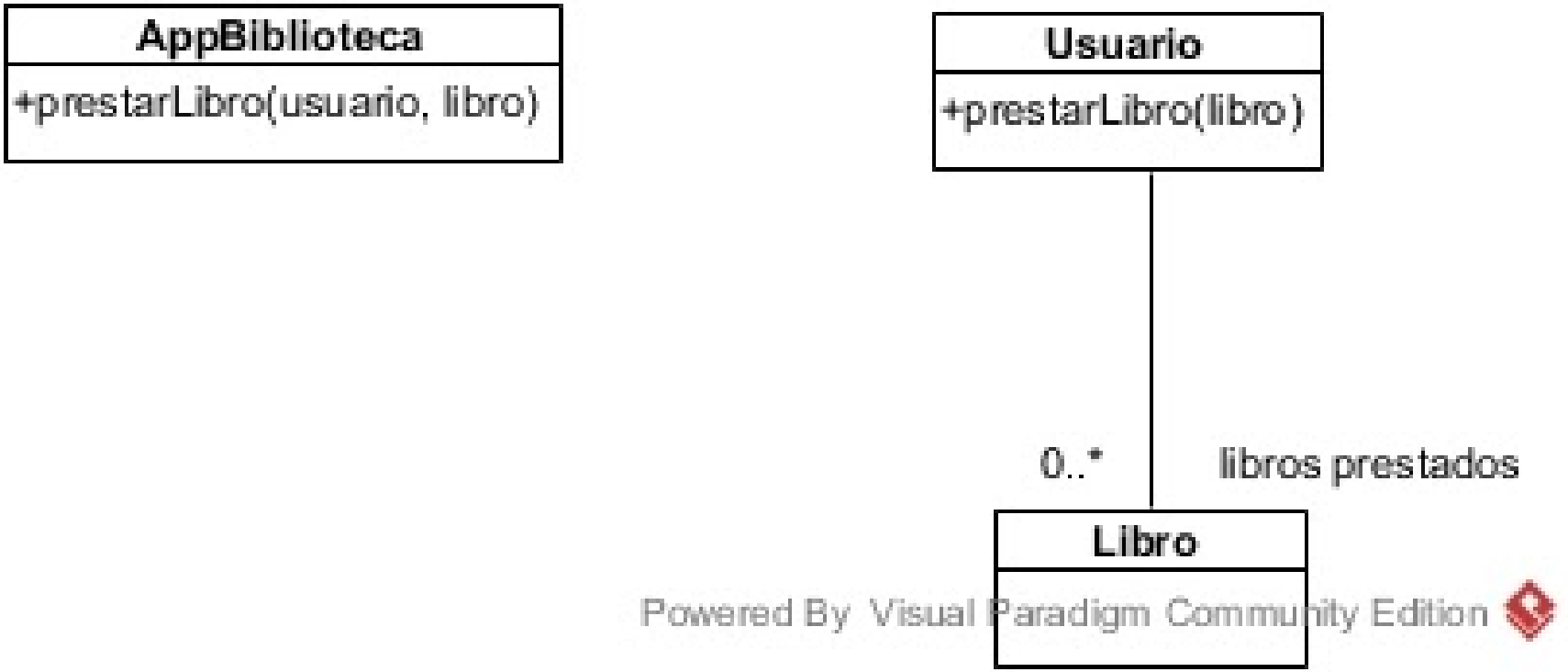
Aplicación Biblioteca	
Registrar el préstamo de un libro a un usuario	Usuario Libro

Usuario	
Prestarse un Libro Conocer los Libros prestados	

Libro	

UNIVERSIDAD DE LIMA

Ejemplo Biblioteca - Prestar un Libro a usuario registrado



Ejemplo Biblioteca - Prestar no más de 10 libros

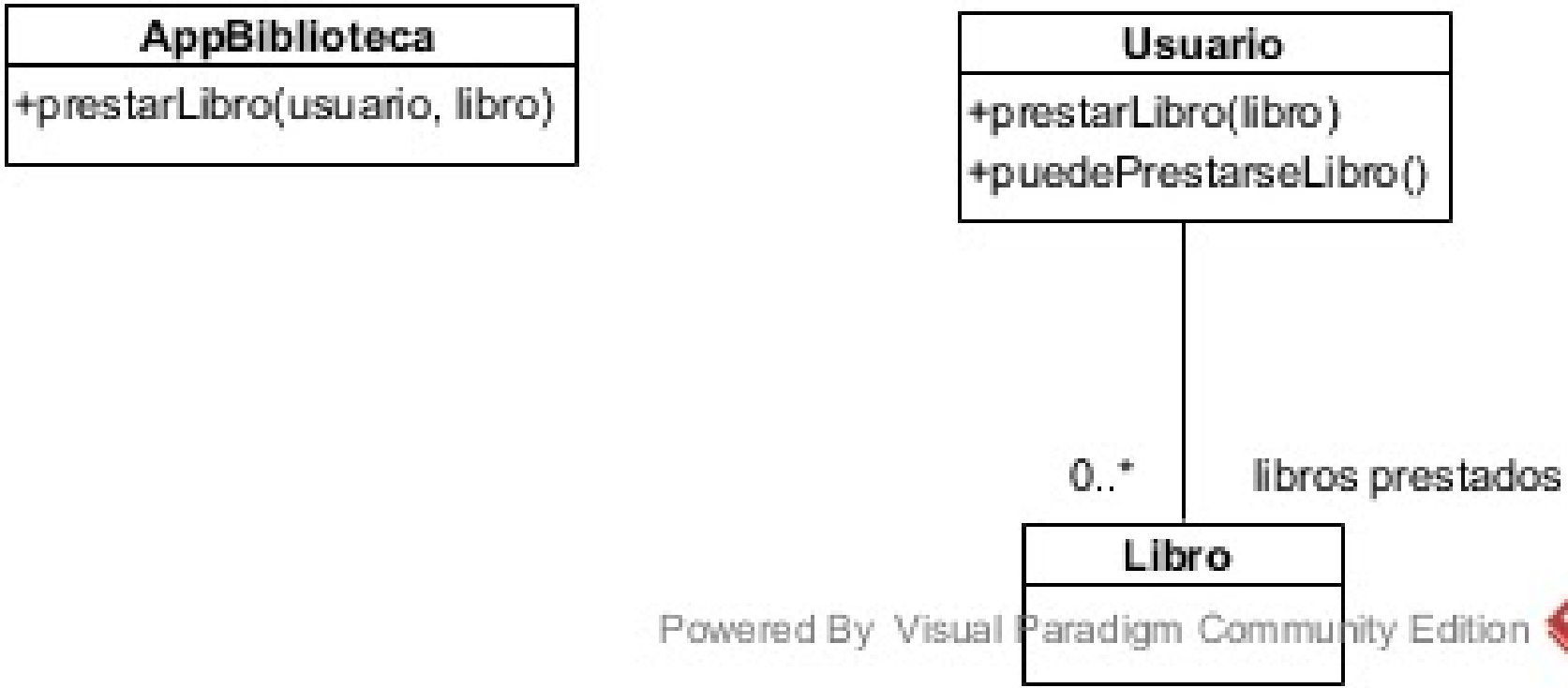
Aplicación Biblioteca	
Registrar el préstamo de un libro a un usuario	Usuario Libro

Libro	

Usuario	
Prestarse un Libro Conocer los Libros prestados **Conocer si puede prestarse**	

UNIVERSIDAD DE LIMA

Ejemplo Biblioteca - Prestar no más de 10 libros



Ejemplo Biblioteca - Libro con fecha de devolución vencida

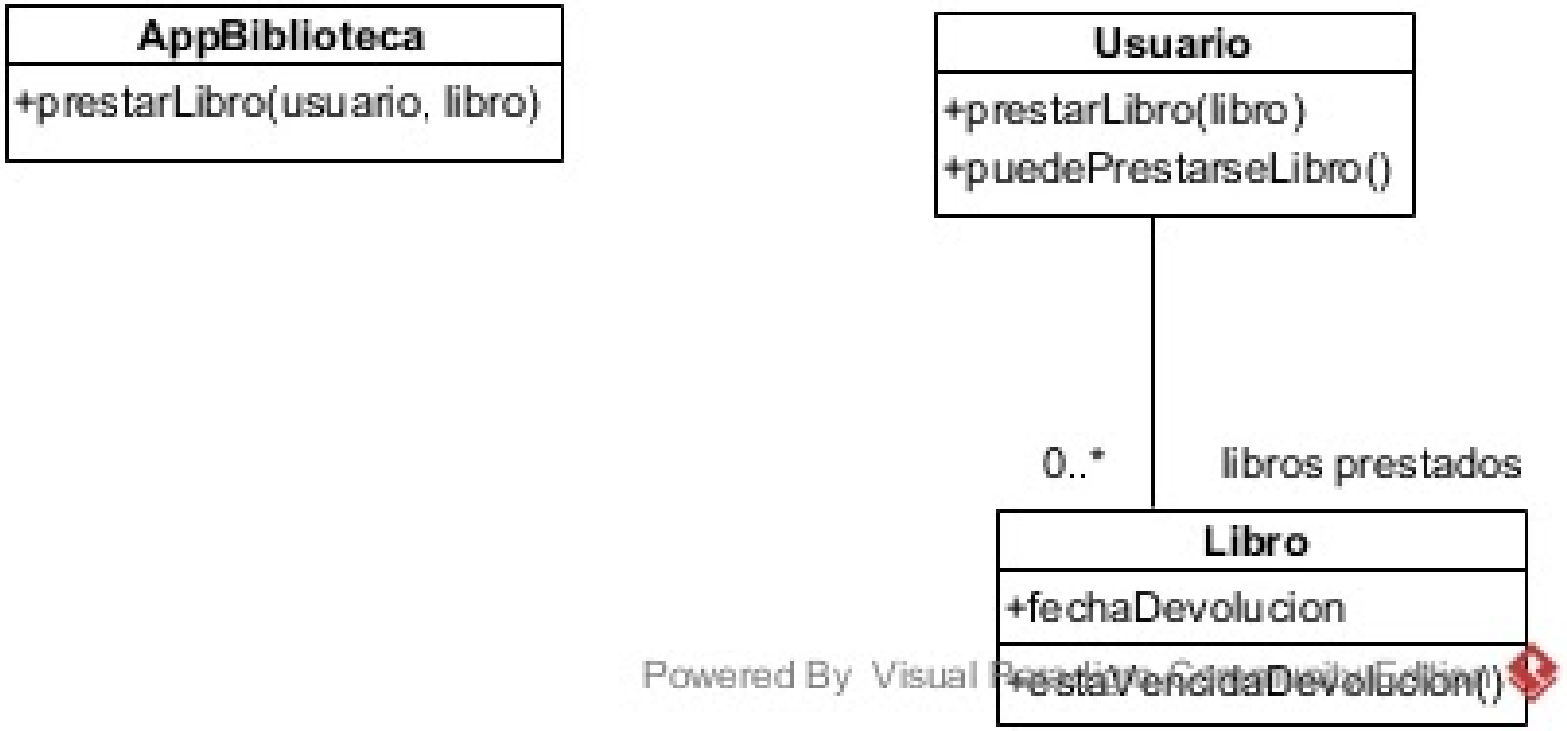
Aplicación Biblioteca	
Registrar el préstamo de un libro a un usuario Conocer la fecha actual	Usuario Libro Fecha

Usuario	
Prestarse un Libro Conocer los Libros prestados **Conocer si puede prestarse un libro**	

Libro	
Conocer si la fecha de devolución está Vencida Conocer la fecha de devolución	Fecha

Fecha	
Comparar fechas	

Ejemplo Biblioteca - Libro con fecha de devolución vencida



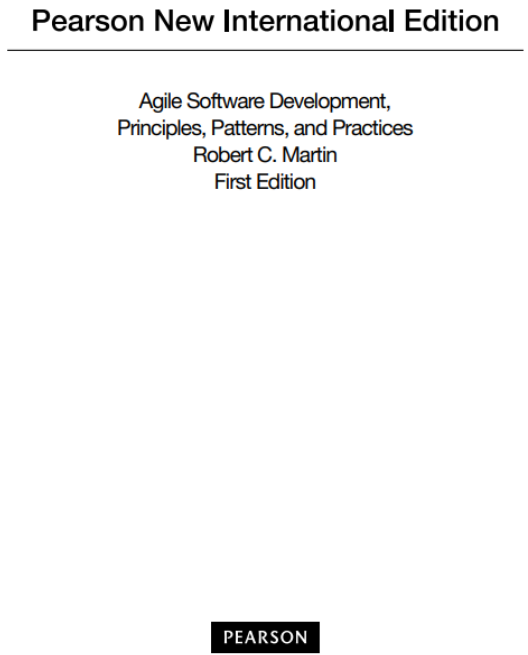
UNIVERSIDAD DE LIMA | 60 años

Principios del diseño de paquetes

Las clases, si bien son una unidad muy conveniente para organizar aplicaciones pequeñas, son demasiado finas para ser utilizadas como la única unidad organizativa para aplicaciones grandes.

Principios del diseño de paquetes

- Se necesita algo "más grande" que una clase para ayudar a organizar aplicaciones grandes. Ese algo se llama ***paquete***.
- Principios.
  - Package cohesion
  - Package coupling
  - Dependency Management (DM) metrics



UNIVERSIDAD DE LIMA

Diseñando con Paquetes

- En UML, los paquetes se pueden usar como contenedores para grupos de clases.
- Pero las clases a menudo tienen dependencias en otras clases. Por lo tanto, los paquetes tendrán relaciones de dependencia entre sí.
- Esto plantea una gran cantidad de preguntas.
  - ¿Cuáles son los principios para asignar clases a paquetes?
  - ¿Qué principios de diseño rigen las relaciones entre los paquetes?
  - ¿Deberían diseñarse los paquetes antes de las clases (de arriba hacia abajo)? ¿O deberían diseñarse las clases antes que los paquetes (de abajo hacia arriba)?
  - ¿Cómo se representan físicamente los paquetes? ¿En C++? ¿En Java? ¿En el entorno de desarrollo?

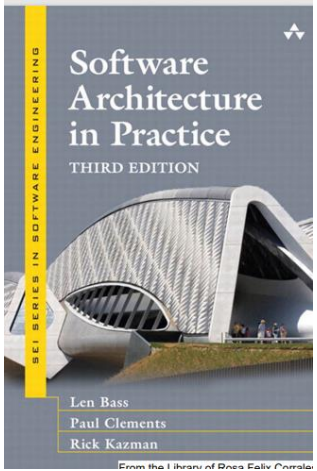
UNIVERSIDAD DE LIMA | 60 años

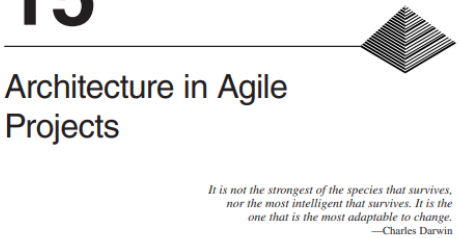
Arquitectura en proyectos ágiles

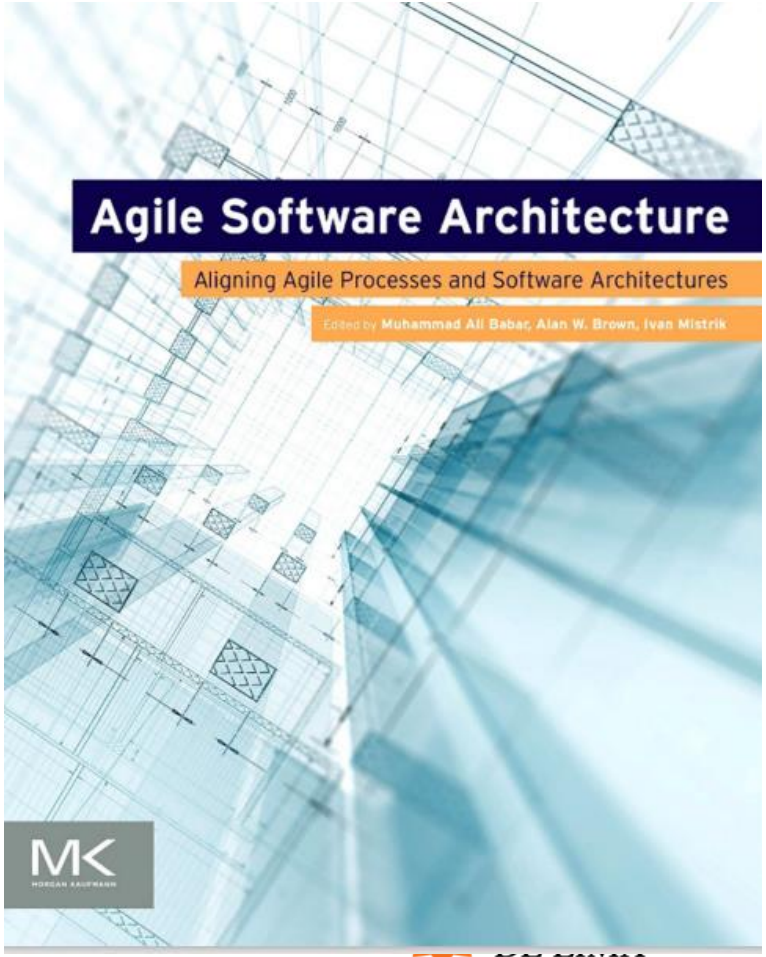
Arquitectura del software con métodos ágiles

Agile Architecture

- Agile Architecture es un conjunto de valores, prácticas y colaboraciones que respaldan el diseño y la arquitectura activos y evolutivos de un sistema.







Scrum y Diseño de arquitectura

Sprint	Iterative nature of general model of software architecture (SA) design with backlogs of architectural concerns to be addressed
Sprint planning	Prioritizing architecturally significant requirements for each iteration
Sprint review	Architectural review
Daily meetings	Sharing architecture rationale and knowledge in architecture group meetings

Referencias

1. Bourque, P., & Fairley, R. E. (2014). Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0. IEEE Computer Society Press.
2. Andersson, E. A., Greenspun, P., & Grumet, A. (2006). Software engineering for Internet applications (No. Sirsi) i9780262511919). MIT press.
3. Kniberg, H. (2007). Scrum y XP desde las trincheras. Estados Unidos: C4Media.