

UNIDAD 5: MÉTODOS ÁGILES

Agenda

1. Manifiesto ágil y principios
2. Modelo tradicional vs modelo ágil
3. Métodos ágiles, DSDM, XP, Crystal, FDD, Scrum
4. Introducción a Design Thinking,
5. Lean Startup, Kanban, Scrum. BDD (Behavior Driven Design).
6. Historias de usuario INVEST. Formato Connextra. Lenguaje Gherkin

Antecedentes

- A inicios de los '90, el desarrollo de software se había transformado en un cuello de botella.
- Este problema, la Crisis del Software, se muestra en el CHAOS Report (1994).
- Se intentó mejorar la creación de software usando ideas de vinculadas con procesos de manufactura como control estadístico y mejora de procesos (**CMM** en 1993), automatización y robotización (**CASE**, **RUP** en 1996 y la generación automática de código).
- Luego en los 90 surgieron las llamadas "Metodologías de Desarrollo de Software de peso liviano" (Lighweight)
 - Crystal (1992)
 - **Scrum** (1995)
 - Programación por Pares (1995)
 - Feature Driven Development (1997)
 - Desarrollo Adaptativo (1999)
 - **Extreme Programming** (1999)
 - Integración continua (1999)

Línea de tiempo

- 1930 - **Ciclo PDCA** Walter Shewhart propone el ciclo de "Planear", "Hacer", "Estudiar" y "Actuar", un concepto que luego fue difundido por **Deming**.
- 1940 - **Kanban**, Sistemas de Producción de Toyota y el **Lean Manufacturing**: Taiichi Ohno inventa el método Kanban en Toyota. El Lean Manufacturing es una fuente de inspiración y precursor del movimiento ágil.
- 1992 – Crystal: Alistair Cockbur presenta los Métodos Crystal, el punto de inicio de la evolución de las metodologías de desarrollo de software que eventualmente resultaron en lo que hoy se conoce como el movimiento ágil.
- 1993 – Refactorización Bill Opdyke presenta el concepto de "Refactorización" en el paper "Creando Superclases Abstractas por medio de la Refactorización".
- 1995 - **Programación en Pares** Es un concepto que fue simultáneamente ideado, pero de forma independiente por varios autores (Jim Coplien y Larry Constantine)
- 1995 – **Scrum** ideado por Ken Schwaber y Jeff Sutherland
- 1997 - **Feature Driven Development (FDD)** explicado por medio de la publicación del libro "Modelado Java a Colores con UML: Componentes y Procesos Empresariales", cuyos coautores son Jeff De Luca y Peter Coad.
- 1999 Desarrollo de Software Adaptativo Jim Highsmith formalizó el concepto de Desarrollo de Software Adaptativo y publicó su libro del mismo nombre. La idea creció y evolucionó hacia las metodologías de Desarrollo Rápido de Aplicaciones (RAD)
- 1999 - **Extreme Programming (XP)** Kent Beck también formuló los conceptos de Historias de Usuario y Planificación de Releases.

Línea de tiempo

- 1999 - Integración Continua Kent Beck definió este concepto también, pero fue un paper de **Martin Fowler** el que lo popularizó.
- 2001 - **El Manifiesto Ágil**
- 2002 - **Test Driven Development (TDD)** El concepto se originó el enfoque de "Probar primero" asociado a la Programación Extrema (XP)
- 2002 - **Planning Poker** se publica la técnica de Planning Poker del autor James Greening
- 2003 - **Lean Software Development** Mary y Tom Poppendieck presentan su obra "Lean Software Development". Presenta 7 principios: Eliminar desperdicio, amplificar el aprendizaje, Decidir tan tarde como sea posible, entregar lo más rápido posible, dar poder al equipo (empowerment), construir integridad y ver la totalidad.
- 2006 - **Behavior Driven Development (BDD)** Dan North presenta su obra "Behavior Driven Development", un método que combina las principales ideas y técnicas del TDD con las ideas del Diseño guiado por dominio y el Análisis y Diseño orientado a objetivos.
- 2007 – **Retrospectivas** Esther Derby y Diana Larsen escriben su obra "Agile Retrospectives", estableciendo las reuniones retrospectivas como práctica ágil estándar.
- 2007 – **Kanban**: David Anderson presenta su obra "Kanban", adaptando el Kanban para el desarrollo de software. El método se enfoca en la entrega "justo a tiempo" y en no sobrecargar a los desarrolladores de software, tal como su precursor el Kanban para manufactura perfeccionado por Toyota
- 2009 - Manifiesto de la Artesanía de Software (**Software Craftmanship**)
- 2009 - **Lean Startup** : Eric Ries escribe su obra "Lean Startup". Es una metodología mayormente teórica para el desarrollo de empresas y productos.



<https://agile-lounge.com/18-years-of-agile-manifesto-for-software-development/>

Manifiesto Agil

En 2001, 17 personas representantes de formas alternativas de desarrollo de software (convocados por **Kent Beck**) deciden unir dichas metodologías bajo el paraguas de Desarrollo Ágil de Software como una manera para llegar a la mayoría de los desarrolladores de software.

En la reunión se acuñó el término “*Métodos Ágiles*” para definir a los que estaban surgiendo como alternativa a los modelos formales, (**CMMSW, PMI, SPICE**) que los consideraban excesivamente “pesados” y rígidos por su carácter normativo y fuerte dependencia de planificaciones detalladas, previas al desarrollo.

Manifiesto Agil

- Los integrantes de la reunión resumieron en cuatro postulados lo que ha quedado denominado como “Manifiesto Ágil”, que son los principios sobre los que se basan estos métodos.

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.



Manifiesto Ágil

- Los integrantes de la reunión resumieron en cuatro postulados lo que ha quedado denominado como “Manifiesto Ágil”, que son los principios sobre los que se basan estos métodos.



Principios Ágiles

Manifiesto Ágil



Principios Agiles

1. la satisfacción del cliente mediante entregas tempranas y continuas de software que funcione;
2. requerimientos cambiantes en cualquier etapa del proyecto;
3. Entregar software funcionando frecuentemente. Desde un par de semanas hasta un par de meses
4. Participación activa del cliente;
5. equipos de desarrollo motivados
6. comunicación efectiva (cara a cara)
7. Software funcionando es la medida principal de progreso
8. Promover el desarrollo sostenible
9. Atención a la excelencia técnica y buen diseño
10. Simplicidad
11. Las mejores arquitecturas, requerimientos y diseños emergen de equipos auto organizados
12. A intervalos regulares el equipo debe ajustar su desempeño

<https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/>



UNIVERSIDAD
DE LIMA



Modelo tradicional vs modelo ágil


- Las metodologías tradicionales son orientadas por planeación.
- Inician el desarrollo de un proyecto con un riguroso proceso de elicitación de requerimientos, previo a etapas de análisis y diseño.
- Con esto tratan de asegurar resultados con alta calidad delimitados por un calendario.
- Las metodologías ágiles tienen dos diferencias fundamentales con las metodologías tradicionales:
 1. Los métodos ágiles son adaptativos –no predictivos-.
 2. La segunda diferencia es que las metodologías ágiles son orientadas a las personas (no orientadas a los procesos)

Metodología Ágil	Metodología No Ágil
Pocos Artefactos	Más Artefactos
Pocos Roles	Más Roles
No existe un contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
Cliente es parte del equipo de desarrollo (además in-situ)	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (< 10 integrantes) y trabajando en el mismo sitio	Grupos grandes
Menos énfasis en la arquitectura	La arquitectura es esencial

Métodologías ágiles

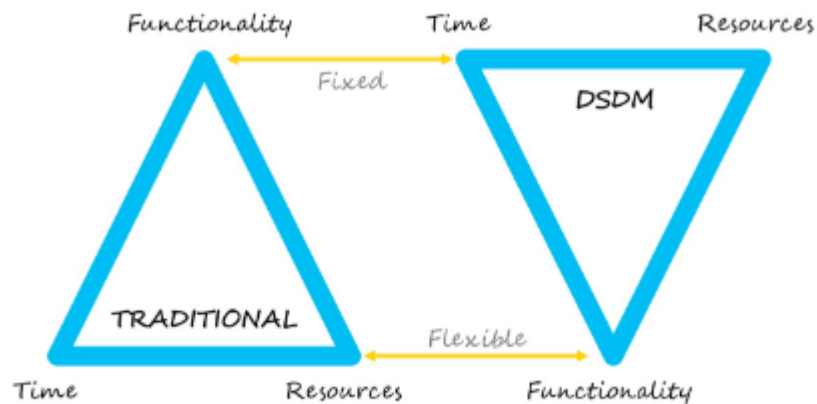
- Las metodologías más populares en los documentos científicos entre el 2003 y el 2007 fueron **XP** y **Scrum**
 - XP se enfoca en prácticas de desarrollo
 - Scrum apunta a la administración de proyectos

- Las más relevantes:

- 
1. **Scrum,**
 2. **Extreme Programming [XP]**
 3. **Dynamic System Development Method [DSDM],**
 4. **Crystal, Adaptive Software Development [ASD]**
 5. **Feature-Driven Development [FDD]**
 6. **Agile Modelling [AM]**

Dynamic Systems Development Method - DSDM

- Es un marco de trabajo creado para entregar la solución correcta en el momento correcto.
- DSDM es la metodología ágil más próxima a los **métodos formales**;
- La implantación de un modelo DSDM en una organización le permite alcanzar lo que **CMM** consideraría un nivel 2 de madurez
- Utiliza un ciclo de vida iterativo, fragmenta el proyecto en periodos cortos de tiempo y define entregables para cada uno de estos periodos
- Los principios de DSDM son:



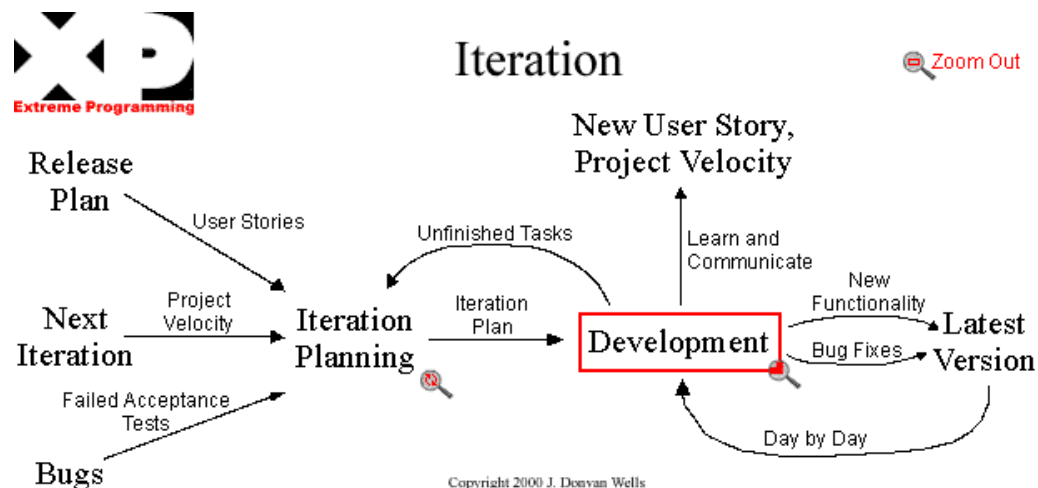
www.agile-scrum.be

- la necesidad del negocio como eje central;
- las entregas a tiempo;
- la colaboración;
- nunca comprometer la calidad;
- construir de modo incremental sobre una base sólida;
- el desarrollo iterativo;
- la comunicación clara y continua;
- la demostración de control

<https://www.agilebusiness.org/what-is-dsdm>

Extreme Programming : XP

- XP es la metodología ágil más conocida
- Fue desarrollada por **Kent Beck** buscando guiar equipos de desarrollo de software pequeños o medianos, entre dos y diez desarrolladores, en ambientes de requerimientos imprecisos o cambiantes
- XP tiene como base cinco valores: Simplicidad, Comunicación, Retroalimentación, Respeto y Coraje
- Las prácticas de esta metodología se derivan de sus valores y principios y están enfocadas en darle solución a las actividades básicas de un proceso de desarrollo: escribir código, realizar pruebas, escuchar (planear) y diseñar.
- Las prácticas de XP incluyen:



- planning game,
- pequeñas entregas,
- diseño simple,
- programación en pareja,
- pruebas,
- refactoring,
- integración continua,
- propiedad común del código,
- paso sostenible,
- cliente en sitio,
- metáfora y estándares de código.



Crystal

- Se basa en los conceptos de Rational Unified Process (RUP) y está compuesta básicamente por Crystal Clear, Crystal Yellow, Crystal Orange y Crystal Red
- El nivel de opacidad del color en el nombre indica un mayor número de personas implicadas en el desarrollo, un mayor tamaño del proyecto y, por lo tanto, la necesidad de mayor control en el proceso
- La filosofía de Crystal define el desarrollo como un juego cooperativo de invención y comunicación cuya meta principal es entregar software útil, que funcione, y su objetivo secundario, preparar el próximo juego
- Dos reglas:
 1. los ciclos donde se crean los incrementos no deben exceder cuatro meses;
 2. es necesario realizar un taller de reflexión después de cada entrega para afinar la metodología.



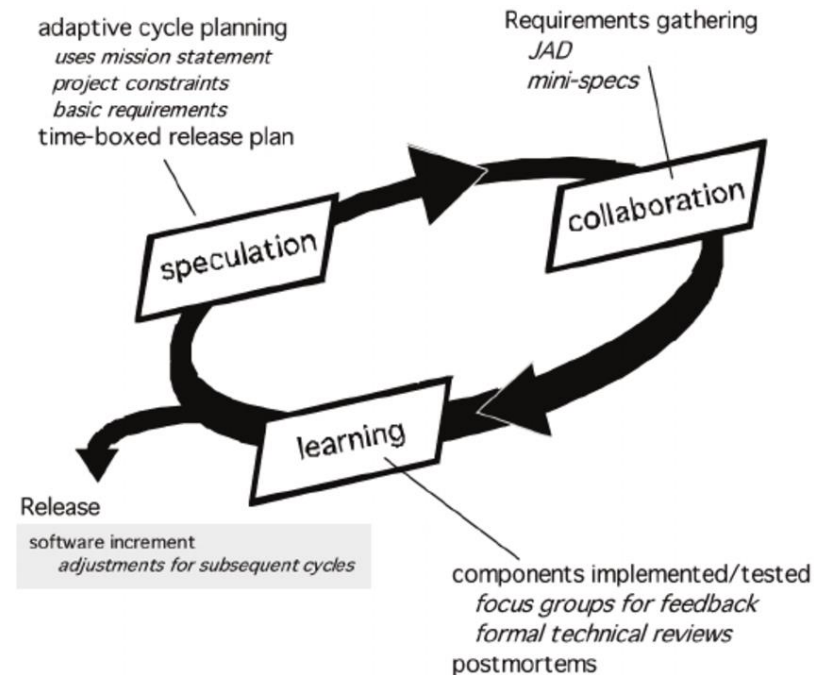
Crystal

Personas
diferentes

Roll- Último Responsable	Productos
Sponsor (patrocinador, quien financia)	La declaración de la Misión con el Trade-off de Prioridades.
Equipo	La estructura y las convenciones del equipo. Los resultados del trabajo de reflexión.
Coordinador, con ayuda del equipo	El Mapa del Proyecto, El Plan de Entrega, El Estado del Proyecto, La Lista de Riesgo, El Plan y Estado de la Iteración, La visualización del Calendario-Cronograma.
Experto del negocio y usuario experto juntos	La lista de los objetivos por actor: Los Casos de Uso. El archivo de Requerimientos: El modelo del rol del usuario.
Líder de diseño (diseñador líder)	La descripción de la Arquitectura
Diseñadores-programadores (incluyendo al líder de diseño)	Borradores de pantalla, Modelo de Dominio Común, Esquemas y notas de diseño, Código fuente, Código de Migración, Las Pruebas, El sistema empaquetado.
Tester	Reporte de errores en ese momento

Adaptative Software Development - ASD

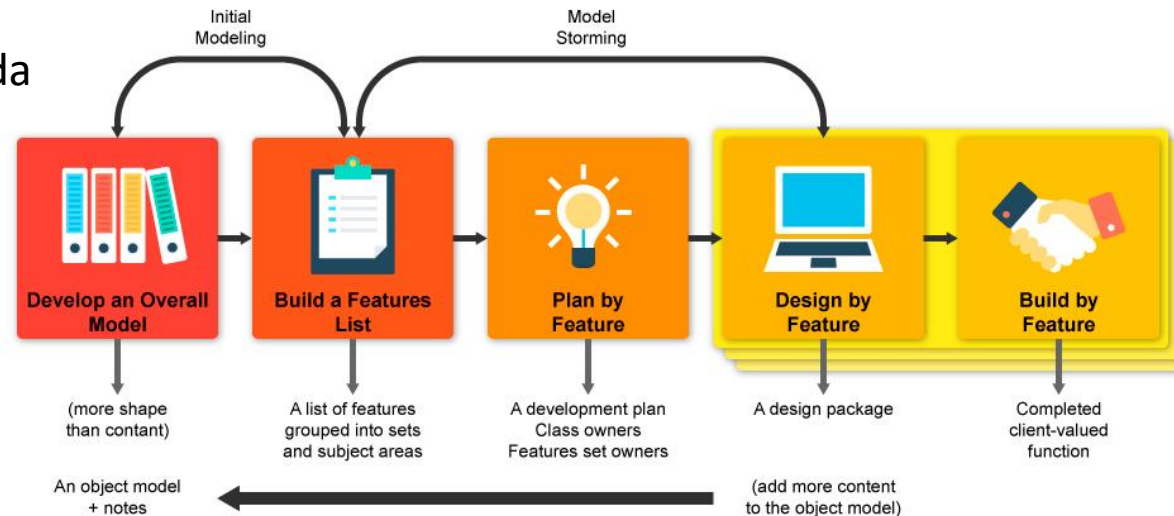
- ASD tiene como fundamento la teoría de sistemas adaptativos complejos. Por ello, interpreta los proyectos de software como sistemas adaptativos complejos compuestos por agentes (los interesados), entornos (organizacional, tecnológico) y salidas (el producto desarrollado).
- Sus principales características son:
 1. Iterativo.
 2. Orientado a los componentes software mas que a las tareas.
 3. Tolerante a los cambios.
 4. Guiado por los riesgos.
 5. La revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo.
- El ciclo utilizado por ASD es conocido como: **Especular-colaborar-aprender**, el cual está dedicado a un constante aprendizaje y colaboración entre desarrollador y cliente



Feature-Driven Development - FDD

- Las prácticas que FDD pregona son:
 1. el modelado de objetos de dominio (domain object modeling),
 2. el desarrollo por características,
 3. Class (code) ownership,
 4. los equipos de características o Feature Teams,
 5. las inspecciones,
 6. la construcción regular de planificación (Regular Build Schedule),
 7. la gestión de configuración
 8. los reportes y visibilidad de los resultados

- Su ciclo de vida



Design Thinking

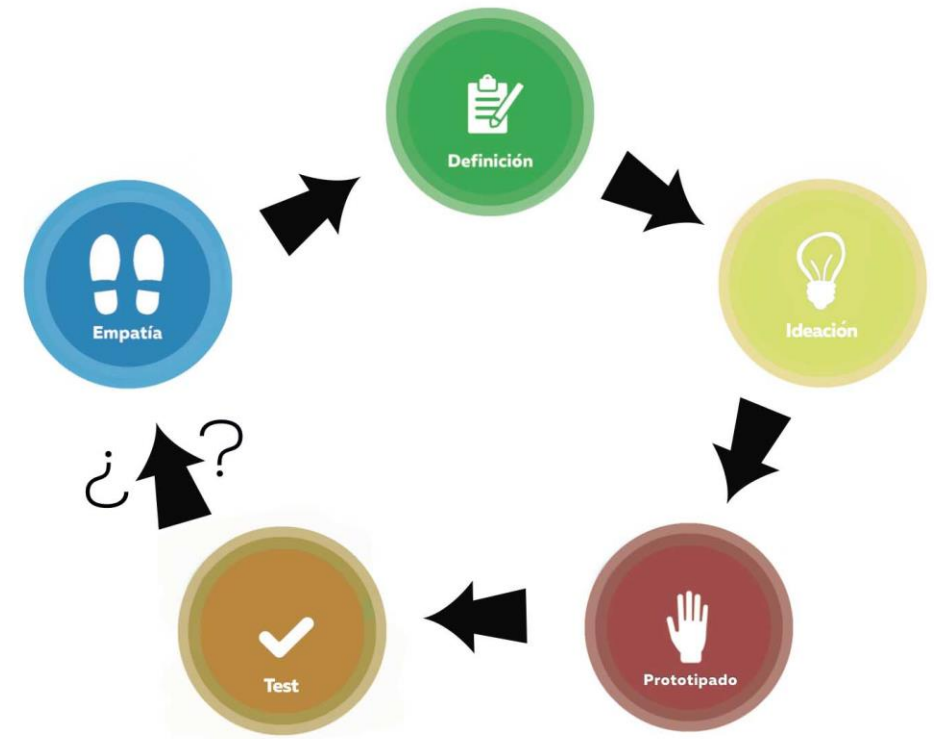
Design Thinking o Pensamiento de Diseño es una metodología para la resolución de problemas abiertos y complejos aplicable a cualquier ámbito que requiera un enfoque creativo.

La historia más conocida del Design Thinking tiene pocos años. Sin embargo, podríamos decir que los inicios de **esta metodología tienen lugar antes de 1960**. Y aparecen en cuestiones relativas al diseño industrial.

Pero es en los noventa cuando el Design Thinking se convierte definitivamente en lo que hoy conocemos. **En 1991 nace [IDEO](#)**, la consultora que revoluciona el mundo del diseño. Y convierte esta metodología en la más utilizada para generar innovación. Muy pronto pasaron por IDEO empresas de la talla de Apple, cuyo ratón fue diseñado desde los principios del Design Thinking.

Etapas de Design Thinking

1. **Empatizar:** En la fase de Empatizar, definimos el arquetipo de usuario al que vamos a dirigirnos, establecemos nuestros objetivos de investigación y, a partir de ellos, decidimos qué técnicas de recopilación de información utilizaremos.
2. **Definir.** organizamos toda la información recopilada para identificar todas las áreas de oportunidad desde la que podemos ofrecer soluciones relevantes para los deseos y necesidades para el usuario.
3. **Idear:** Una vez establecido el reto pasamos a la parte de diseño de la solución. En esta parte del proceso, la primera fase por la que pasaremos será la de idear.
4. **Prototipar:** A partir de las ideas generadas, se lleva a cabo una selección, y éstas pasan a prototiparse. La fase de prototipado es aquella en la que damos forma a las ideas, las tangibilizamos.
5. **Validación o test.** La fase de validación exige preparación. Tendremos que establecer los objetivos, construir la guía y, por último, mostrar al usuario nuestra solución.



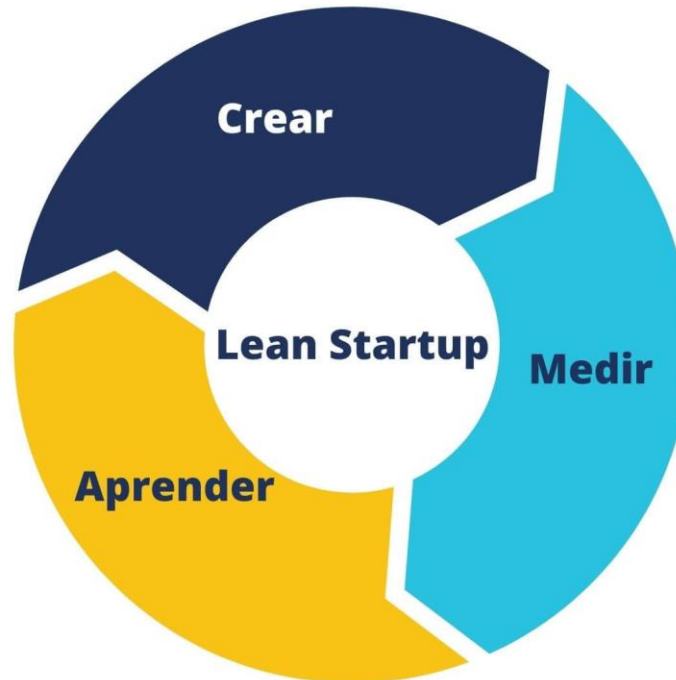
Características Design Thinking

Las **características claves** del Design Thinking son:

- Diseño centrado en las personas: valor de la empatía.
- Experimentación y prototipado: se trata de una parte integral del proceso de innovación. Se prototipa para aprender y pensar.
- Orientado a la acción.
- «Muéstralo, no lo cuentes únicamente»: genera experiencias, cuenta historias, sé visual.
- El poder de la iteración: ciclo tras ciclo llegamos a una mejor solución.

Lean Startup

- El método, o modelo “**lean startup**” surgido en el Silicon Valley y difundido por Eric Ries. El concepto que fue desarrollado por Eric Ries en 2008 en su obra “**El método Lean Startup: Cómo crear empresas de éxito utilizando la innovación continua**”, tiene como objetivo reducir el **tiempo y el costo a la hora de crear una empresa**, startup o un producto... ¿cómo? utilizando las hipótesis y la experimentación.



Fase de Lean Startup

1. Construir: plantear una hipótesis y validarla

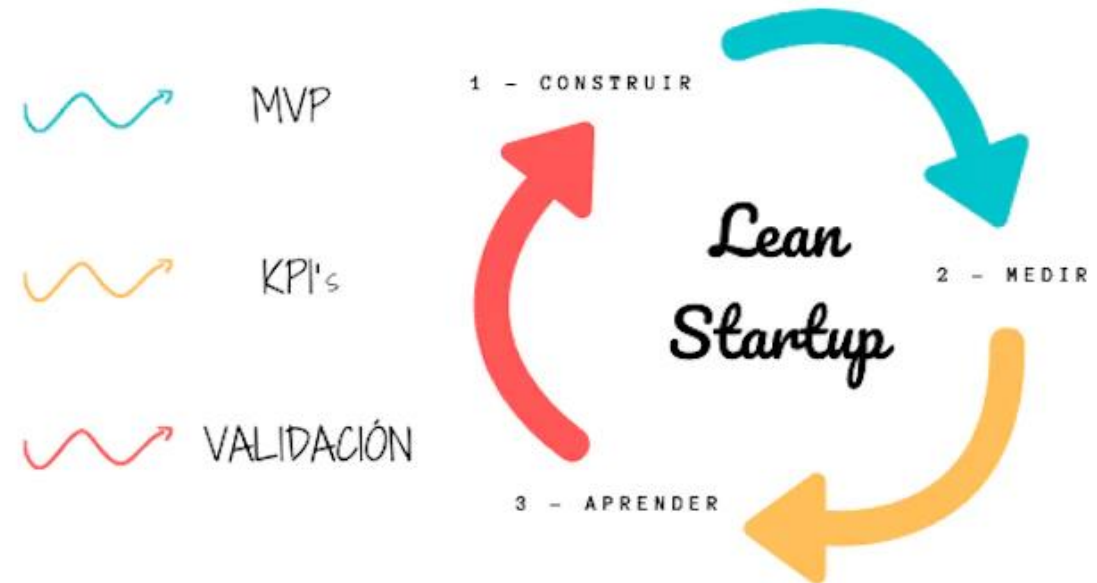
Esta etapa consiste en planificar tu lanzamiento y desarrollar una hipótesis formal de diseño para el Producto Mínimo Viable (MVP) y ponerlo a prueba.

2. Medir: constatar la pertinencia de la hipótesis

El objetivo de la etapa de medición es obtener información sobre la cual basarse para decidir si **el producto o servicio tiene el potencial necesario para beneficiar a tu negocio**. La forma de llevar a cabo dicha medición, se hace a partir de la implementación de Indicadores Clave de Desempeño (KPIs).

3. Aprender: experiencia validada, pivote y mejora continua

A partir de la retroalimentación recogida gracias al **estudio de mercado** y a la información recolectada por medio de tus **métricas clave**, ahora puedes realizar los cambios y ajustes que sean necesarios, con miras a la **mejora continua de tu oferta**.



Kanban

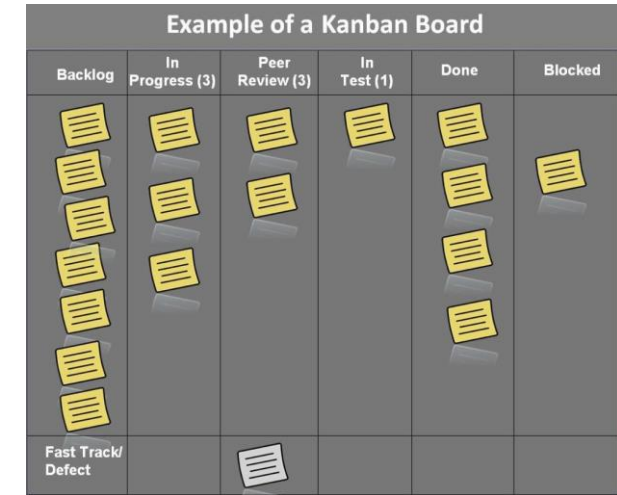
- Kanban tiene su origen en Japón, dentro de Toyota y nació para aplicarse a los procesos de fabricación de automóviles.
- El Kanban se basa en el método **Just in Time** que trataba de dividir el proceso en fases delimitadas que debían cumplir los objetivos para pasar a la siguiente fase, dando lugar así a la calidad.
- La metodología Kanban es un **método visual que se utiliza para controlar las tareas a través de su división por fases hasta su finalización.**
- Este método ha pasado a formar parte de las conocidas como **metodologías ágiles**, las cuales tratan de gestionar el trabajo y adaptarlo a las condiciones de los proyectos.



Principios y reglas del Kanban

El método está basado en cuatro pilares básicos:

1. Se puede implantar en el momento, no es necesario empezar un proyecto de nuevo.
2. Se persigue el cambio y evolución del equipo.
3. Respetar los roles, responsabilidades y cargos.
4. Alentar al liderazgo.



El método está basado en tres Reglas:

1. **Visualizar el trabajo** : Kanban trata de visualizar en una pizarra la división de tareas. La pizarra tiene tantas columnas como tareas haya. En cada columna se establecen las tareas en forma de “post-it”.
2. **Limitar el trabajo**: Es importante definir el número máximo de tareas que se pueden realizar en cada fase. La idea principal de esta regla es que los trabajadores se centren en cerrar tareas, no empezar nuevas.
3. **Medir el tiempo**: Se debe medir el tiempo que se tarda en cerrar cada tarea, además de tener en cuentas otras métricas importantes como el tiempo desde que se empieza una tarea hasta que se entrega al cliente. Es necesario para controlar y mejorar continuamente.

Pasos para implementar Kanban

1. **Prepara a tu equipo de trabajo.** Añadir a tu empresa el método Kanban supondrá un gran cambio en el personal.
1. **Visualizar el flujo de trabajo.** Debes diseñar el trabajo en la pizarra. Dividir los proyectos en las fases necesarias. Con una pizarra/tablero y post-it, o con un software capacitado de realizar esta labor. Algunos de los más recomendados por el mercado son Kanban Tool, Trello, JIRA Greenhopper, Cardmapping, Tablero Kanban Online y Targetprocess.
1. **Delimita el número de tareas en curso.** No sirve de mucho empezar varias tareas y dejarlas a mitad. Lo mejor es limitar el número de tareas iniciadas.
1. **Controla el trabajo.** Hay que revisar de manera constante el funcionamiento.
2. **Mejora a tu equipo continuamente.**



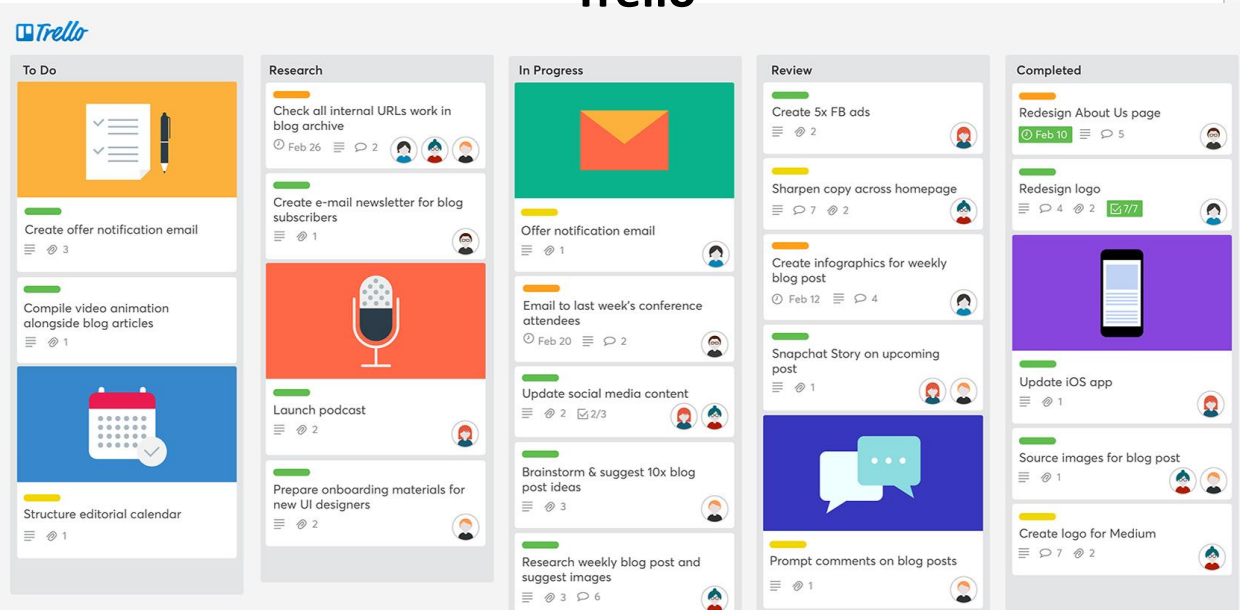
UNIVERSIDAD
DE LIMA



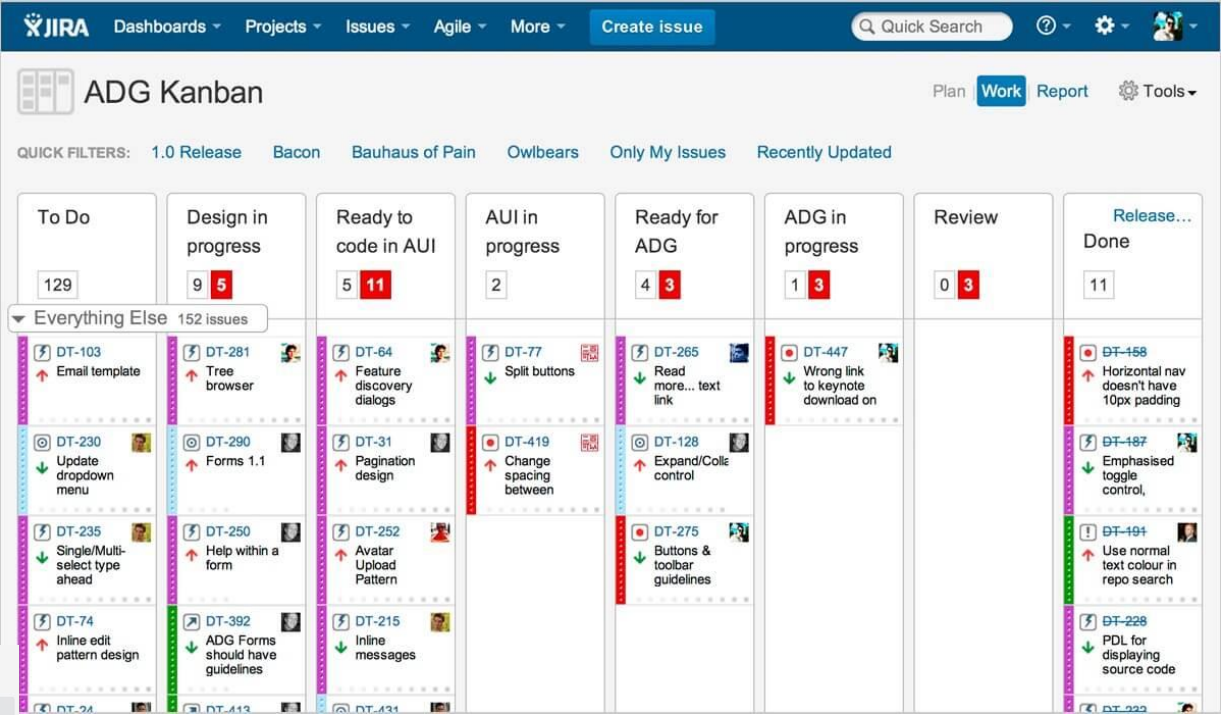
Tableros Kanban



Trello

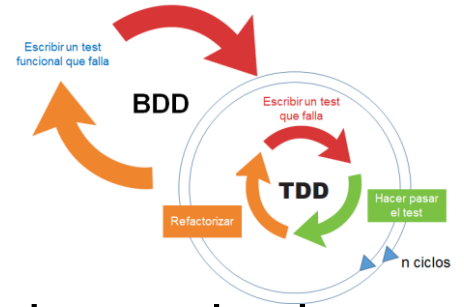


Jira



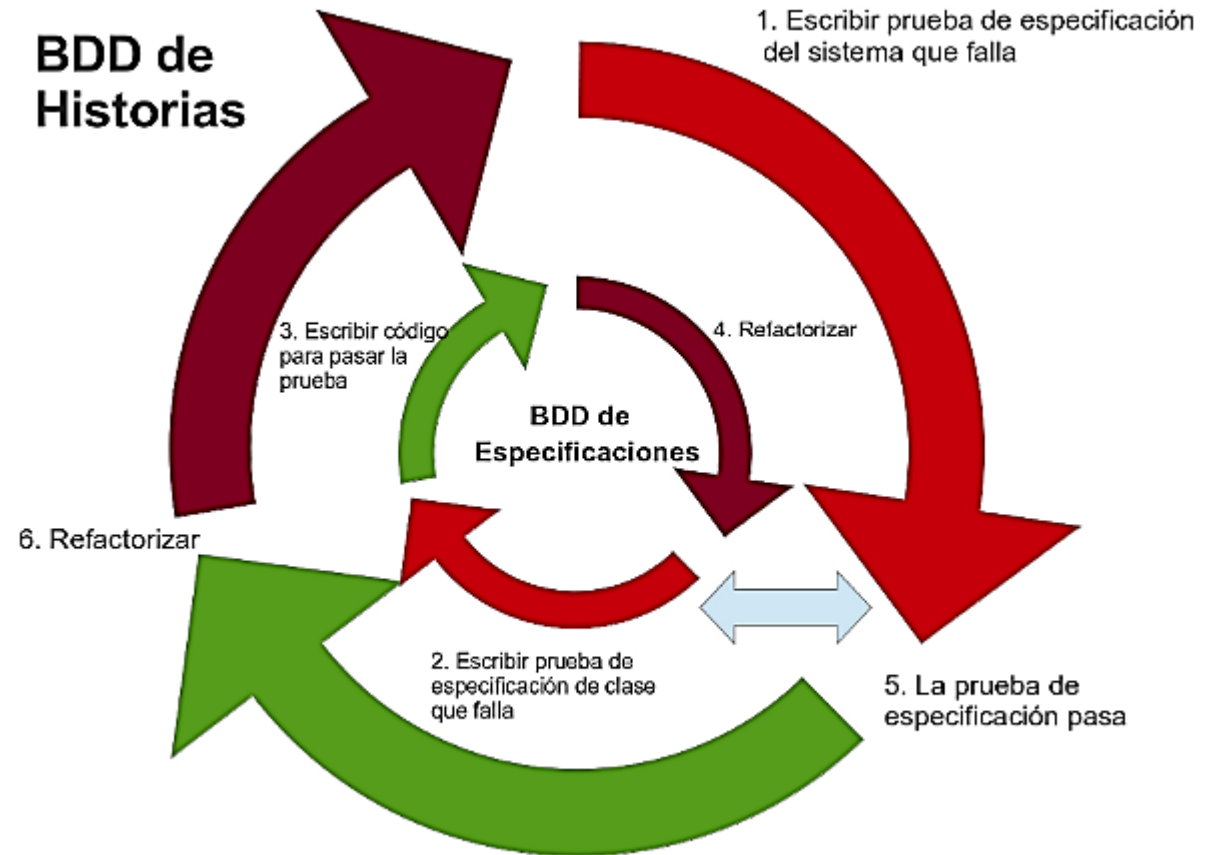
BDD-Behavior Driven Development

- BDD significa Behaviour Driven Development, o desarrollo orientado al comportamiento. Es una convención que nos permite escribir el comportamiento de un sistema en lenguaje natural para que pueda ser entendido por todos los actores
- No es una técnica de testing, sino que es una estrategia de desarrollo (así como TDD, que es *test driven development*). Lo que plantea es definir un lenguaje común para el negocio y para los técnicos, y utilizar eso como parte inicial del desarrollo y el testing.
- Esta estrategia encaja bien en las metodologías ágiles, ya que generalmente en ellas se especifican los requerimientos como historias de usuario. Estas historias de usuario deberán tener sus criterios de aceptación, y de ahí se desprenden pruebas de aceptación, las cuales pueden ser escritas directamente en lenguaje **Gherkin**.
- BDD es un proceso Agile de desarrollo de software que fomenta la colaboración entre diferentes actores en un equipo de desarrollo.



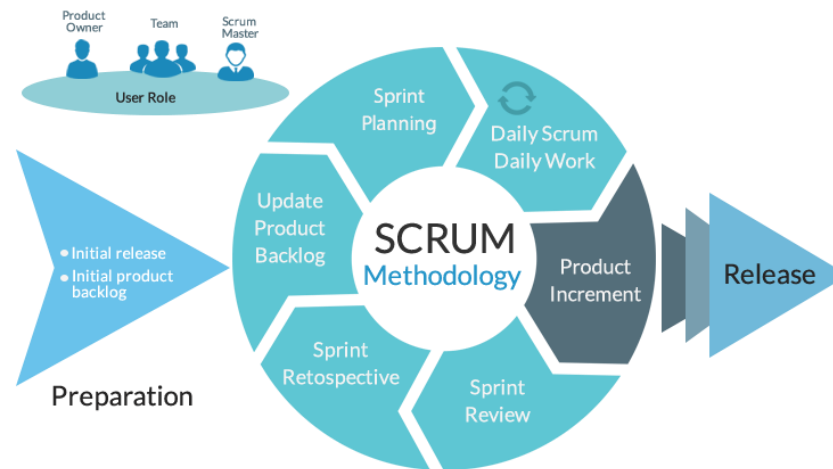
Ciclo BDD

- ✓ Iniciamos en un ciclo externo, donde escribimos la prueba de la especificación (paso 1), la misma falla.
- ✓ Luego entramos en un ciclo interno donde vamos escribiendo la especificación de nuestras clases (pasos 2, 3 y 4) y que se repite hasta que todo lo necesario para hacer pasar la prueba externa este listo (paso 5).
- ✓ finalmente refactorizamos a nivel del ciclo externo (paso 6). El ciclo externo lo repetimos hasta que todas las historias estén codificadas.



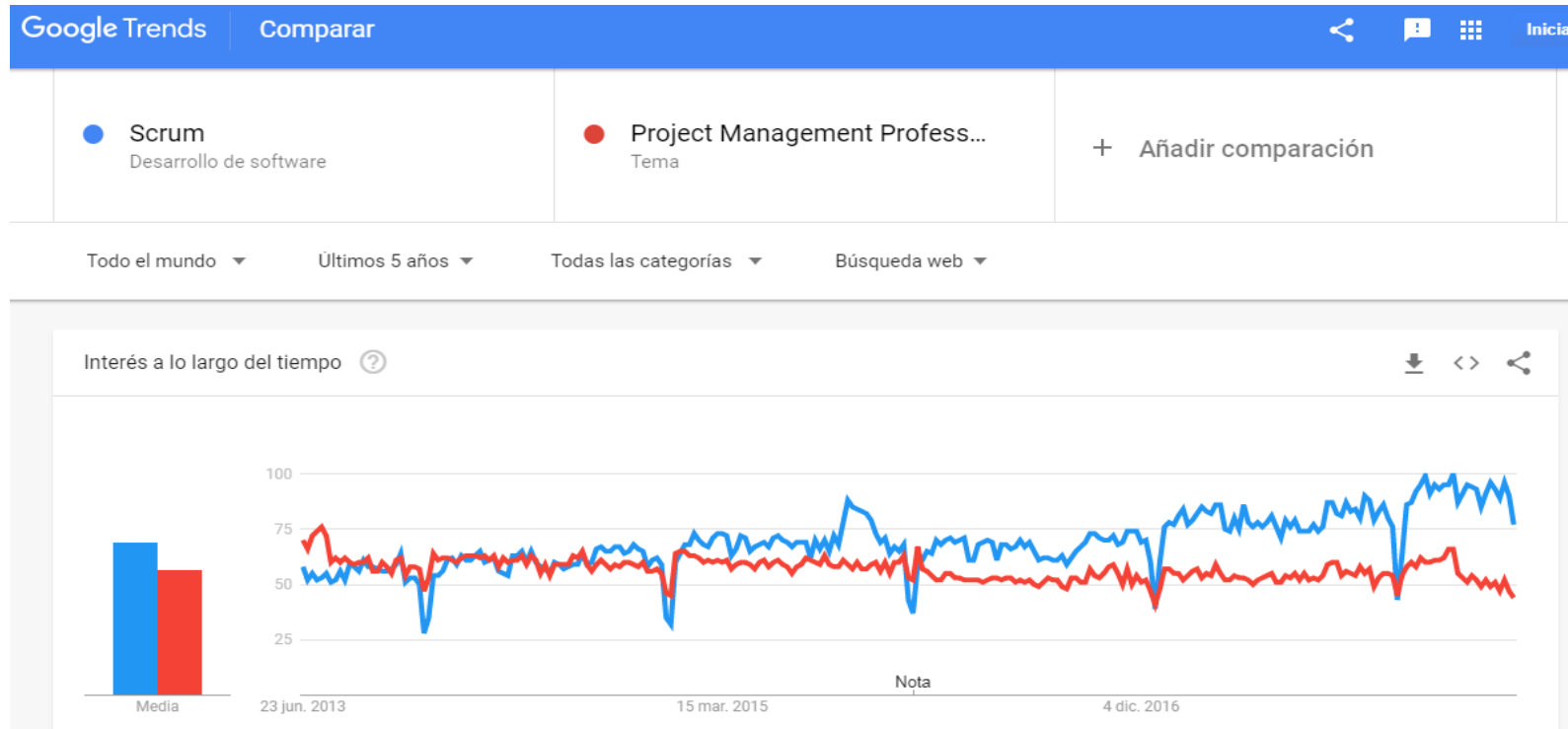
Scrum

- Su nombre corresponde a un concepto deportivo, propio del **rugby**, relacionado con la formación requerida para la recuperación rápida del juego ante una infracción menor.
- Su primera referencia en el contexto de desarrollo data de **1986**
- La metodología Scrum es un marco de trabajo diseñado para lograr la colaboración eficaz de equipos en proyectos, que emplea un conjunto de reglas y artefactos y define roles que generan la estructura necesaria para su correcto funcionamiento.
- Scrum utiliza un **enfoque incremental** que tiene como fundamento la teoría de control empírico de procesos. Esta teoría se fundamenta en transparencia, inspección y adaptación;



Resultados

- Hay pocos datos concretos del índice de éxito de proyectos
- Está teniendo un gran auge
- Aumento en el número de proyectos porque es un proceso para gente que odia los procesos



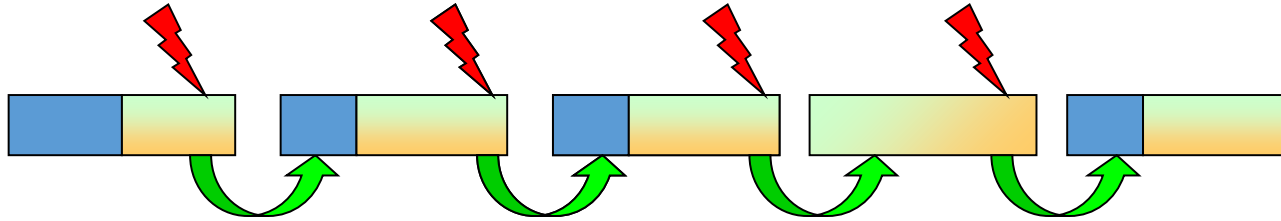
¿Cuando un método es ágil?

- El desarrollo de software es
 - **Incremental**
 - liberaciones pequeñas y ciclos rápidos.
 - **Cooperativo**
 - clientes y desarrolladores trabajando juntos.
 - **Simple y Directo**
 - el método es fácil de aprender y modificar.
 - **Adaptativo**
 - es posible realizar cambios de último momento.

Problemas del desarrollo iterativo

- El skill del grupo cambia a lo largo del tiempo
 - Más difícil adaptarse, hace más costoso los cambios.
- La prueba se vuelve costosa y repetitiva
 - Pérdida de motivación
 - Recorte de la prueba, pérdida de confianza

Desarrollo ágil



- Diseño en (casi) cada iteración
- La prueba con costo constante
- Siempre cerca de calidad de liberación

Consecuencias

- Grupo multidisciplinario y flexible
 - El grupo no puede cambiar continuamente, pero las necesidades cambian
 - La carga de trabajo por tipo de tarea son difíciles de predecir
- Los costos de los cambios deben mantenerse acotados
 - Se debe automatizar la prueba
 - Se debe refactorizar

Historias de Usuario

- Las historias de usuarios se originan con Extreme Programming
- La primera descripción escrita en 1998 solo afirma que los clientes definen el alcance del proyecto "con historias de usuarios, que son como casos de uso".
- En lugar de ofrecerse como una práctica distinta, se describen como una de las "piezas del juego" utilizadas en el "juego de planificación".
- Técnica para especificar los requisitos.
- Son tarjetas de papel
- Debe ser lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas



Historias de Usuario

- En el 2006 se genera la plantilla “**Connextra format**” basada en la propuesta del 2001
 - As a (**who** wants to accomplish something)
 - I want to (**what** they want to accomplish)
 - So that (**why** they want to accomplish that thing)
- Algunas herramientas como Jbehave, RSPEC o Cucumber emplean el formato **Given-When-then**
 - (Given) some context
 - (When) some action is carried out
 - (Then) a particular set of observable consequences should obtain

Historias de Usuario

- 2001: El formato “**role-feature-reason**” para expresar la historias de usuario fue inventado por **Connextra** en el Reino Unido
- 2001: El modelo “Card, Conversation, Confirmation” es propuesto por Ron Jeffries para distinguir las historias de usuario “sociales” de las prácticas de requisitos “documentales” como los casos de uso
- 2003 se define los criterios de verificación de calidad de una historia de Usuario basados es SMART (Specific, Measurable, Achievable, Relevant, Time-boxed), se plantea INVEST

