



# GESTIÓN DEL SOFTWARE Y DE LOS PROYECTOS DE DESARROLLO

© 2017, ACTIBYTI PROJECT SLU, Barcelona  
Autor: Ricardo Ahumada



MINISTERIO  
DE ENERGÍA, TURISMO  
Y AGENDA DIGITAL

red.es



ESTRATEGIA DE  
EMPRENDIMIENTO Y  
EMPLEO JUVENIL  
*garantía juvenil*



UNIÓN EUROPEA

Fondo Social Europeo  
“El FSE invierte en tu futuro”

# ÍNDICE DE CONTENIDOS

1. Caso práctico
2. Ciclo de vida de los proyectos de software
3. Scrum para desarrollo de software
4. Técnicas de desarrollo de software: XP, TDD

1

# CASO PRÁCTICO



# Caso Práctico: Banana Apps es Agile

Banana Apps diseña e implementa aplicaciones de gestión no triviales multicanal para empresas.

Para realizar el desarrollo de los productos, el equipo de desarrolladores de Banana, necesita un conjunto de herramientas y métodos que les permitan generar el software de la manera más eficiente posible.

Aspectos como la visión del proyecto, comunicación, cohesión de objetivos, reparto del trabajo, integración de los desarrollados, etc, de la manera más ágil posible, son esenciales para que el desarrollo de los productos de Banana tengan éxito; siempre teniendo en cuenta que se quieren generar productos de calidad.



# Discutamos

- Qué tipo de herramientas y métodos son necesarios?
- Cómo podemos cohesionar el equipo en torno a los objetivos de un proyecto?
- Qué ciclo de vida tiene el desarrollo de los proyectos de software?
- Qué tendencias existen hoy en día en la industria de software, respecto a las metodologías de gestión y desarrollo?
- Cómo podemos garantizar la calidad, incluso antes de comenzar a desarrollar?

2

## CICLO DE VIDA DE LOS PROYECTOS DE SOFTWARE

# El ciclo de vida del software

- El software **nace, crece y muere**, es su ciclo de vida
  - **Nace** con sus requerimientos y diseño
  - **Crece** con su desarrollo y mantenimiento
  - **Muere** cuando se reemplaza por otro: Software obsoleto

## ➤ El ciclo de vida de desarrollo:

"Es un proceso por el cual los analistas de sistemas, los ingenieros de software, los programadores y los usuarios finales elaboran sistemas de información y aplicaciones informáticas".

(Whitten J., Bentley L., Barlow V. 1996)

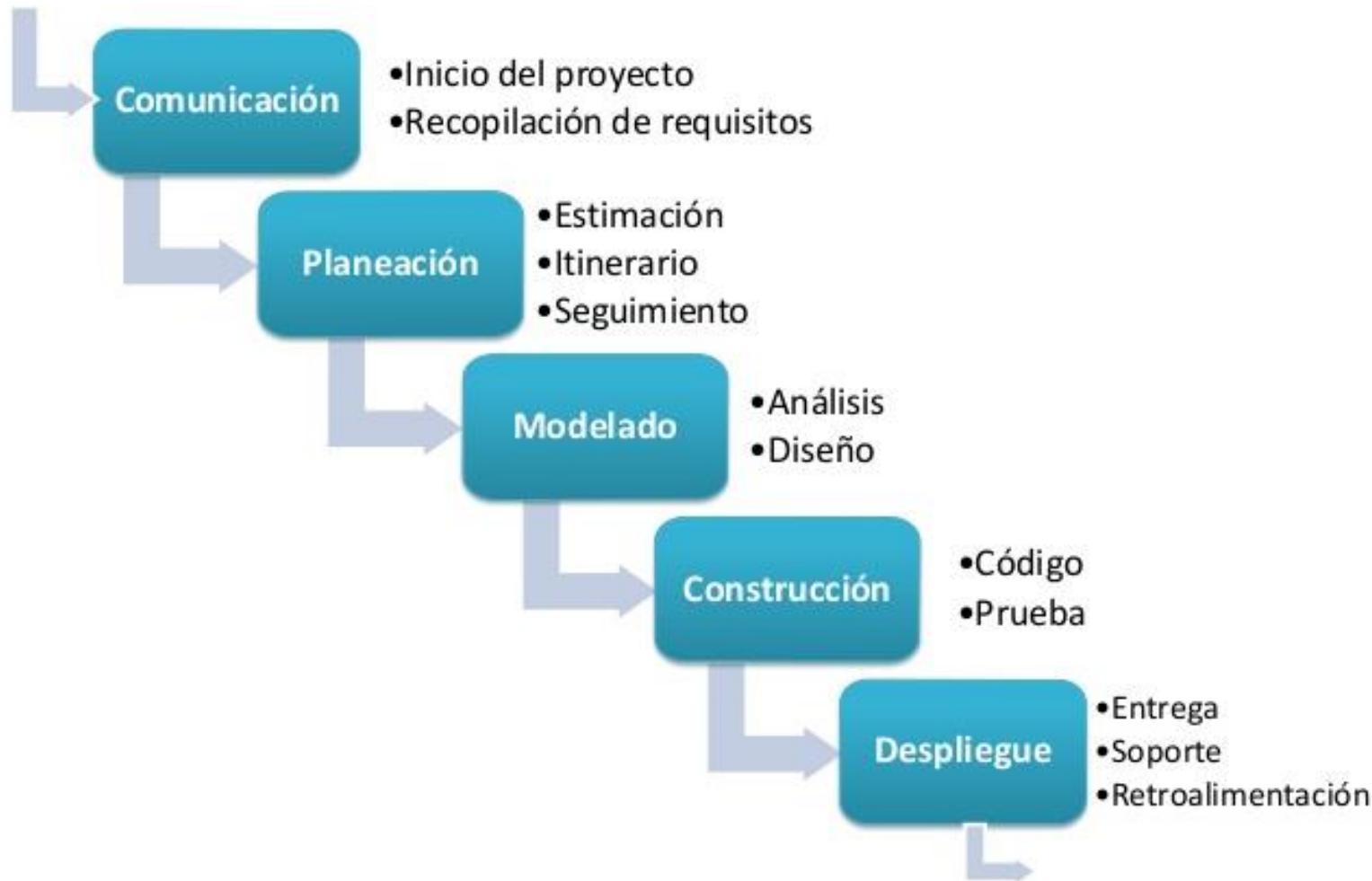


# Modelos para el ciclo de vida de desarrollo de software

- Existen varios modelos se han ido desarrollando a lo largo del tiempo con el objetivo de mejorar la producción del software y su calidad:

CASCADA	ESTRUCTURADO	ESPIRAL	PROPTOTIPO
<ul style="list-style-type: none"><li>• Análisis de requerimientos</li><li>• Especificaciones.</li><li>• Diseño.</li><li>• Implementación.</li><li>• Prueba</li><li>• Mantenimiento</li></ul>	<ul style="list-style-type: none"><li>• Encuesta</li><li>• Análisis.</li><li>• Diseño.</li><li>• Implantación..</li><li>• Pruebas</li><li>• Control de calidad.</li><li>• Procedimientos.</li><li>• Conversión B.D.</li><li>• Instalación.</li></ul>	<ul style="list-style-type: none"><li>• Requerimientos.</li><li>• Análisis de riesgo.</li><li>• Prototipo 1, 2.</li><li>• Req. software</li><li>• Validación de Req.</li><li>• Análisis de riesgo.</li><li>• Prototipo 3.</li><li>• Diseño software.</li><li>• Validación diseño.</li><li>• Integración y prueba.</li></ul>	<ul style="list-style-type: none"><li>• Requerim. Básicos</li><li>• Desarr. Prot. oper.</li><li>• Uso prot.</li><li>• Usuario satisfecho?.</li><li>• Si. Aceptar.</li><li>• No. Revisar y mejorar.</li></ul>

# Modelo en cascada: versión ideal



# Modelo en Cascada

## ➤ Inconveniencias

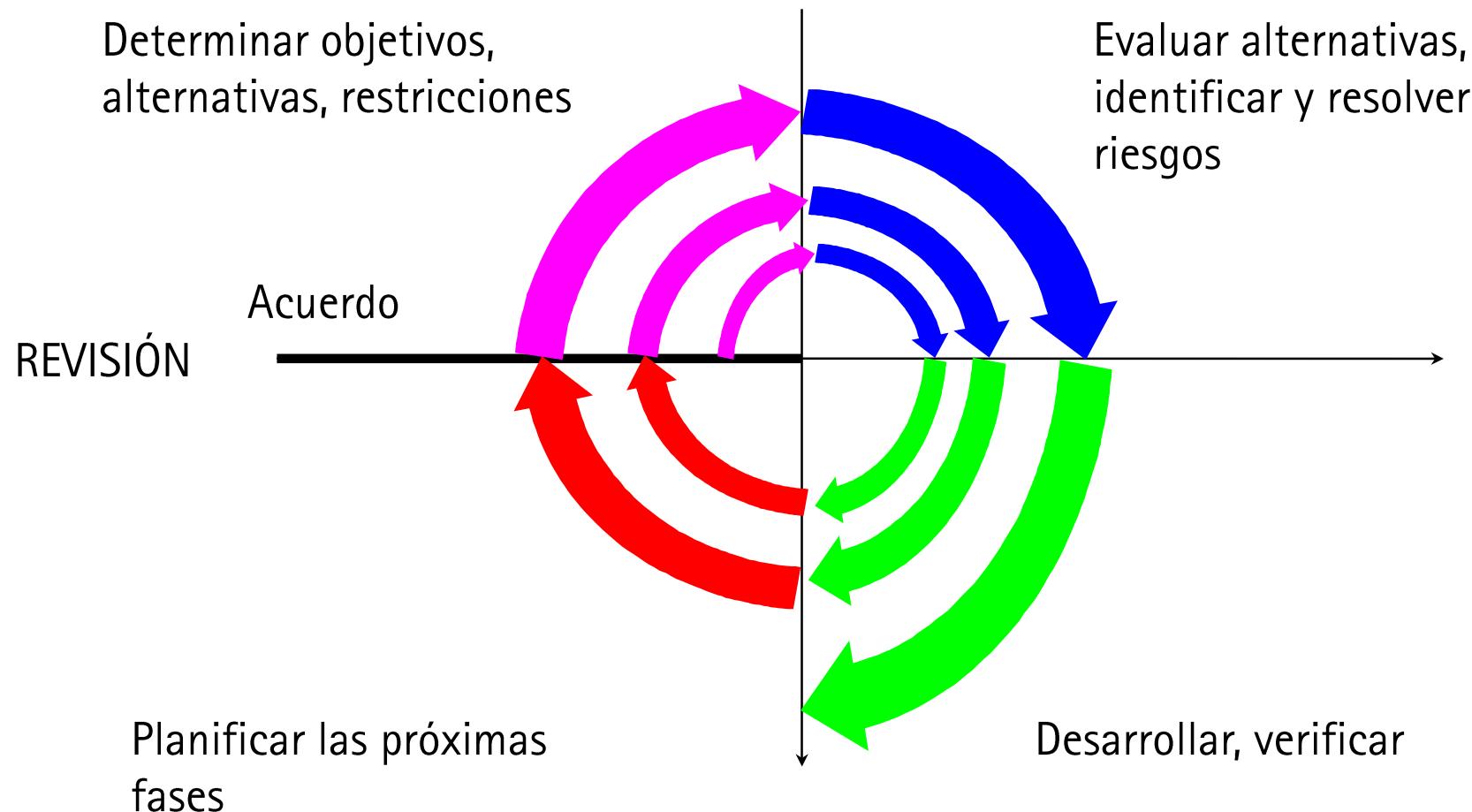
- › Rígido, difícil de rectificar
- › Documentación inicial se vuelve obsoleta

## ➤ Desarrollo evolutivo

- › Ciclo de vida en espiral
- › Uso de prototipos (de diversa fidelidad)
- › Extreme Programming
- › RAD (Rapid Application Development)

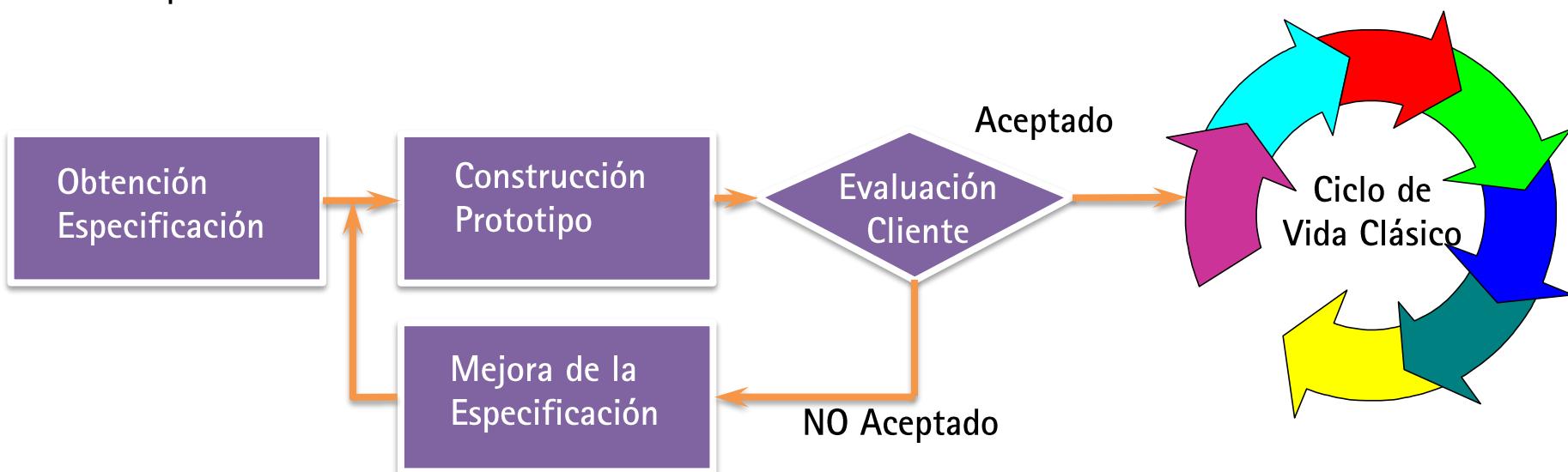
## ➤ Cambia el proceso pero no las actividades

# Modelo en Espiral

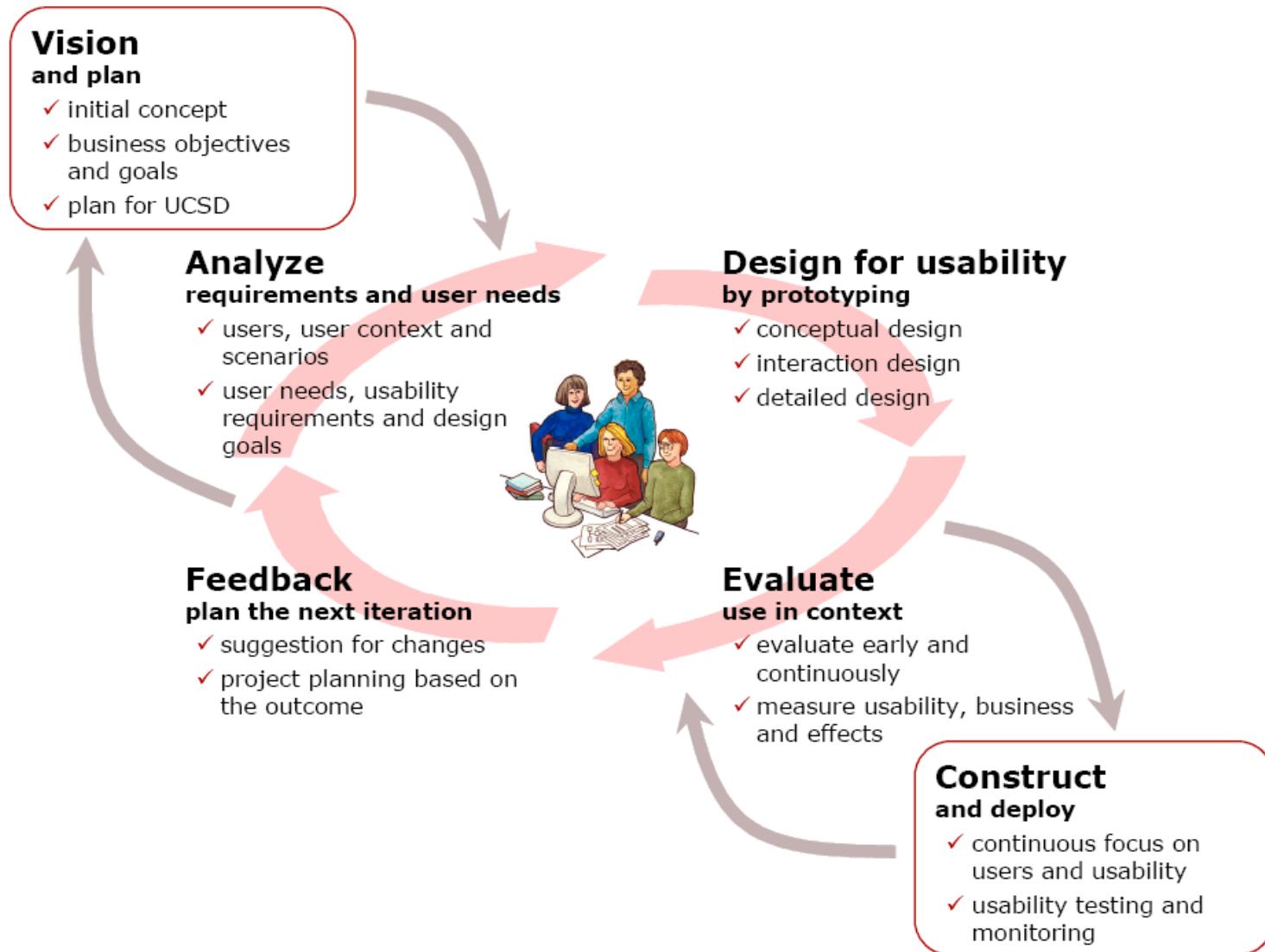


# Prototipos

- Se utilizan los prototipos para que el cliente observe, confirme y mejore el producto
- Este enfoque es apropiado cuando:
  - El cliente no tiene claro lo que quiere,
  - Al cliente le gustaría ver algo similar para poder hacerse una idea de lo que obtendrá



# Diseño Centrado en el Usuario



# Actividades en el proceso de desarrollo

Requerimiento

El sitio debe tener una tienda online.

Historias

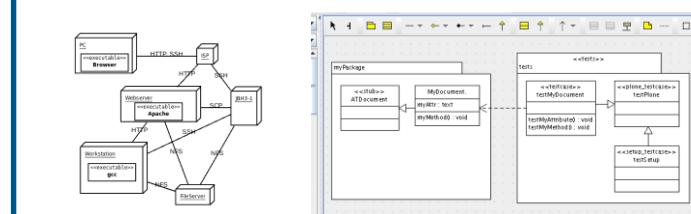
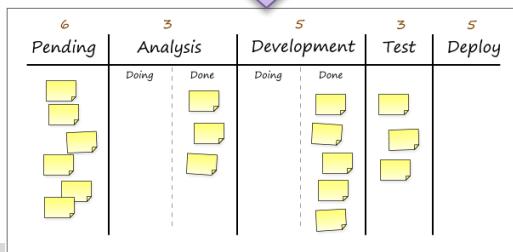
- Como consumidor, quiero una funcionalidad de carrito de la compra para comprar fácilmente los elementos en línea.
- + Criterios de aceptación

Diseño



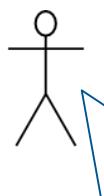
Desarrollo

HTML  
JS  
CSS  
JEE

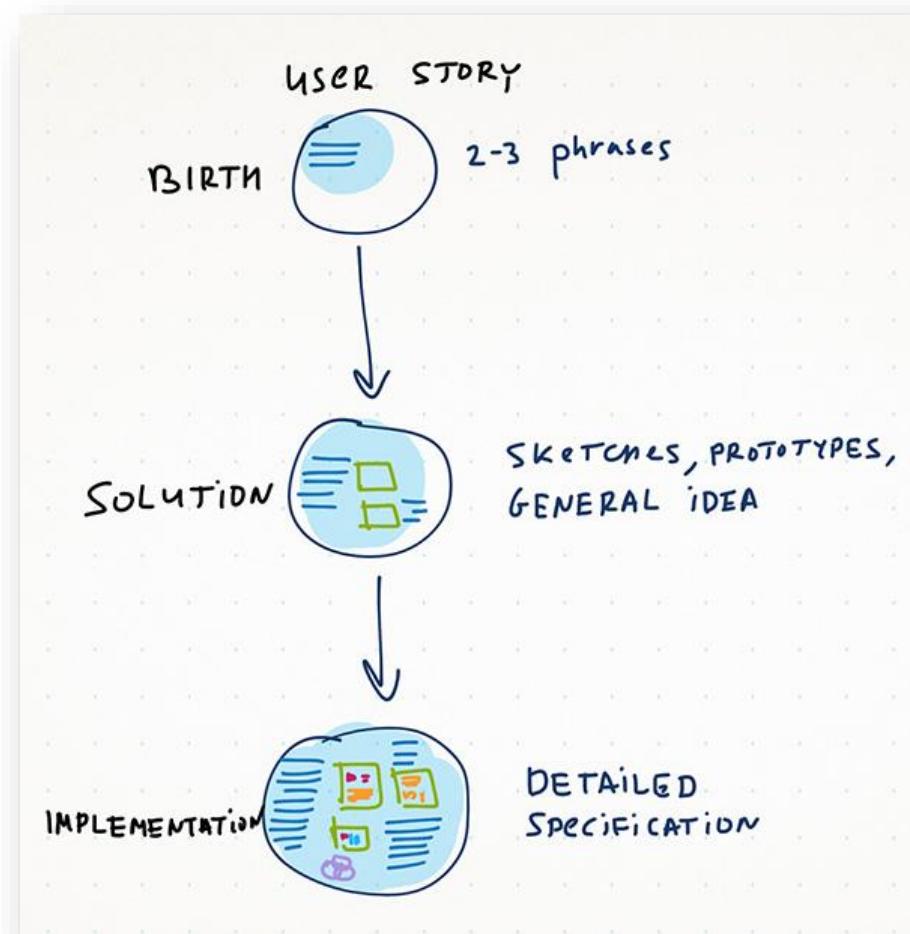


# Las historias de usuarios

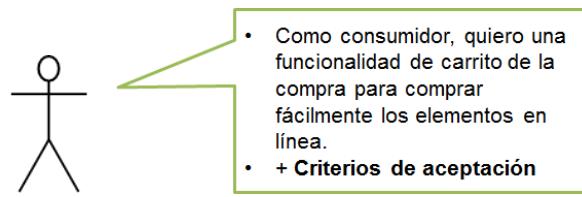
- Una historia de usuario representa una pequeña pieza de valor de negocio que un equipo puede entregar en una iteración



- ✓ Como usuario me gustaría enviar un mensaje corto a otro usuario para actualizarlos sobre los detalles del proyecto.



# Del diseño a las Tareas



TODO	DOING	TESTING	TOPRODUCTION	DONE
<div style="background-color: #ffffcc; border: 1px solid #ccc; padding: 5px;">Generar HTML</div> <div style="background-color: #ffffcc; border: 1px solid #ccc; padding: 5px;">Generar clases</div>	<div style="background-color: #e0f2e0; border: 1px solid #ccc; padding: 10px; text-align: center;">Diseñar interfaz</div> <div style="background-color: #e0f2e0; border: 1px solid #ccc; padding: 10px; text-align: center;">Diseñar orgánico + Tests</div>		<div style="border: 1px solid #ccc; padding: 10px; text-align: center;">Diseñar sistemas</div>	

# Análisis

## ➤ Entrada

- Conocimiento del dominio de la aplicación, actividades de los usuarios, mercado, etc.

## ➤ Actividades

- Identificar las necesidades del usuario
- Análisis de viabilidad
- Determinar los requerimientos de la aplicación

## ➤ Salida

- Documento de requerimientos del software

# Diseño

## ➤ Entrada

- Documento de requerimientos del software

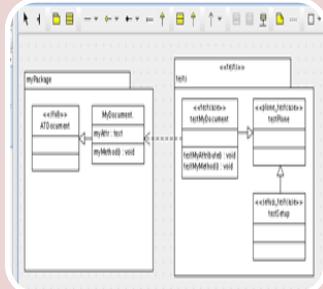
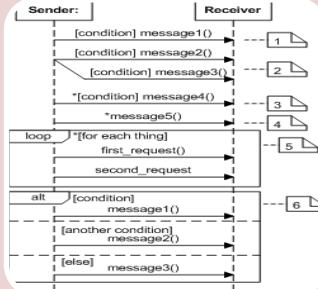
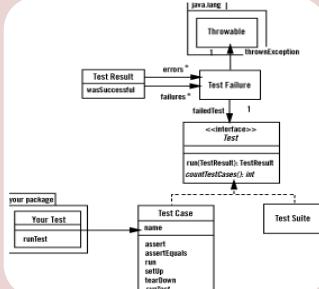
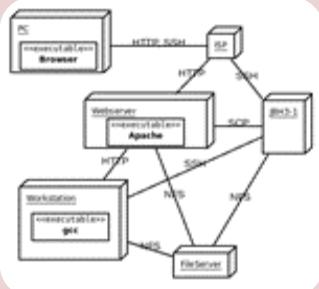
## ➤ Actividades

- Establecer estrategia de solución
- Análisis de alternativas. Formalizar la solución
- Descomponer y organizar la aplicación
- Fijar descripciones de cada módulo

## ➤ Salida

- Documento de diseño del software
- UML (Universal Modeling Language)

# Tipos de diseño



De  
interfaz

Sistemas

Orgánico

Funcional

Tests

# Codificación

- Entrada
  - Documento de diseño del software
- Actividades
  - Creación del código fuente
  - Pruebas de unidades
- Salida
  - Código de módulos, probado

# Validación

- Entrada
  - Código de módulos, probado
  - Documento de requerimientos del software (validación)
- Actividades
  - Pruebas de integración
  - Pruebas de validación
- Salida
  - Aplicación completa, lista para usar

# Etapa de Mantenimiento

- Entrada
  - Software listo para usar
- Actividades
  - Instalación
  - Uso en paralelo
  - Implementación
  - Nuevos requerimientos, correcciones y modificaciones
  - Soporte de usuarios
- Salida
  - Aplicación respondiendo a las necesidades actuales

3

# SCRUM PARA DESARROLLO DE SOFTWARE

# En un mundo IDEAL...

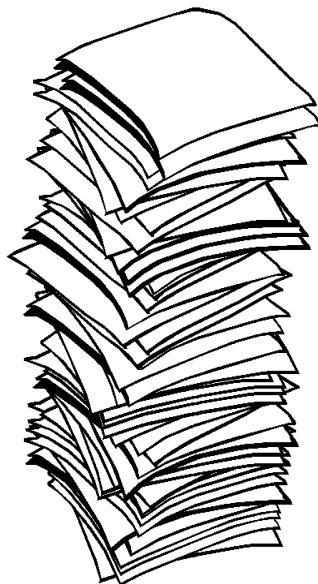
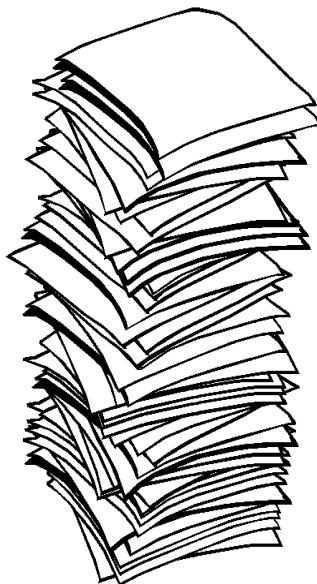
Análisis

Diseño

Construcción

Validación

Entrega



Requisitos &  
Plan de Proyecto

Diseño funcional

Diseño técnico

Resultados de las pruebas



El cliente obtiene lo que pidió.  
*(¡A tiempo y sin sobrecostes!)*

# Manifiesto por el Desarrollo Ágil de Software

Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros.

A través de este trabajo hemos aprendido a valorar:

**Individuos e interacciones** sobre procesos y herramientas

**Software funcionando** sobre documentación extensiva

**Colaboración con el cliente** sobre negociación contractual

**Respuesta ante el cambio** sobre seguir un plan

Esto es, aunque valoramos los elementos de la derecha,  
valoramos más los de la izquierda.

Kent Beck  
Mike Beedle  
Arie van Bennekum  
Alistair Cockburn  
Ward Cunningham  
Martin Fowler

James Grenning  
Jim Highsmith  
Andrew Hunt  
Ron Jeffries  
Jon Kern  
Brian Marick

Robert C. Martin  
Steve Mellor  
Ken Schwaber  
Jeff Sutherland  
Dave Thomas

# Principios del Manifiesto Ágil

*Seguimos estos principios (1 de 2):*

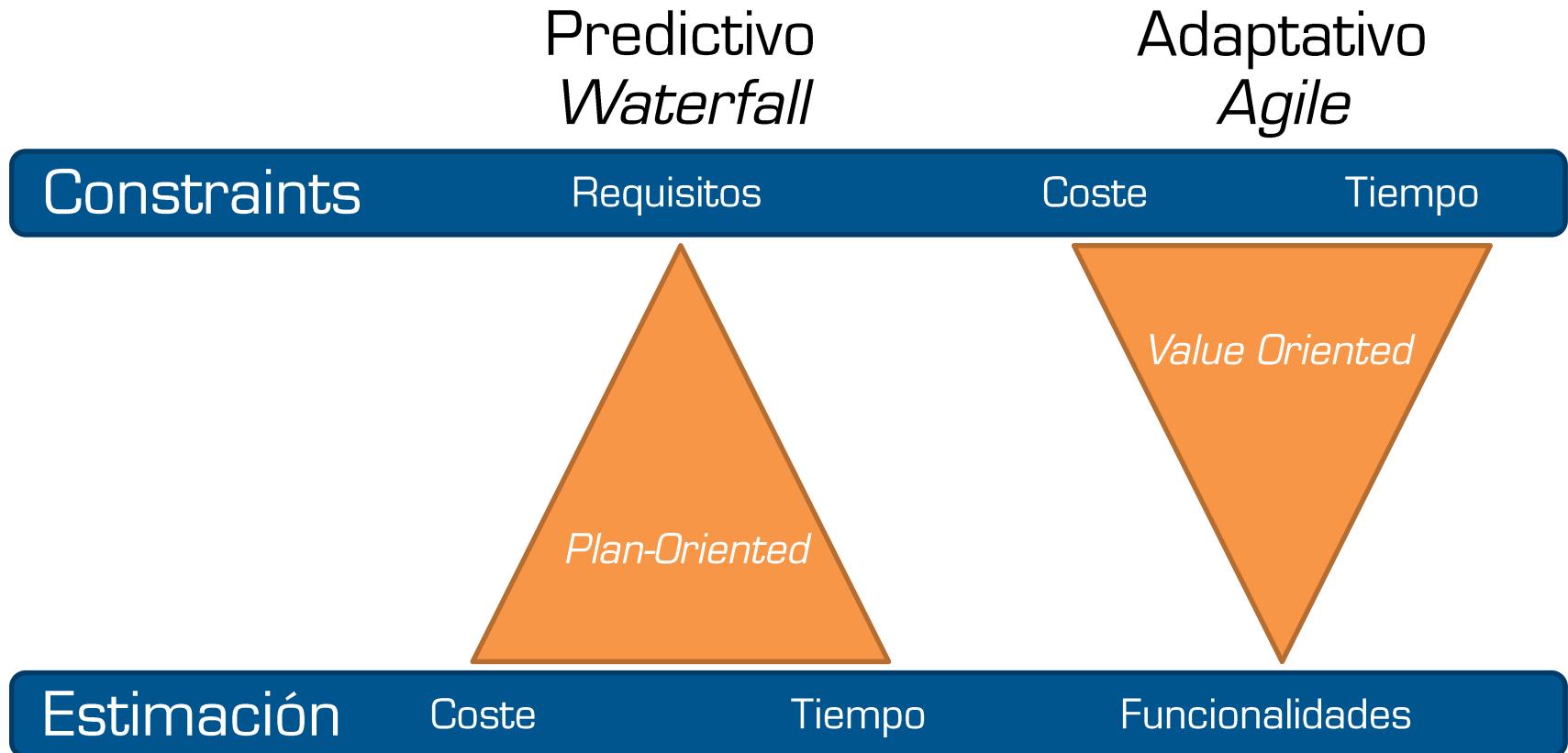
- 1.- Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
- 2.- Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- 3.- Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
- 4.- Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
- 5.- 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.- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.

# Principios del Manifiesto Ágil

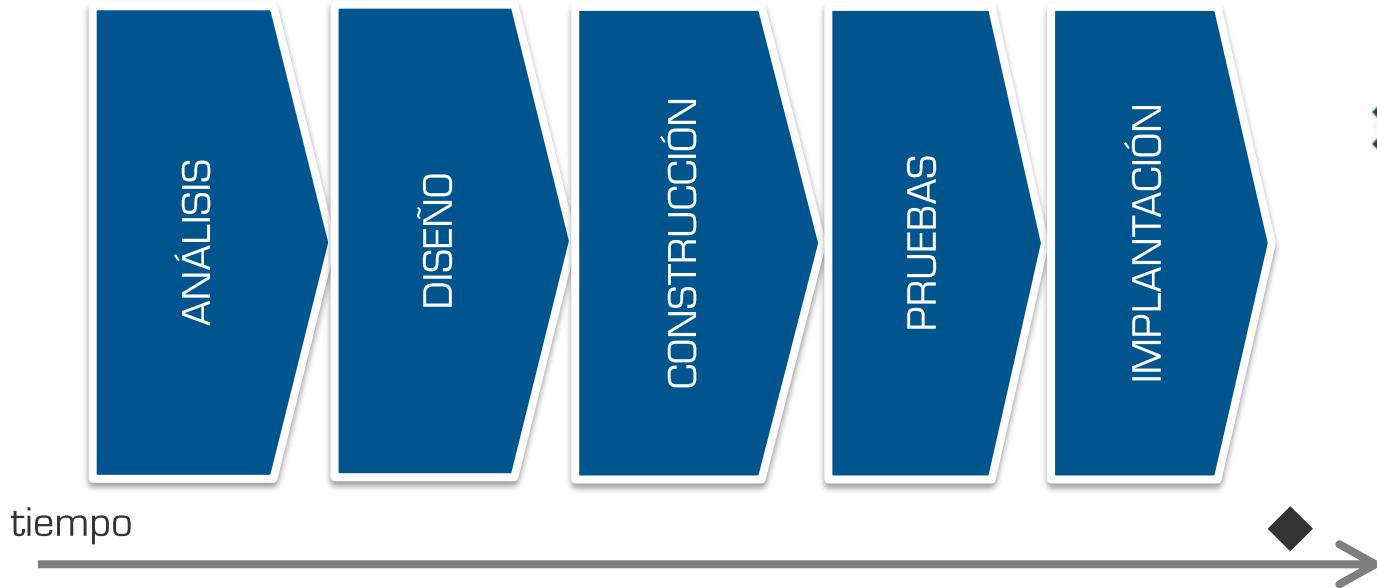
*Seguimos estos principios (2 de 2):*

- 7.- El software funcionando es la medida principal de progreso.
- 8.- Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
- 9.- La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
- 10.- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- 11.- Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
- 12.- A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

# Cambiando la orientación del Triangulo de Hierro

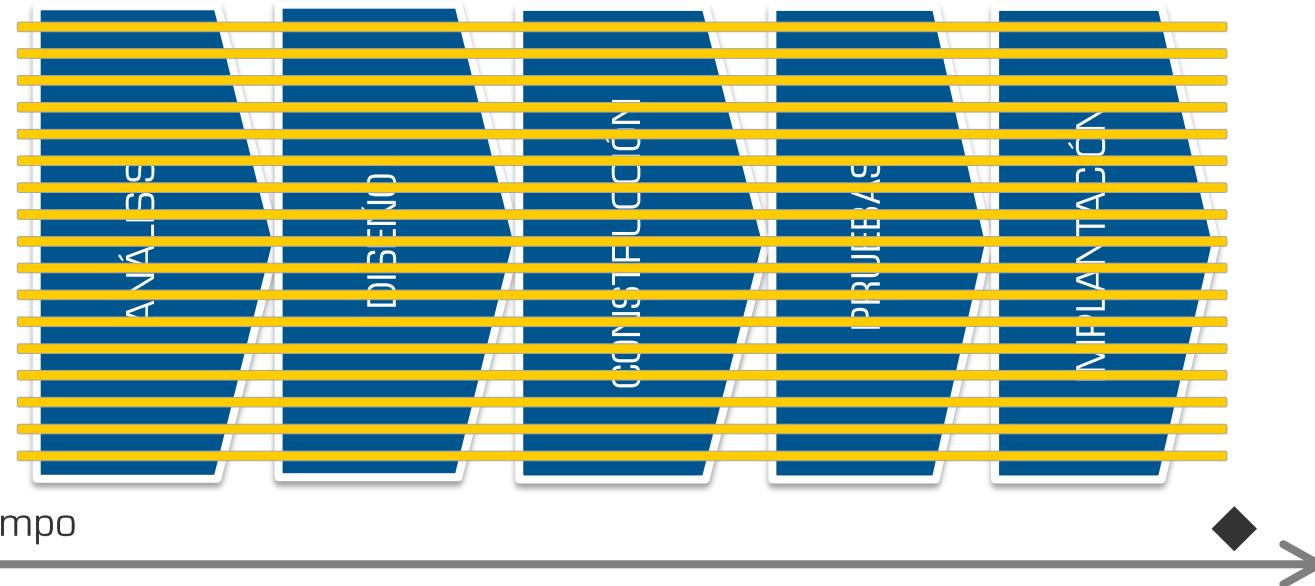


# Ciclo de vida tradicional vs ágil



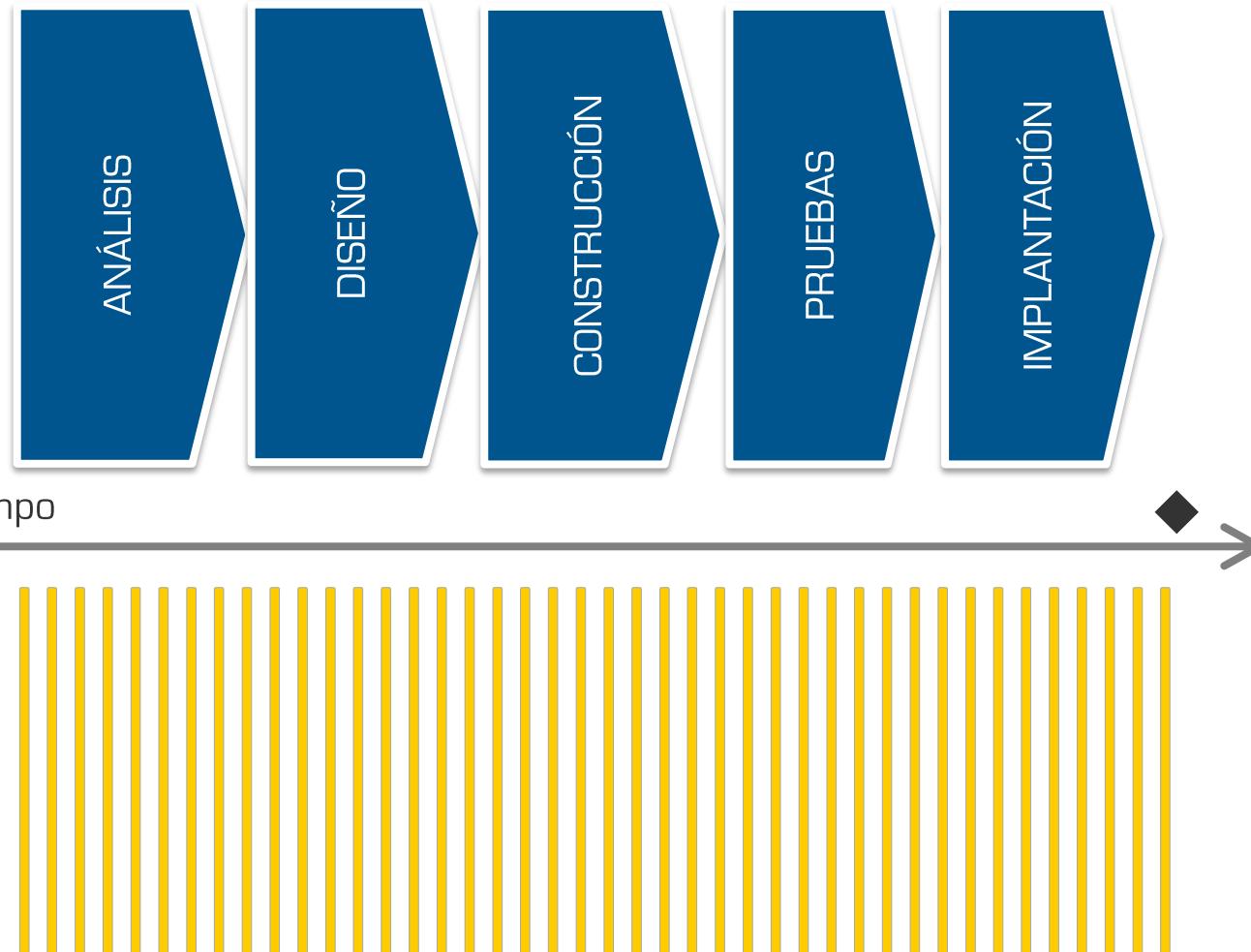
- Supongamos un proyecto con las clásicas fases en cascada

# Ciclo de vida tradicional vs ágil



- Rompemos el proyecto en pequeñas piezas que van de inicio a fin de todo el proceso....

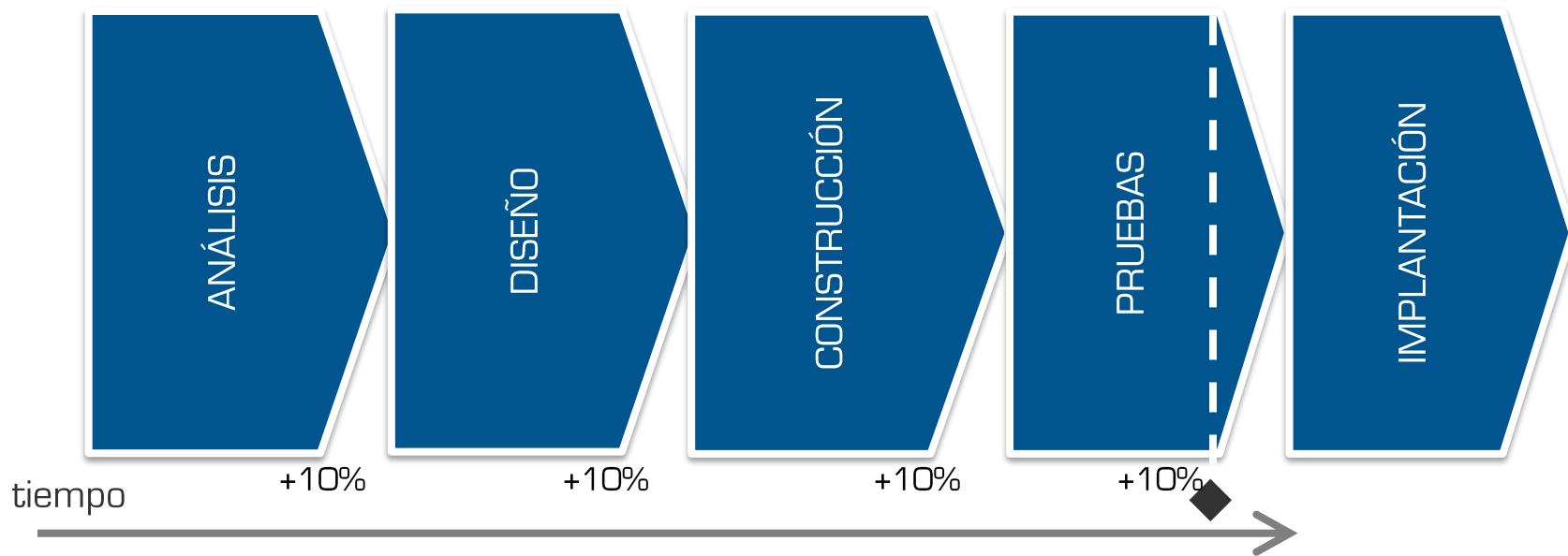
# Ciclo de vida tradicional vs ágil



- Rompemos el proyecto en pequeñas piezas que van de inicio a fin de todo el proceso....
- ... y las vamos ejecutando secuencialmente , por iteraciones.

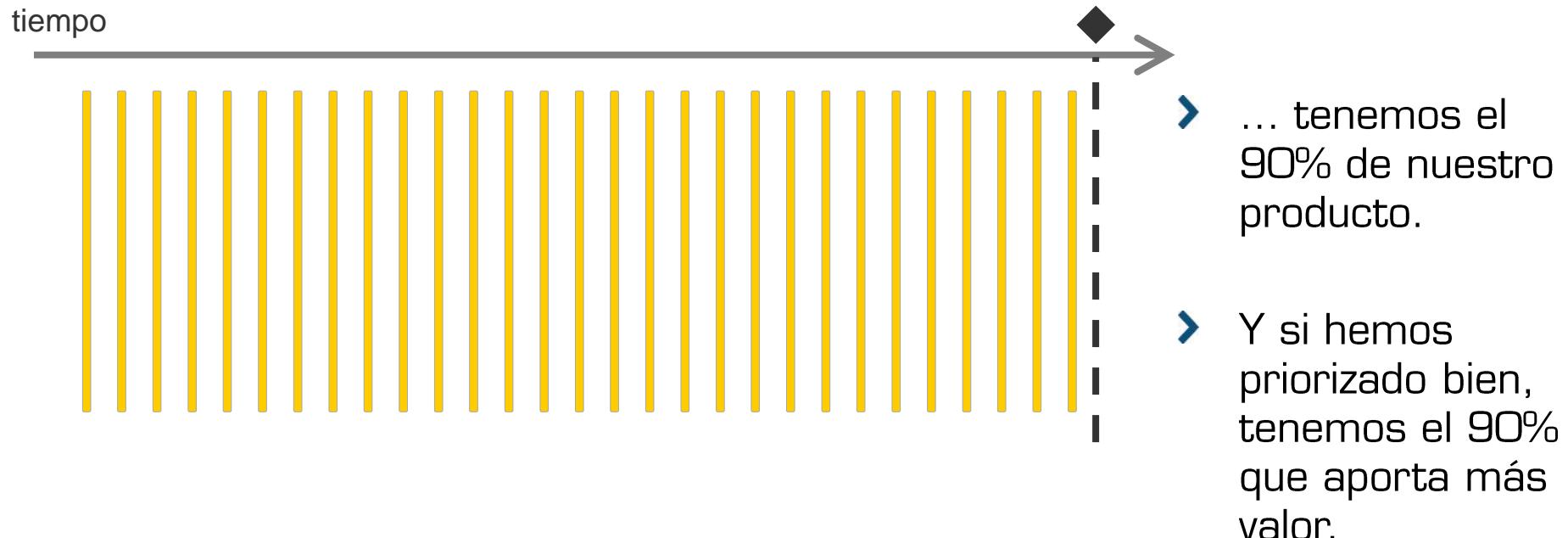
# Ciclo de vida tradicional vs ágil

- › Si por cualquier motivo nos desviamos un 10% en cada fase y tenemos comprometida la fecha de entrega, normalmente intentamos recuperar el tiempo perdido corriendo más al final, a costa de las pruebas.
- › Como consecuencia, entregamos un producto incompleto, con errores y tarde.



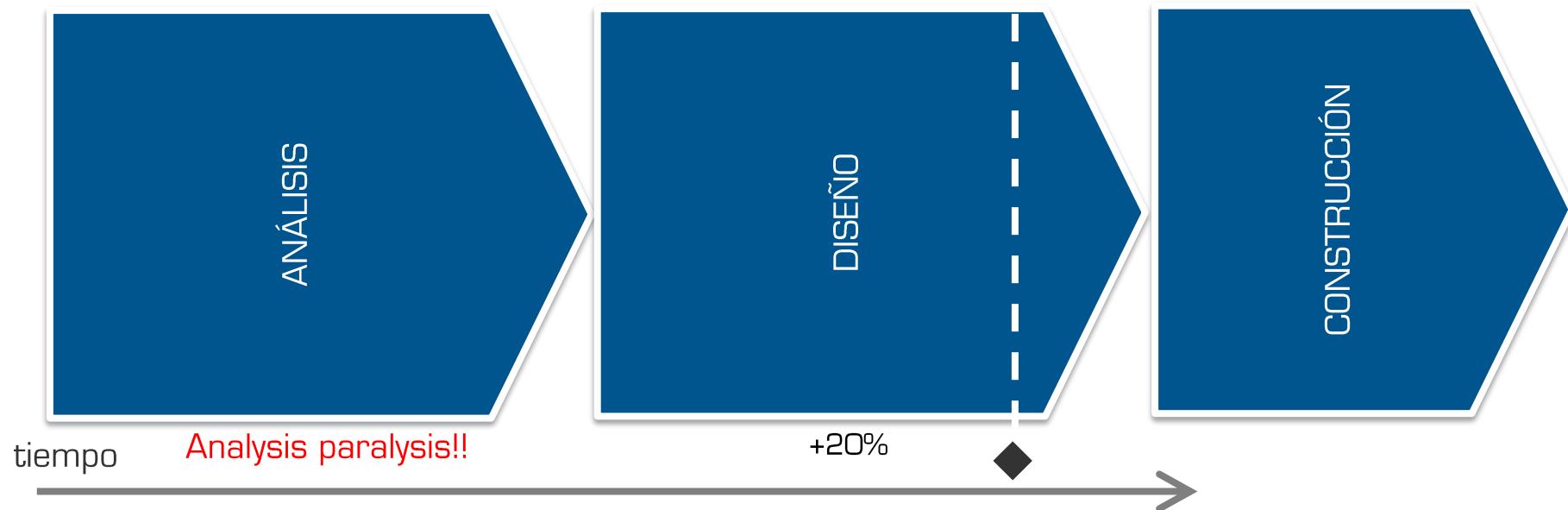
# Ciclo de vida tradicional vs ágil

- › Si nos retrasamos un 10% en un enfoque incremental...



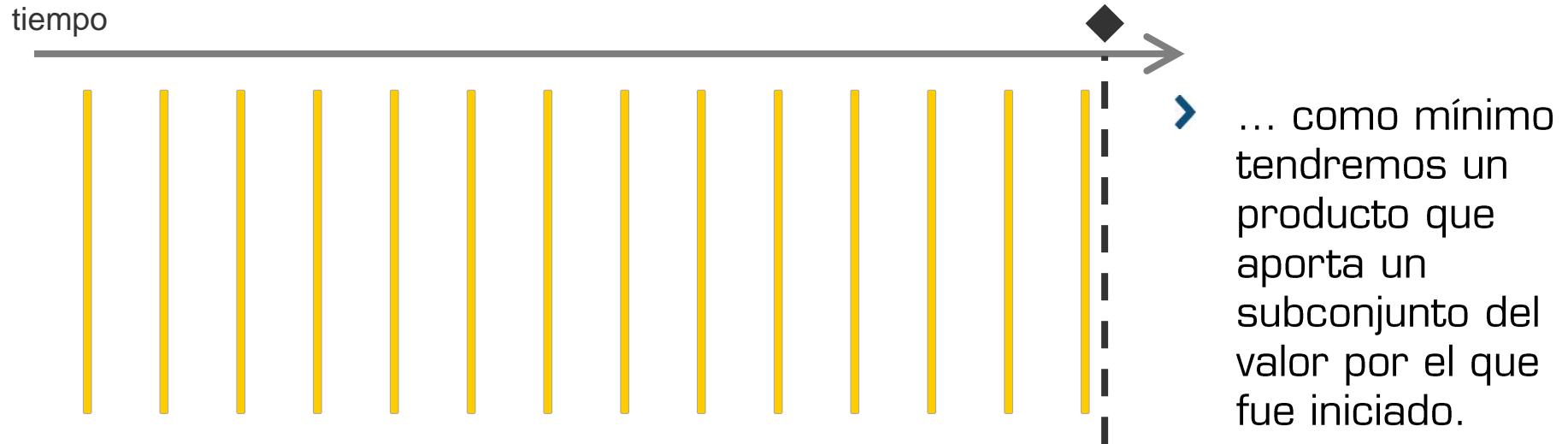
# Ciclo de vida tradicional vs ágil

- Y si, además, nos desviamos o nos encallamos en las fases iniciales, al llegar la fecha comprometida no tenemos más que documentos funcionales que no aportan ningún valor.

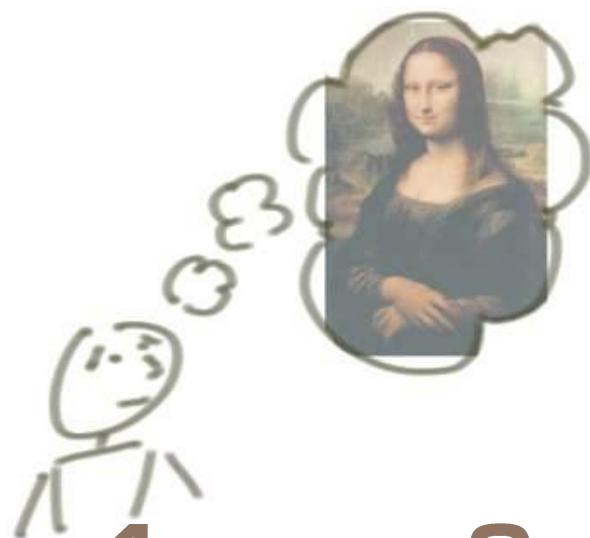


# Ciclo de vida tradicional vs ágil

- Y si somos realmente lentos y poco efectivos....



# “incrementing” builds a bit at a time



- Incrementing calls for a fully formed idea.
- And, doing it on time requires dead accurate estimation.

1

2

3

4

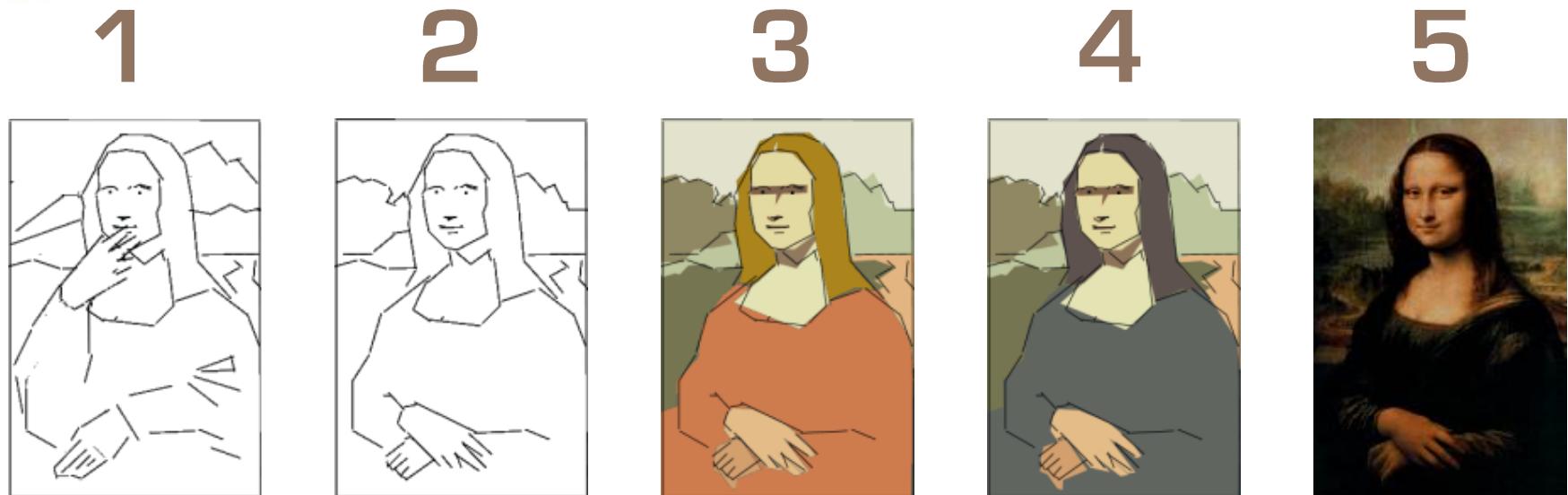
5



“iterating” builds a rough version, validates it, then slowly builds up quality



- A more iterative allows you to move from vague idea to realization making course corrections as you go.



# Scrum



Scrum is an iterative and incremental framework for building a product

# Scrum

## The Agile: Scrum Framework at a glance

Inputs from Executives,  
Team, Stakeholders,  
Customers, Users



**Product Owner**



**The Team**



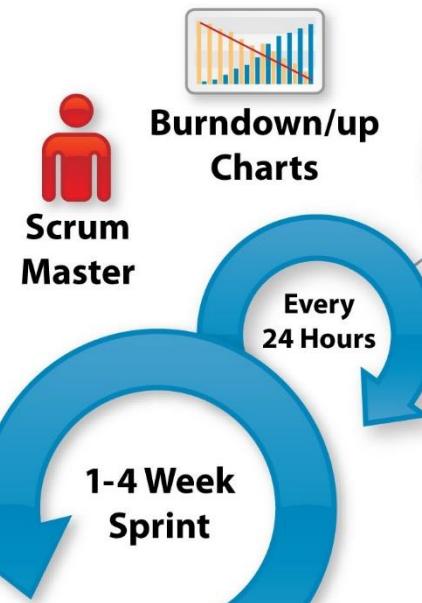
**Product Backlog**



**Sprint Planning Meeting**



Sprint end date and team deliverable do not change



# Scrum

## Roles

- Product Owner
- Development Team Member
- Scrum Master

## Artefactos

- Product Backlog
- Sprint Backlog
- Product Increment

## Actividades / Reuniones

- Product Backlog Refinement
- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective

# The Scrum Guide™

The Scrum Guide™

---

The Definitive Guide to Scrum:  
The Rules of the Game



July 2016

Developed and sustained by Ken Schwaber and Jeff Sutherland

employs an iterative, incremental approach to optimize predictability and control risk.

Three pillars uphold every implementation of empirical process control: transparency, inspection, and adaptation.

the Development Team isn't allowed to act on what anyone else says.

ide contains the  
nd the rules that  
scrum Guide is

Master.  
how best to  
-functional  
others not  
vity, and

es for  
on of

development  
ts; rather, it is a  
um makes clear  
o that you can

ts, artifacts, and  
essential to

relationships and  
ady of this

ows what

the level

However, the

ent the

ect Backlog

decisions. The

cklog. No

ments, and



# Crea tu propio Scrum

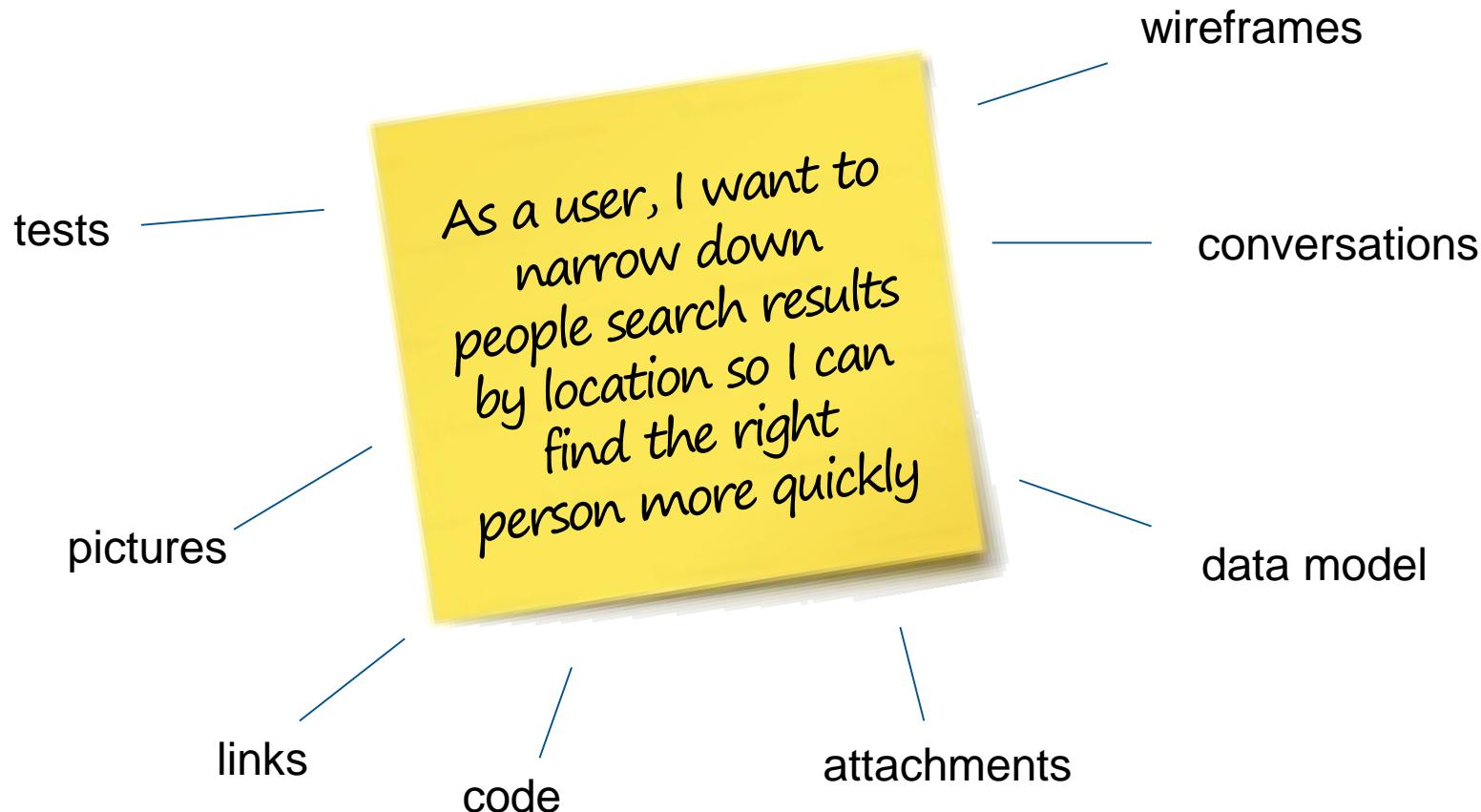


# Historias de usuario – User Stories

- Descripción breve de la funcionalidad que aporta valor al cliente
- Es un recordatorio. Lo verdaderamente importante es la conversación cara a cara
- Idealmente son independientes, aunque no siempre es posible
- Las historias las escribe el cliente
- El cliente prioriza las historias en función del valor aportado a la organización

# Verbalización de historias

- Como <usuario tipo> quiero <una función> y que me aportará <valor>
- Se colocan en un postit y se le puede complementar con más datos “ala vista”





## Capturando las historias de usuario

- Cada grupo debe escoger un compañero que haga el rol de cliente y conjuntamente generar 5 o más historias para BananaTube (Usa postits)
  
- Guarda las historias, las necesitaremos más tarde.

# Características de las historias - INVEST

- I Independent (of all others, as much as possible)
- N Negotiable (not a specific contract for features)
- V Valuable (to the customer)
- E Estimable (to a good approximation)
- S Small (so it can be done by a few person-days work)
- T Testable (in principle, even if there isn't a test for it yet)

# Product Backlog

- El backlog está compuesto por todas las necesidades identificadas por el equipo de proyecto
- Necesario para planificar las entregas e iteraciones
- Las US generan una lista de propiedades: el Product Backlog
  - Requisitos funcionales y no funcionales
  - Priorizados
  - Esfuerzo valorado a alto nivel
  - Al nivel de detalle suficiente para continuar
- El Product Backlog es una extensión de la Visión del Producto



## Generando el backlog

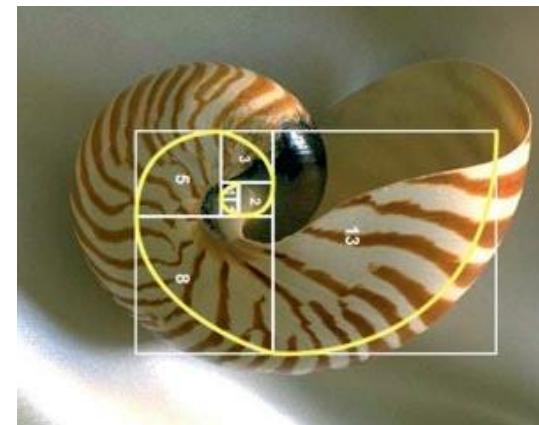
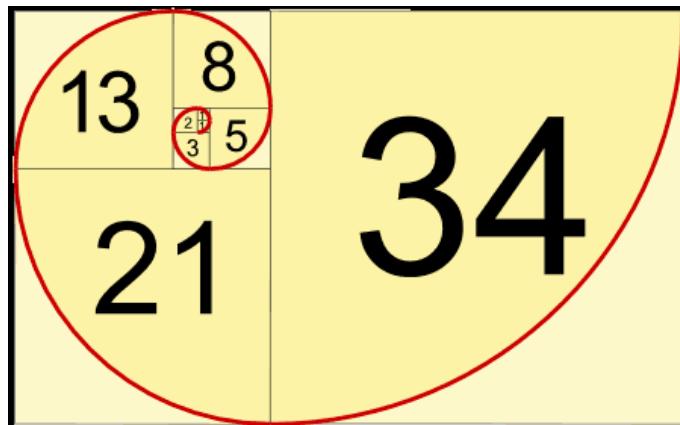
- Ordena y prioriza las historias de BananaTube en el product backlog

# Estimando con Story Points

- Un Story Point es una unidad para dimensionar historias
- Es una **unidad arbitraria del ESFUERZO necesario para realizarla**
- Depende del esfuerzo, de la complejidad y de la incertidumbre asociada a la Historia de Usuario
- Cada equipo tiene su propia “traducción” del esfuerzo necesario para realizar un Story Point
  - Para algunos equipos significa “1 dia”
  - Para algunos equipos significa “1 semana”
  - Para algunos equipos significa “1 día ideal”
  - Para algunos equipos significa “4 horas”

# Secuencia de Fibonacci y otros métodos

- Nos interesa un “orden de magnitud” del esfuerzo, así que podemos utilizar diferentes enfoques:
  - Fibonacci: 1, 2, 3, 5, 8, 13, 21, 34,...
  - Pseudo Fibonacci (most common): 0,  $\frac{1}{2}$ , 1, 2, 3, 5, 8, 13, 20, 40, 100
  - T-Shirts: XL, L, M, S, XS



<http://www.mathsisfun.com/numbers/images/fibonacci-spiral.gif>

# Planning Poker



# Relative sizing



# Affinity Estimating

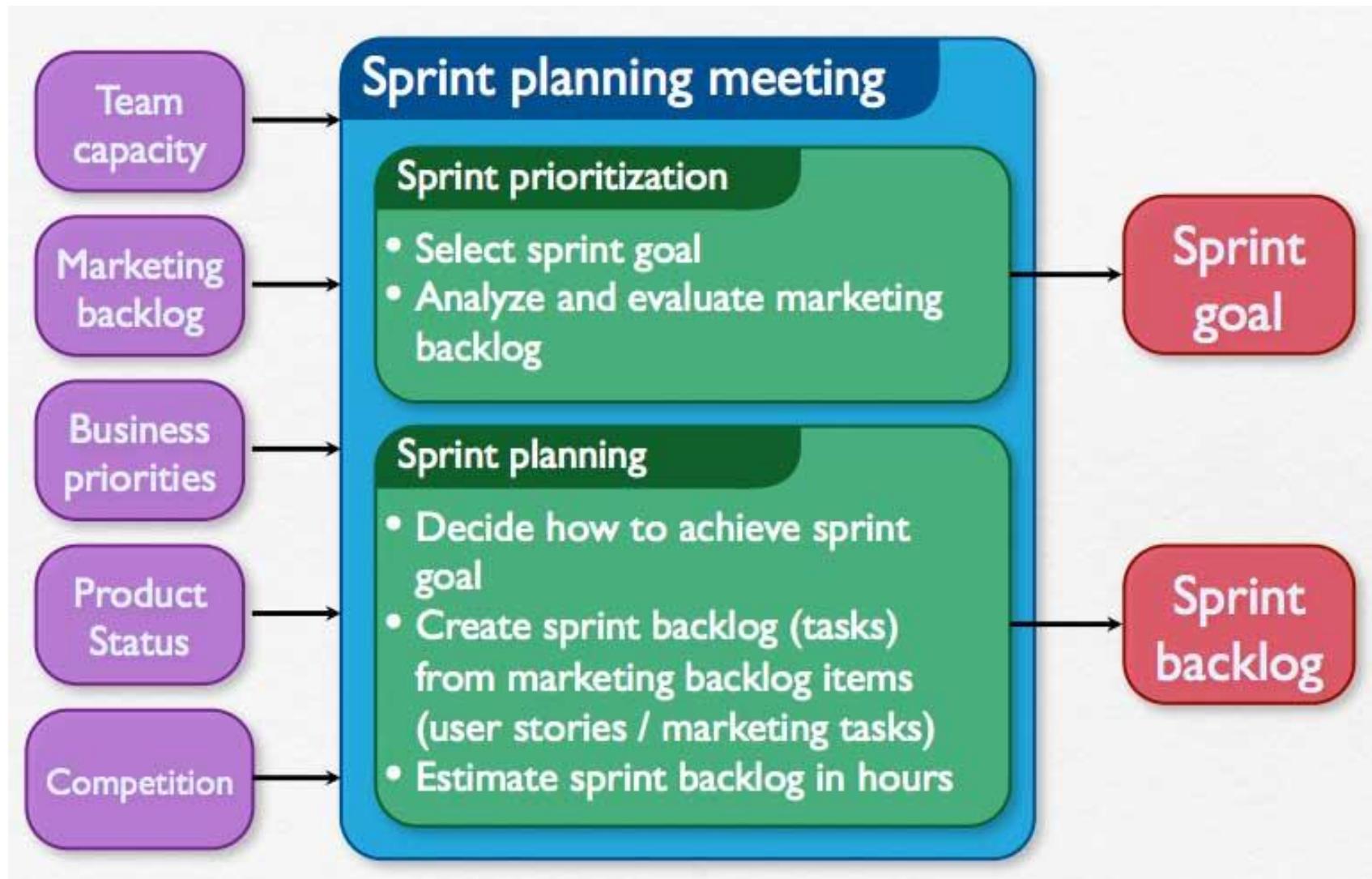




## Estimado las historias

- Usando el método del planning poker
  - Escoged en grupo una historia base y dadles una puntuación
  - Estimad las siguientes historias en función de esta puntuación
- Decidid cuanto durará un sprint
- Estimad cuantos puntos podéis asumir en un sprint

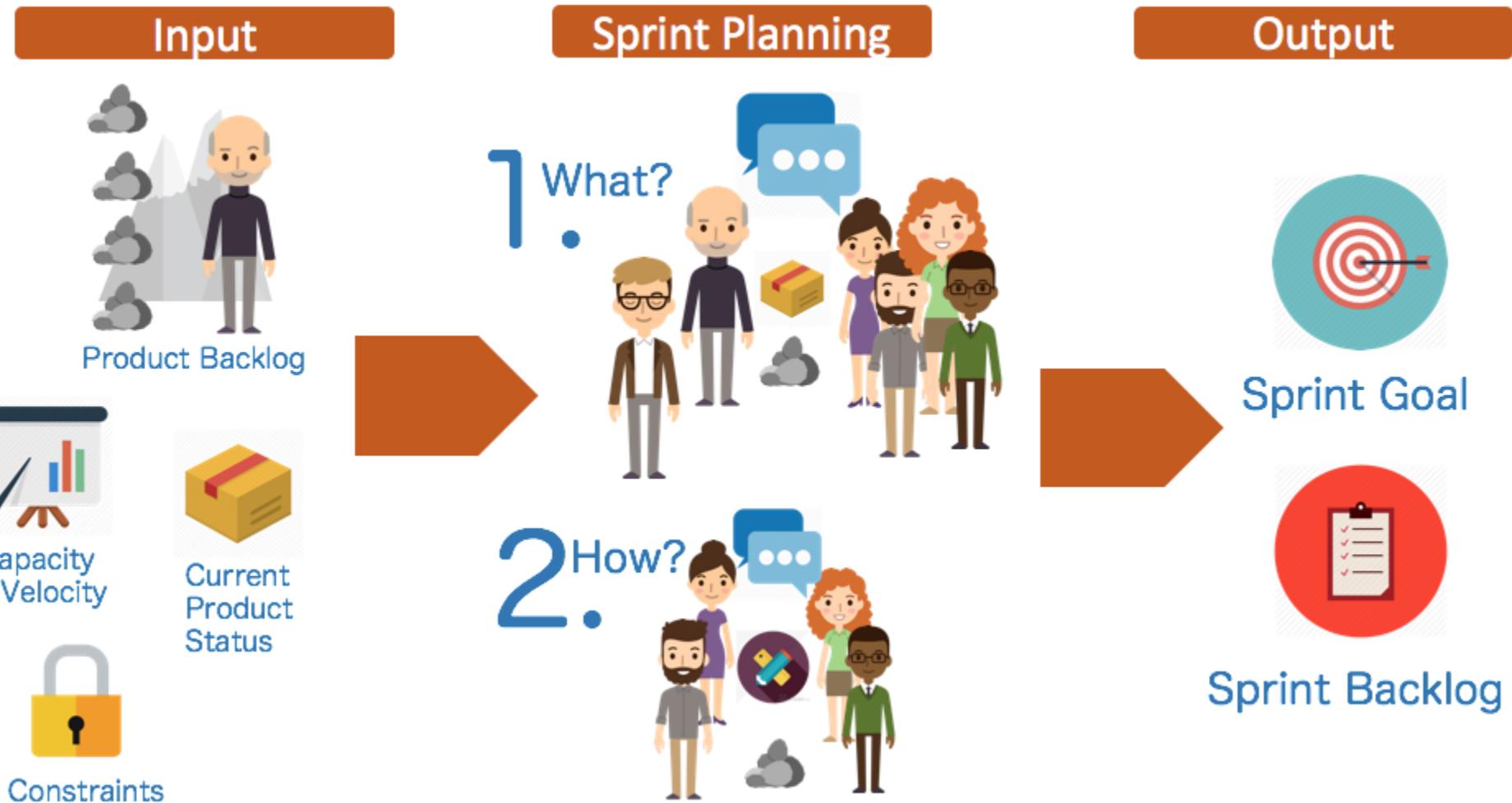
# Sprint Planning



# Sprint Planning

Sprint Backlog	Tasks				
Send invitations (1)	Get map hypertext link	Create email invite	Get email addresses	Send emails	Test emails were sent
Prepare Drinks (3)	Make list soft drinks	Make liquor list	Go to liquor store	Go to grocery store	Put drinks in fridge
Prepare Food (8)	Make menu/ list of food	Go to store	Cook foods that need it	Put food in fridge	
Clean LR/DR (5)	Clean living room	Clean dining room			

# Sprint Planning





## Planificando un primer sprint

- Selecciona las historias con las que comenzaréis a trabajar y generad las tareas
  - Intentad indentificar qué sería necesario hacer
  - Asimismo, cómo se haría → tareas
- **Nota:** ten en cuenta las actividades del ciclo de desarrollo del software
- Prioriza las tareas y asignar responsables
- Cuál es el goal del sprint?

# No es lo mismo

Task	Status
Acceptance Criteria Samples	Ongoing
Gantt Plan Samples	Ongoing
Status Report A6 Template	Pending
Role Information Updated	Pending
Risk A6 Template	Ongoing
Risk Samples	Done
Initiative Form Fulfilled	Pending
Core Exercise Decided	Ongoing
Review Team Deliverables	Done
Requisite Sample Template	Ongoing

# No es lo mismo

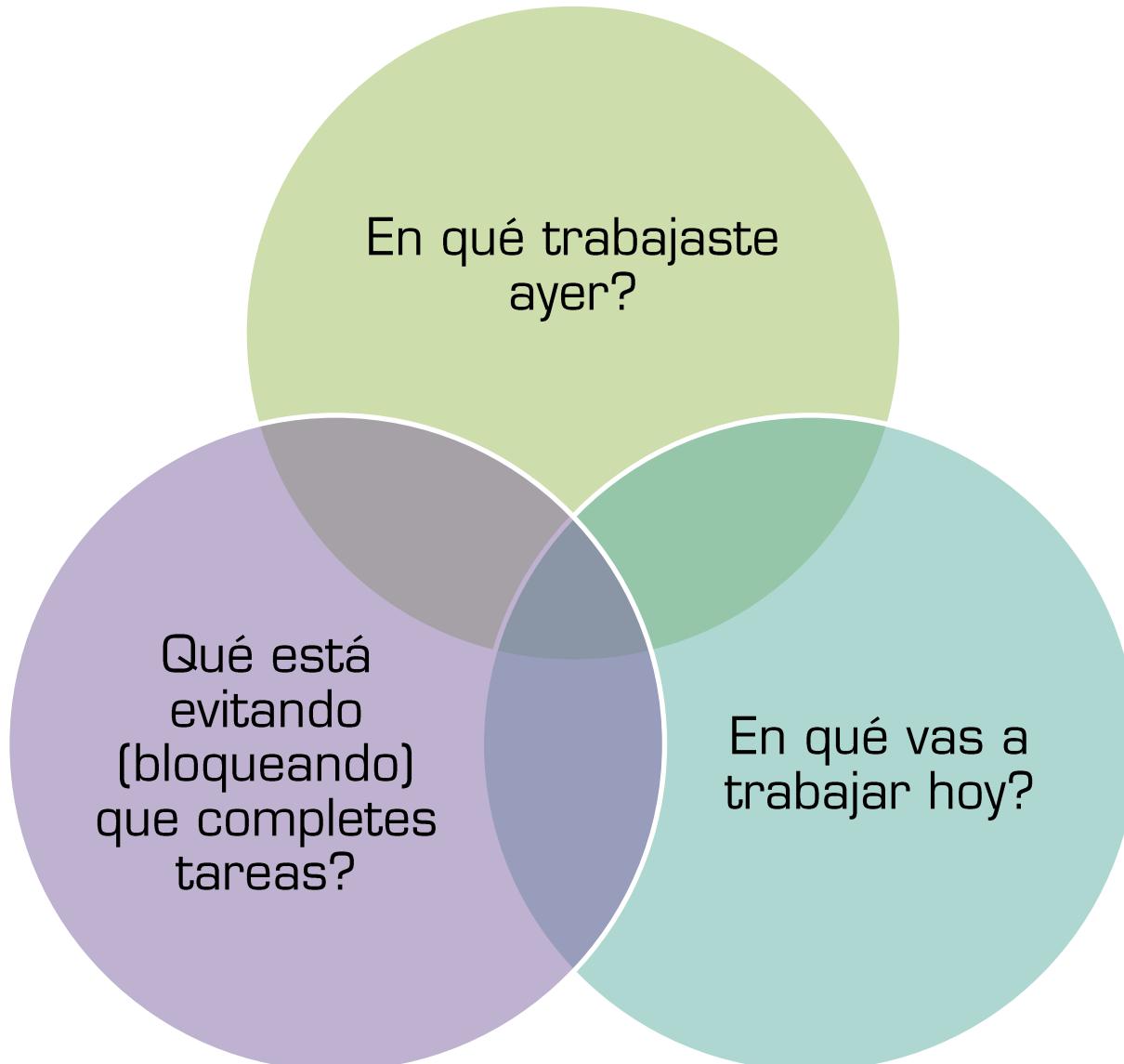
Pendiente	Ongoing	Hecho
<p>Role Information Updated</p> <p>Status Report A6 Template</p> <p>Initiative Form Fulfilled</p>	<p>Acceptance Criteria Samples</p> <p>Gantt Plan Samples</p> <p>Risk A6 Template</p>	<p>Core Exercise Decided</p> <p>Requisite Sample Template</p> <p>Risk Samples</p> <p>Review Team Deliverables</p>



## Generando nuestro canvas

- Genera tu canvas y apunta las tareas allí

# Daily Meeting



# Daily Meeting

**PRODUCT BACKLOG**

- View Grades, current semester**  
As a student I can see my grades online so that I don't have to wait until I get to school to know whether I'm passing.
- Acceptance Criteria:** Columns align neatly on FingerFly 4.1 and iPhone.  
**EFFORT: SMALL**
- Update Grades, current semester**  
As a teacher, I can update grades online so I don't depend on administrators to do it for me.  
**EFFORT: MEDIUM**
- View Grades, previous semester**  
**EFFORT: SMALL**
- Attendance**  
**EFFORT: MEDIUM**
- GPA**  
**EFFORT: SMALL**
- Report Cards**  
**EFFORT: EXTRA LARGE**
- Event Calendar**
- Alumni Archives**

**SPRINT BACKLOG**

COMMITTED BACKLOG ITEMS	NOT STARTED	IN PROGRESS	COMPLETED

**SPRINT TIMEBOX**

**Two weeks**

M	T	W	Th	F	M	T	W	Th	F
Sprint Planning Meeting					Sprint Planning Meeting				
					Building Deliverables	Reviewing Deliverables			
							Sprint Review Meeting	Sprint Retrospective Meeting	

**MEETING TIMEBOX**

**SCRUMMASTER**

**SPRINT EXECUTION**

**Development Team**

**Product Owner**

**ScrumMaster**

**Meeting Timebox**



## Pongámoslo en práctica

- Simulemos un daily meeting

# 4

## TÉCNICAS DE DESARROLLO DE SOFTWARE: XP, TDD

# eXtreme Programming (XP)

# eXtreme Programming

- Proceso software ligero
  - Diseñado para entornos dinámicos
  - Ideal para equipos pequeños (hasta 10 programadores)
  - Basado en el código
  - Alta dependencia en la comunicación informal, verbal
  - Creado por Kent Beck para la plantilla del proyecto C3 en Chrysler
- Valores que intenta fomentar la filosofía XP:
  - Comunicación
  - Simplicidad
  - Retroalimentación
  - Coraje

# Actores

- Programador (Programmer)
  - Responsable de decisiones técnicas
  - Responsable de construir el sistema
  - Sin distinción entre analistas, diseñadores o codificadores
  - En XP, los programadores diseñan, programan y realizan las pruebas
- Cliente (Customer)
  - Es parte del equipo
  - Determina qué construir y cuándo
  - Escribe tests funcionales para determinar cuándo está completo un determinado aspecto

# Actores

- Entrenador (Coach)
  - El líder del equipo - toma las decisiones importantes
  - Principal responsable del proceso
  - Tiende a estar en un segundo plano a medida que el equipo madura
- Rastreador (Tracker)
  - Metric Man
  - Observa sin molestar
  - Conserva datos históricos
- Probador (Tester)
  - Ayuda al cliente con las pruebas funcionales
  - Se asegura de que los tests funcionales se ejecutan

# Proceso

Mientras(sistema\_es\_útil) {

## Captar requisitos

- ✓ User Stories
- ✓ Metaphor

## Planificar

- ✓ Release planning
- ✓ Iteration planning

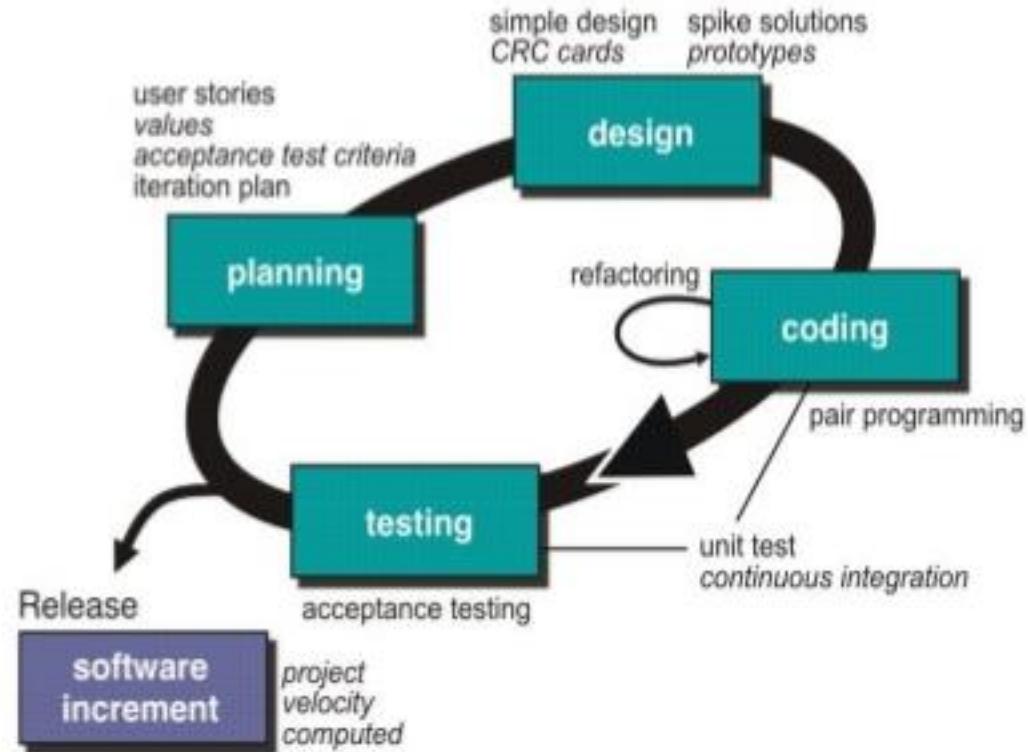
## Desarrollar

- ✓ Programming

## Presentar la entrega

- ✓ Releasing

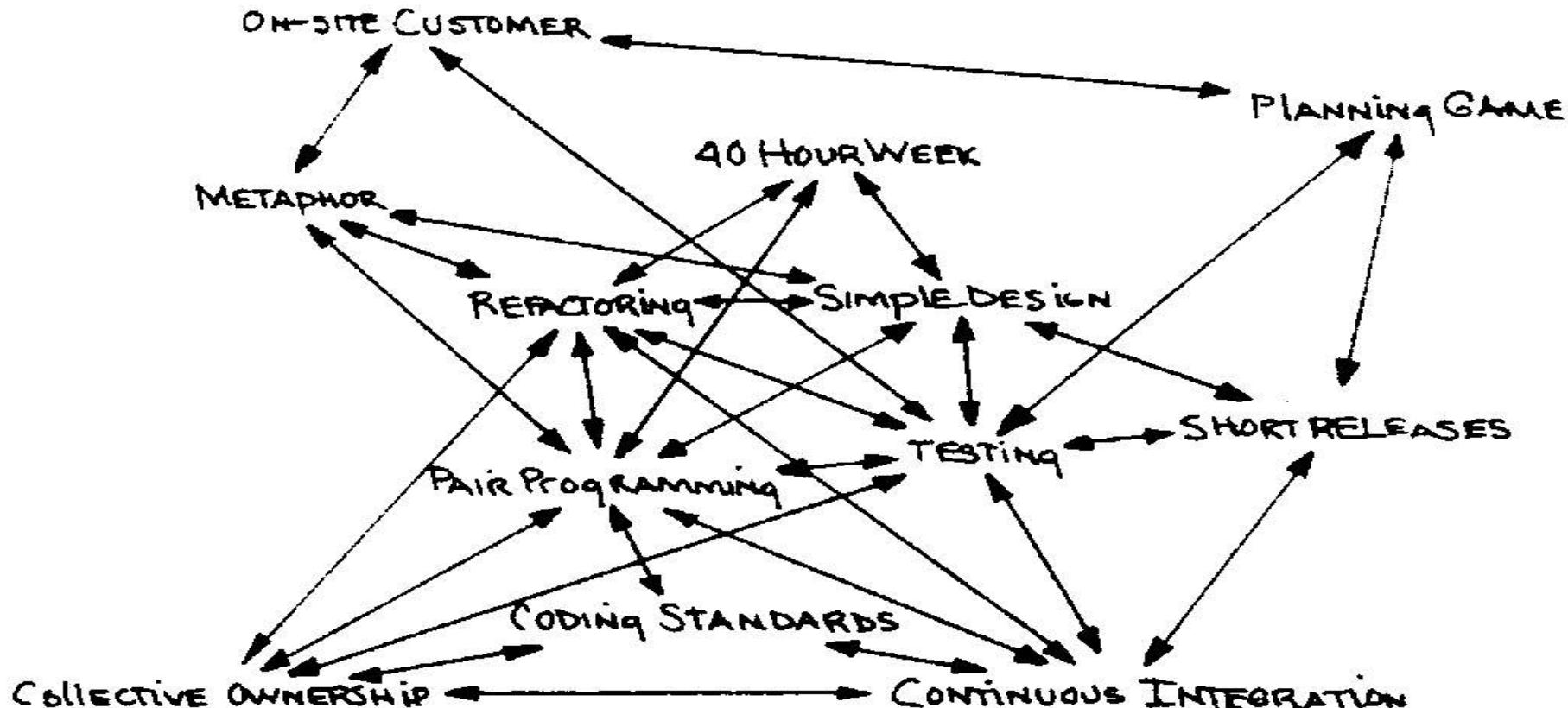
}



# Prácticas clave en XP

- El juego de planificación (The planning game)
- Entregas pequeñas (Short releases)
- Metáfora (Metaphor)
- Diseños simples (Simple designs)
- Pruebas (Testing)
- Refactorización (Refactoring)
- Programación en parejas (Pair programming)
- Dominio colectivo del código  
(Collective code ownership)
- Integración continua (Continuous integration)
- Semana de 40 horas (40-hour week)
- Cliente in situ (On site customer)
- Estándares de codificación (Coding standard)

# La gran foto



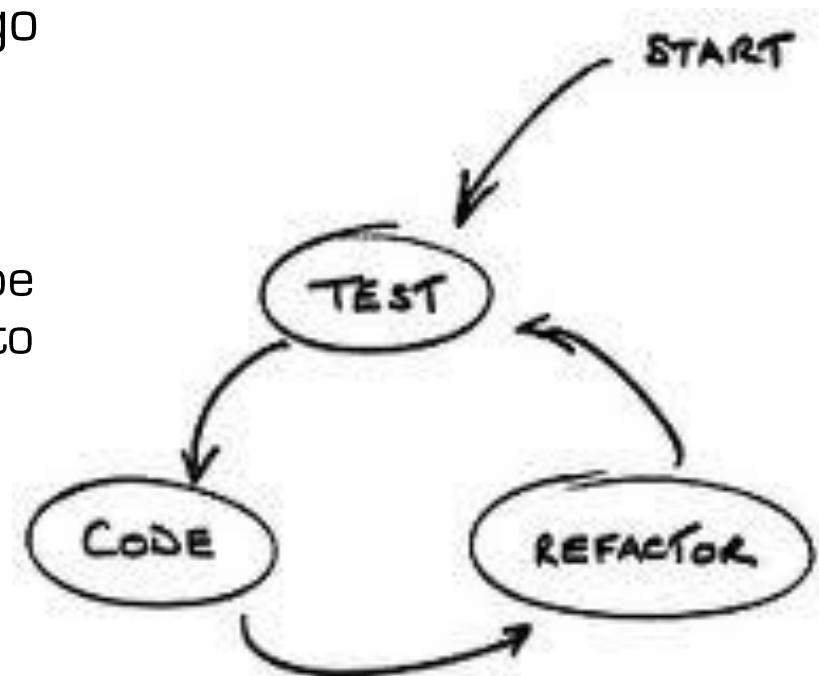
# Test Driven Development (TDD)

# Qué es TDD?

- Proceso iterativo en el cual el desarrollo está guiado por los test.
- Primero escribimos los test que expresan los requerimientos a cumplir luego desarrollamos para cumplir con dichos requerimientos.
- <http://www.youtube.com/watch?v=hWeKICBQuOk>

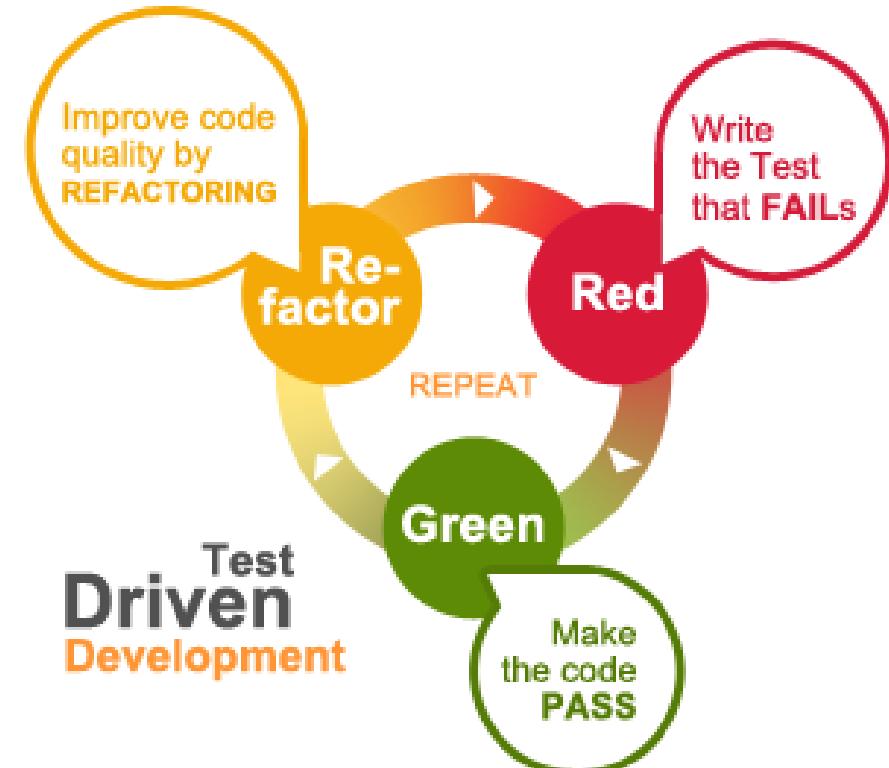
# Reglas TDD

- Nunca escribir una línea de código a menos que tengamos un test fallando.
  - Los tests representan los requerimientos que el código debe satisfacer, si no hay requerimiento es porque no hay nada que implementar.
- Eliminar duplicación de código.

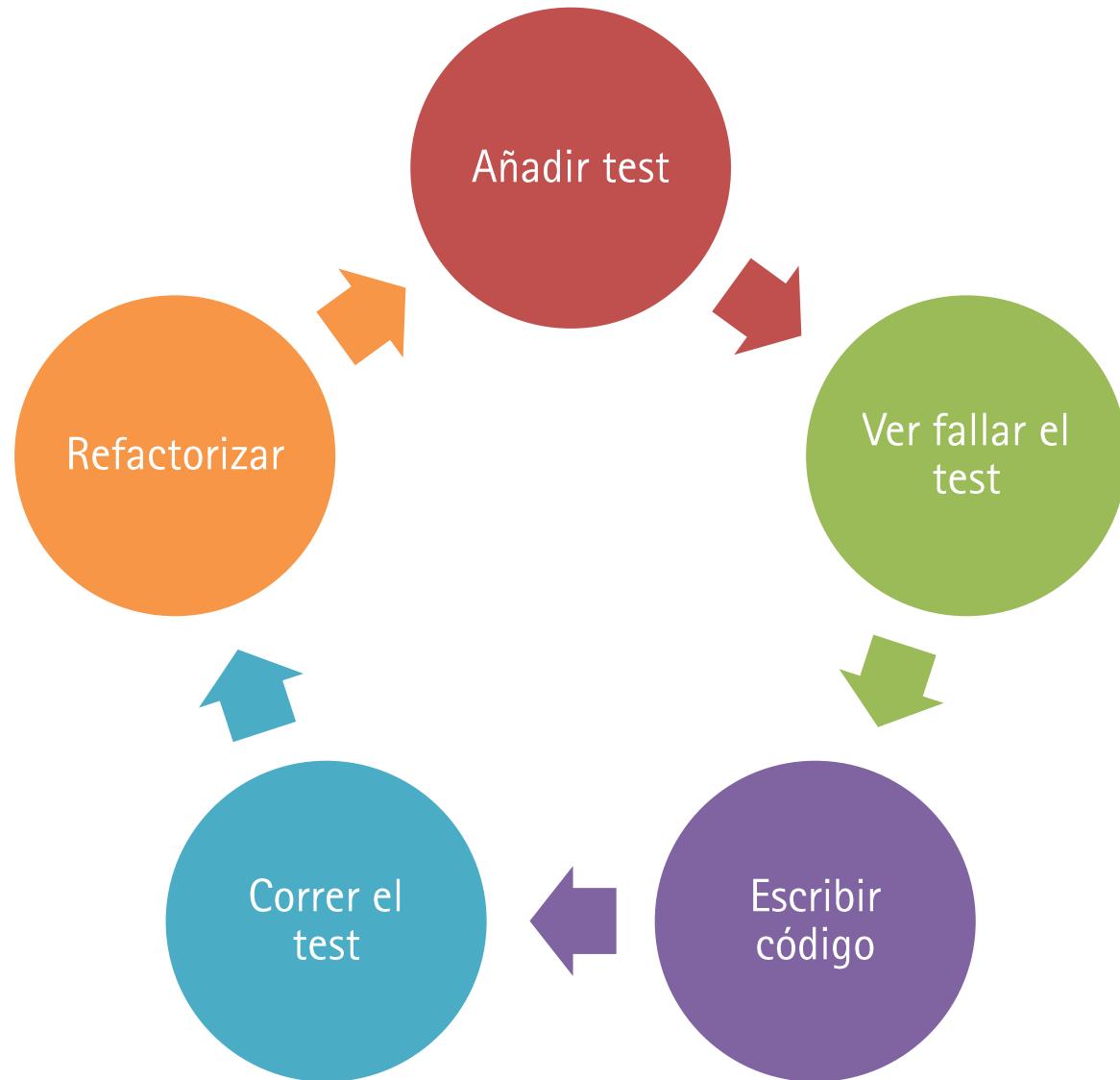


# TDD como práctica metodológica

- › **Test-First:** las pruebas se escriben antes del propio código a probar.
- › **Automatización:** las pruebas del programa deben ser hechas en código, y con la sola ejecución del código de pruebas debemos saber si lo que estamos probando funciona bien o mal.
- › **Refactorización posterior:** para mantener la calidad del diseño, se cambia el diseño sin cambiar la funcionalidad.



# Proceso





## Ejemplo Sumador

- Queremos desarrollar un programa que sume 2 números

# Discutamos decisiones como desarrolladores



- Decisiones sobre el sumador:
  - ¿Usaremos una llamada a un función?
  - ¿Cuál será el nombre de esa función?
  - ¿Qué parámetros tendrá y de qué tipo serán?

# Definimos los tests



- Verificar que llamando al programa Sumador con 2 y 2 da como resultado 4
- Verificar que llamando al programa Sumador con 2 y 0 da como resultado 2
- Verificar que llamando al programa Sumador con 2 y null da como resultado 2
- Verificar que llamando al programa Sumador con null y null da como resultado 0
- Verificar que llamando al programa Sumador con 'hola' y 'adiós' da como resultado 0

# Decisiones como desarrolladores



## ➤ Añadir test:

Verificar que llamando al programa Sumador con 2 y 2 da como resultado 4

## ➤ Ver fallar el test:

➤ Como no existe Sumador el test falla

## ➤ Escribir código: Escribimos el código mínimo (más corto) para que la prueba pase con éxito

```
function Sumador (a,b) {  
    return 4;  
}
```



# Decisiones como desarrolladores

## ➤ Correr el test:

Verificar que llamando al programa **Sumador** con 2 y 2 da como resultado 4

- Pasa el test!
- Refactorizar
  - Nombres de parámetros más descriptivos?

```
function Sumador (numero1,numero2) {  
    return 4;  
}
```



# Añadimos nuevo test

- Repetir el proceso con

Verificar que llamando al programa Sumador con  
2 y 0 da como resultado 2



## Pongámoslo en práctica

- Queremos un programa que concatene todos los elementos de un array con una coma
- Define un conjunto de tests (piensa en todas las posibilidades) para el programa y realiza el proceso de manera iterativa hasta completar un programa exitoso

# Ventajas

- Escribir las pruebas antes del código a probar minimiza el condicionamiento del autor por lo ya construido.
- También da más confianza al desarrollador sobre el hecho de que el código que escribe siempre funciona.
- Escribir las pruebas antes del código a probar permite especificar el comportamiento sin restringirse a una única implementación.
- La refactorización constante facilita el mantenimiento de un buen diseño a pesar de los cambios que, en caso de no hacerla, lo degradarían.

## En consecuencia...

- ninguna funcionalidad futura debería escribirse por adelantado, si no se tiene el conjunto de pruebas que permita verificar su corrección.
- Si a eso se le suma que sólo se debería escribir de a una prueba por vez, tenemos un desarrollo incremental extremo, definido por pequeños incrementos que se corresponden con funcionalidades muy acotadas.



[...]**netmind**

WeKnowIT

Barcelona

C. Almogàvers, 123  
08018 Barcelona  
Tel. 93 304.17.20  
Fax. 93 304.17.22

Madrid

Plaza Carlos Trías Bertrán, 7  
28020 Madrid  
Tel. 91 442.77.03  
Fax. 91 442.77.07

[www.netmind.es](http://www.netmind.es)



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE ENERGÍA, TURISMO  
Y AGENDA DIGITAL

**red.es**



ESTRATEGIA DE  
EMPRENDIMIENTO Y  
EMPLEO JUVENIL  
*garantía juvenil*



Agenda Digital para España



**UNIÓN EUROPEA**

Fondo Social Europeo  
*"El FSE invierte en tu futuro"*