

# Software Project Management

Bob Hughes and Mike Cotterell

Fifth Edition

# **Selection of an Appropriate Project Approach**

## **Chapter Four**

# Outline of Lecture

→ *Entae comprar y construir un software.*

- Building versus buying software

→ *Teniendo en cuenta las características del proyecto.*

- Taking account of the characteristics of the project

→ *modelos de procesos (forma de trabajar).*

- Process models

- Waterfall

- Prototyping and iterative approaches

- Incremental delivery

- Agile approaches

El desarrollo de software interno generalmente significa que:

- 👉 los desarrolladores y los usuarios pertenecen a la misma organización;
- 👉 la aplicación se incluirá en una cartera de sistemas informáticos existentes;
- 👉 Las metodologías y tecnologías están dictadas en gran medida por las normas y políticas de la organización, incluida la arquitectura empresarial existente.

Suelen estar en la estantería.

## Selection of Project Approaches

- correcto enfoque para un proyecto.  
• This lecture concerned with choosing the right approach to a particular project: variously called *technical planning*, *project analysis*, *methods engineering* and *methods tailoring*
- In-house: often the methods to be used dictated by organizational standards
- Suppliers: need for tailoring as different customers have different needs
  - ↳ incrementos de precios ¿?



Sin embargo, un proveedor de software podría llevar a cabo sucesivos proyectos de desarrollo para una variedad de clientes externos. Deberían revisar las metodologías y tecnologías que se utilizarán para cada proyecto individual.

→ *Análisis de proyectos... tecnologías usadas etc...*

cualquier característica del nuevo proyecto que requiera un enfoque diferente de proyectos anteriores deben tenerse en cuenta.

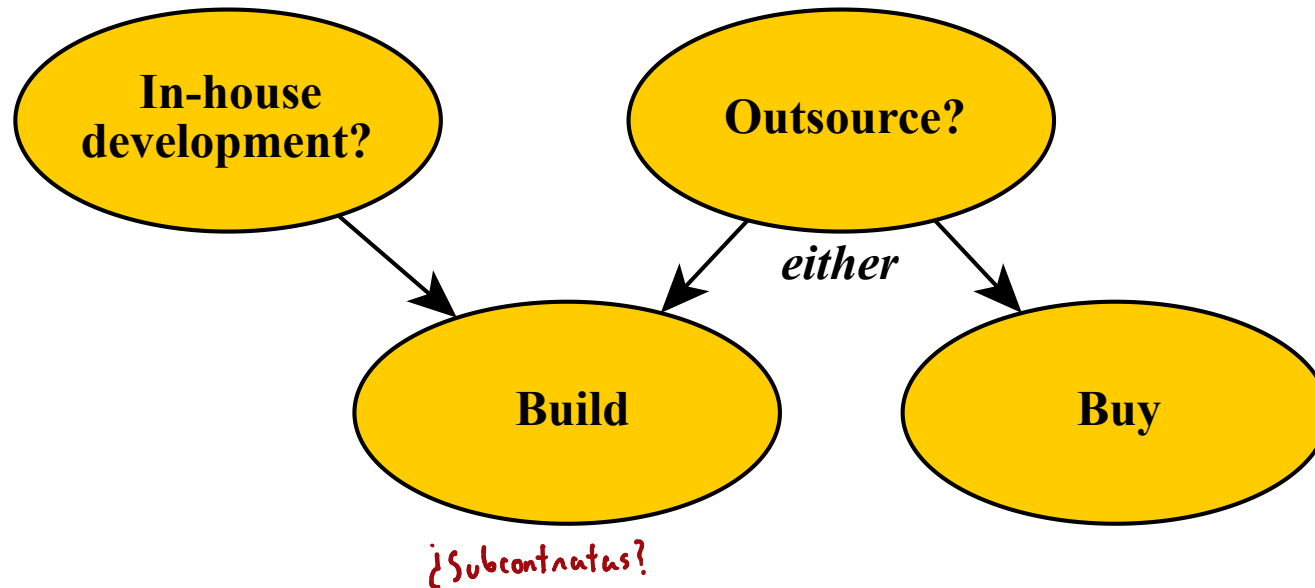
→ *Ejemplo del Sastre.*

*Si quieres un traje barato vas al ganso que fabrican miles...*  
*Si quieres exclusividad pues sastre por ejemplo...*

Software Project Management - 04.01 Introduction



# Build or Buy?



Proyectos de estanteria vs new development.  
en el marco de analisis de proyecto (paso a seguir en el ciclo  
del proyecto).

Ventajas de coger un proyecto  
de la estanteria... lo haces una vez y lo vendes 1000...

Un saftne... o una  
tira de del camofoun  
que hee en miles

# Some Advantages of Off-The-Shelf (OTS) Software

- Cheaper as supplier can spread development costs over a large number of customers
- Software already exists
- Can be trialed by potential customer
- No delay while software being developed
- Where there have been existing users, bugs are likely to have been found and eradicated

# Some Possible Disadvantages of Off-The-Shelf

- Customer will have same application as everyone else: no competitive advantage, *but* competitive advantage may come from the *way* application is used
- Customer may need to change the way they work in order to fit in with OTS application
- Customer does not own the code and cannot change it
- Danger of over-reliance on a single supplier

sistema de  
información =

sería especialmente apropiado cuando el proyecto emplea una gran cantidad de personal de desarrollo cuyo trabajo deberá ser coordinado: el método establece en detalle las actividades y productos necesarios. en cada paso.

# General Approach

- Look at risks and uncertainties e.g.
  - Are requirements well understood? *hemos entendido bien los requisitos?*
  - Are technologies to be used well understood? *hemos entendido bien las tecnologías?*

- Look at the type of application being built e.g.

- Information system? embedded system? *debe ser de hacer funcionar el proyecto en algún sistema.*

*donde la seguridad y confiabilidad son esenciales. (Podría justificar gasto adicional).*

- Criticality? differences between target and development environments?

*En entorno donde lo haces te hará restricciones. Necesitas tiempo - Respuesta rápida.*

- Clients' own requirements *El cliente me pide que lo haga con unas determinadas tecnologías.*
  - Need to use a particular method <!-- uniform applications and technologies, standards, accreditations-->

Es necesario emprender una serie de actividades interrelacionadas para crear un producto final.

Estas actividades se pueden organizar de diferentes formas y podemos llamar a estos modelos de proceso. métodos y especifica cómo se aplicarán.

El planificador selecciona métodos y especifica cómo se aplicarán.

Sistema de acción  
↑

## Choice of Process Models

- *Modelo de proceso de una sola pasada de en la escalera.*  
'Waterfall' also known as 'one-shot', 'once-through'
- Incremental delivery *→ vas entregando "muestras" poco a poco al cliente.*
- Evolutionary development *→ ¿if user?*

Also use of 'agile methods' e.g. extreme programming  
*en convivencia con lo anterior me parece.*

Usamos metodología ágil?  
Creo que esto es independiente de  
si usamos un modelo waterfall o...

Entrega rápida...

# Structure versus Speed of Delivery

hacen software robusto, no fallan...  
pero esta va en contra de la rapidez de la entrega...  
→ ¿esa duda, entre rapidez - Robustez.

## Structured Approach

métodos ágiles que se centra en los  
procesos ligeros.

- Also called <sup>de peso pesado</sup> 'heavyweight' approaches
- Step-by-step methods where each step and intermediate product is carefully defined

↑ Proyectos estructurados  
más antiguos (ya ahora se suele usar métodos  
ágiles).

- Emphasis on getting quality right first time

→ Usar XML.

- Example: use of UML and USDP

→ Automatizar la construcción a partir del diseño. dar un botón en el ashta y que se construya.

- Future vision: Model-Driven Architecture (MDA). UML supplemented with Object Constraint Language, press the button and application code generated from the UML/OCL model



existe un enfoque contrastante que es el intento de crear arquitecturas impulsadas por modelos (MDA). El desarrollo del sistema mediante MDA implica la creación de un modelo independiente de la plataforma (PIM) que especifica la funcionalidad del sistema mediante diagramas UML complementados con información adicional registrada en Object Constraint Language (OCL). Un PIM es la estructura lógica que debe aplicarse independientemente del entorno de software y hardware en el que se implementará el sistema. Esto se puede transformar en un modelo específico de plataforma (PSM) que tiene en cuenta un entorno de desarrollo e implementación particular. Luego, un PSM se puede transformar en código ejecutable para implementar un sistema que funcione.

# Structure versus Speed of Delivery

*intentan llegar al producto,  
fomentando el desarrollo rápido...  
No te fijas tanto en los requisitos.*

*Si te dan el producto de golpe... esto después de 3 años  
ya no se puede cambiar...  
En el método ágil cada X días... se reúne con el  
cliente... y le dice... que te parece?*

## Agile Methods

- Emphasis on speed of delivery rather than documentation
- **RAD**, Rapid Application Development emphasized use of quickly developed prototypes
- **JAD**, Joint Application Development. Requirements are identified and agreed in intensive workshops with users

Debido al esfuerzo adicional necesario y su mayor aplicabilidad a proyectos grandes y complejos, estos a menudo se denominan métodos de peso pesado. *(estructurados)*.

Se podría pensar que los usuarios generalmente agradecerían el enfoque más profesional que implican los métodos estructurados. Sin embargo, a los clientes de software les preocupa que las aplicaciones de trabajo se entreguen rápidamente y a un costo menor y, a menudo, ven los métodos estructurados como innecesariamente burocráticos y lentos. Una respuesta a esto ha sido el desarrollo rápido de aplicaciones (RAD), que hace hincapié en la producción rápida de prototipos del software para que los usuarios los evalúen.

*RAD → Producción rápida de prototipos.*

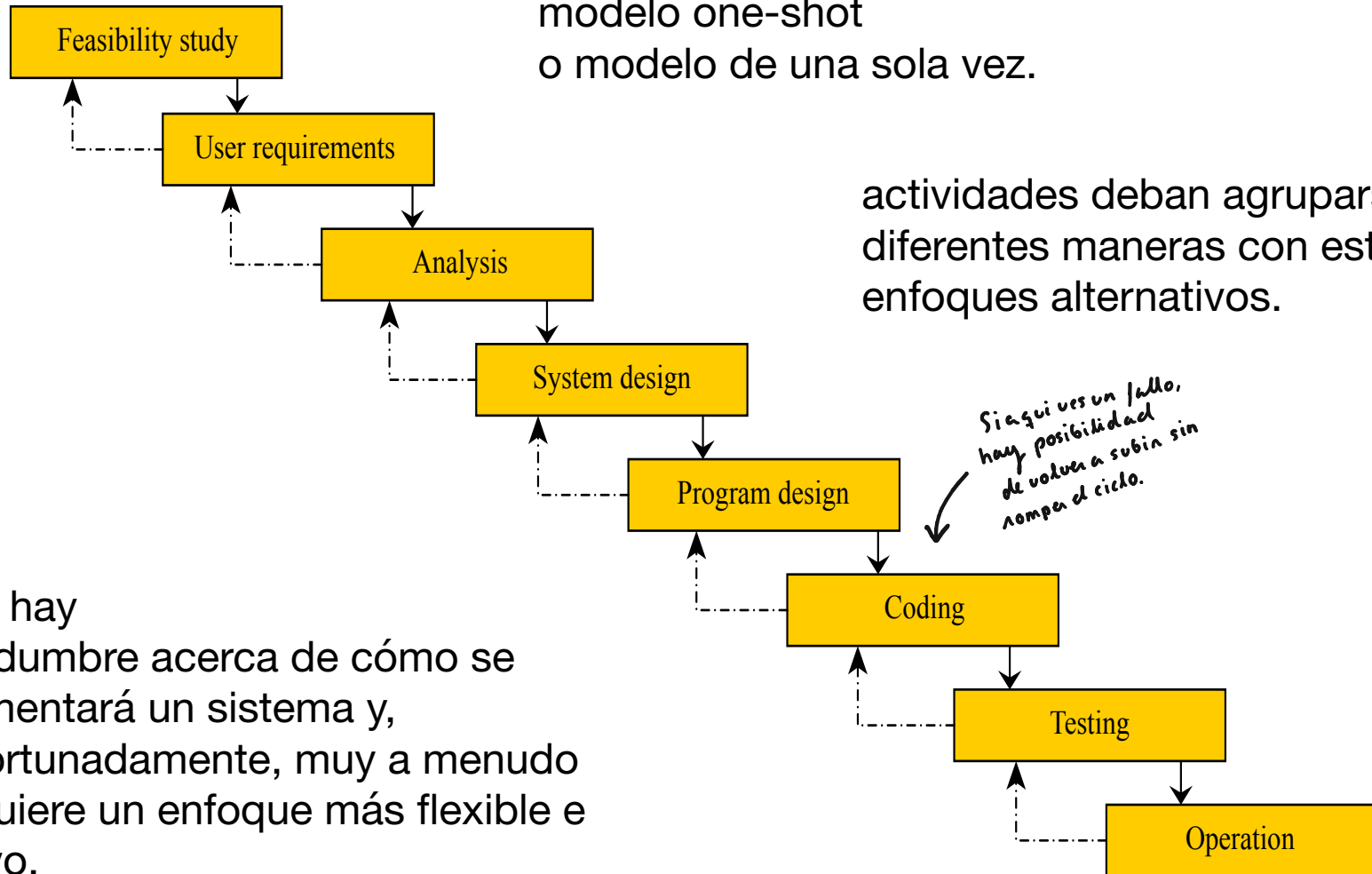
El enfoque RAD no excluye el uso de algunos elementos de métodos estructurados, como la redacción de diagramas de estructura de datos lógicos, sino que también adopta tácticas como talleres de desarrollo de aplicaciones conjuntas (JAD).

*→ Aunque RAD sea rápido,  
también hace los diagramas...  
incluyen enfoques JAD.*

(JAD). En estos talleres, los desarrolladores y los usuarios trabajan juntos de manera intensiva durante, digamos, de tres a cinco días e identifican y acuerdan los requisitos del sistema completamente documentados. A menudo, estos

# Waterfall (cascada).

Este es el modelo "clásico" de desarrollo de sistemas que también se conoce como el modelo one-shot o modelo de una sola vez.



donde hay incertidumbre acerca de cómo se implementará un sistema y, desafortunadamente, muy a menudo se requiere un enfoque más flexible e iterativo.

Secuencia de actividades que funcionan de arriba a abajo. El diagrama muestra algunas flechas apuntando hacia arriba y hacia atrás. Esto indica que una etapa posterior puede revelar la necesidad de un trabajo adicional en una etapa anterior, pero esto definitivamente debería ser la excepción y no la regla. Después de todo, el flujo de una cascada debe ser hacia abajo, con la posibilidad de que solo salpique un poco. El alcance limitado de la iteración es, de hecho, uno de los puntos fuertes de este

# Waterfall

- The 'classical' model
- Imposes structure on the project
- Every stage needs to be checked and signed off
- BUT
- Limited scope for iteration
- V model approach is an extension of waterfall where different testing phases are identified which check the quality of different development phases

→ El modelo en V

Vas pasando etapas y  
cuando tienes el software comprobas cada capa.

- Cascada
- V
- Estrategia Evolutiva.

# Evolutionary Delivery: Prototyping

Sprague and McNurlin

*'An iterative process of creating quickly and inexpensively live and working models to test out requirements and assumptions'*

Main types

prueba algunas ideas y luego se descarta cuando se inicia el verdadero desarrollo del sistema

- 'Throw away' prototypes
- Evolutionary prototypes

El prototipo se desarrolla y modifica hasta que finalmente se encuentra en un estado en el que puede convertirse en la operación.

Ésta es una forma en la que podemos comprar conocimiento y reducir la incertidumbre. Un prototipo es un modelo de trabajo de uno o más aspectos del sistema proyectado. Se construye y se prueba de forma rápida y económica para probar supuestos. Son desechables o evolutivos.

## Reasons for Prototyping

*Podemos mirar atrás en una tarea y ver donde hemos cometido errores.*

*los prototipos son buena idea...*

*le enseñar un prototipo*

El prototipo prueba algunas ideas y luego se descarta cuando se inicia el verdadero desarrollo del sistema.

- Learning by doing
- Improved communication *¿Que sabes hacer? ¿Como lo haces?*
- Improved user involvement *el usuario ya ha visto el producto / prototipo y te dice lo que le gusta / no ...*
- A feedback loop is established *hay realimentación.*
- Reduces the need for documentation *mejor hablar que un documento.*
- Reduces maintenance costs i.e. changes after the application goes live
- Prototype can be used for producing expected results

Los usuarios pueden malinterpretar la función del prototipo. Por ejemplo, pueden esperar que el prototipo tenga una validación de entrada tan estricta o una respuesta tan rápida como el sistema operativo, aunque esto no fue lo que pretendía.

*Puede que el cliente  
no entienda que es un  
prototipo.*

## Prototyping: some Dangers

- Users may misunderstand the role of the prototype
- Lack of project control and standards possible
- Additional expense of building prototype
- Focus on user-friendly interface could be at expense of machine efficiency



Es fundamental identificar desde el principio qué se debe aprender del prototipo.

Los estudiantes de informática a menudo se dan cuenta de que el software que van a escribir como parte de su proyecto de último año no puede ser utilizado de forma segura por usuarios reales. Por lo tanto, llaman al software un "prototipo". Si se trata de un embargo, si se trata de un prototipo real, deben:especifique lo que esperan aprender del prototipo; planificar cómo se evaluará el prototipo; informar sobre lo que realmente se ha aprendido.

# Other Ways of Categorizing Prototyping

- What is being learnt?
  - Organizational prototype
  - Hardware/software prototype ('experimental')
  - Application prototype ('exploratory')
- To what extent?
  - Mock-ups
  - Simulated interaction
  - Partial working models: vertical versus horizontal

↳ Conocer nuevas técnicas de desarrollo.

Sería inusual que se creara un prototipo de toda la aplicación. La creación de prototipos generalmente simula solo algunos aspectos de la aplicación de destino. Por ejemplo, puede haber:

*la interfaz de usuario es el principal prototipo para el usuario...  
T6 se puede hacer prototipo del funcionamiento interno.*

*depende  
de lo que  
necesites...*

# What Is Being prototyped?

- Human-computer interface
  - The physical vehicle for the prototype should be as similar as possible to the operational system
- Functionality
  - The precise way the system should function internally is not known (real world simulation)
  - The algorithms might need to be adjusted repeatedly

# Controlling Changes during Prototyping

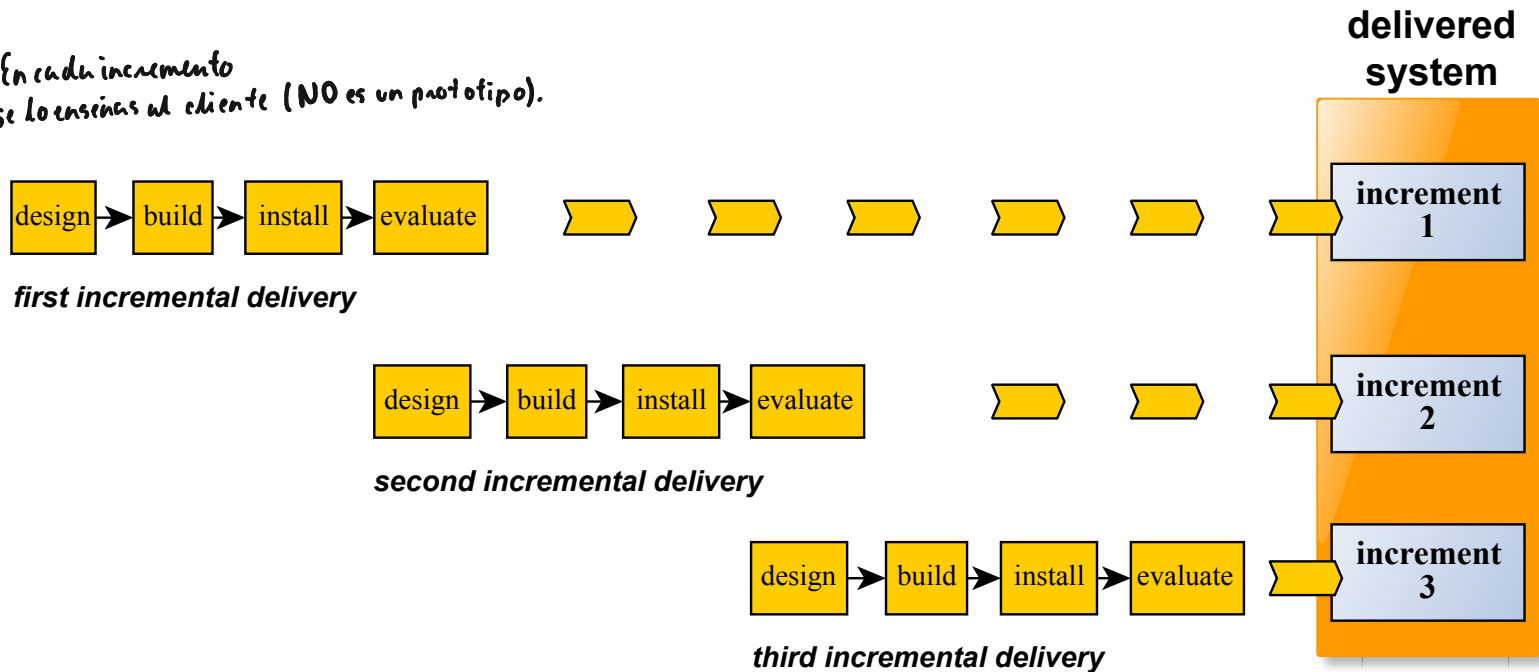
*• los cambios que sugieren el cliente:*

- Follow all user suggestions?
- **Cosmetic** ( $\approx 35\%$ )  
*cambian un boton por ejemplo.*
  - Simple changes to the layout of the screen or reports
- **Local** ( $\approx 60\%$ )
  - The way that a screen or report is processed, but does not affect other parts
  - Backed-up, inspected retrospectively
- **Global** ( $\approx 5\%$ )
  - Affects more than one part of the system
  - Must be subject of a design review before implemented

Este enfoque divide la aplicación en pequeños componentes que luego se implementado y entregado en secuencia. Cada componente entregado debe brindar algún beneficio al usuario.

# Incremental Delivery

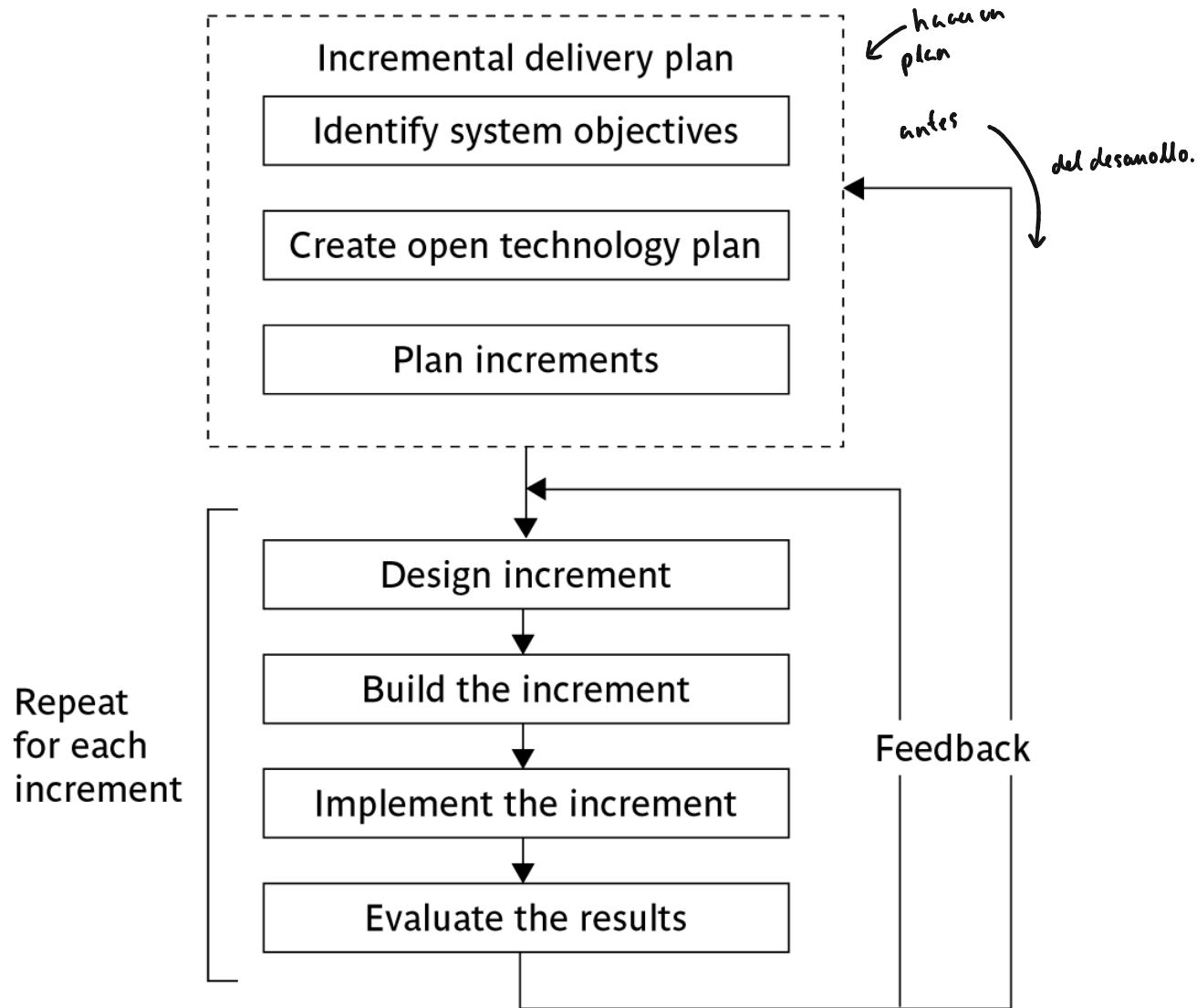
{en cada incremento  
se lo enseña al cliente (NO es un prototipo).



El time-boxing se asocia a menudo con un enfoque incremental. Aquí, el alcance de los entregables para un incremento está estrictamente limitado por un plazo acordado. Este plazo debe cumplirse, incluso a expensas de eliminar algunas de las funciones planificadas. Las funciones omitidas se pueden transferir a incrementos posteriores.

# The Incremental Process

Los módulos que se construyen y a no se hacen.



# Incremental Approach: Benefits

La retroalimentación de los primeros incrementos mejora las etapas posteriores.

- Feedback from early stages used in developing latter stages

La posibilidad de cambios en los requisitos se reduce debido al intervalo de tiempo más corto entre el diseño de un componente y su entrega.

- Shorter development thresholds

Los usuarios obtienen beneficios antes que con un enfoque convencional.

- User gets some benefits earlier

La entrega anticipada de algunos componentes útiles mejora el flujo de efectivo, ya que obtiene algún retorno de la inversión desde el principio.

- Project may be put aside temporarily

Los subproyectos más pequeños son más fáciles de controlar y gestionar.

- Reduces 'gold-plating' ↖

a solicitud de funciones que son innecesarias y que de hecho no se utilizan, es menor, ya que los usuarios saben que si una función no está en el incremento actual, puede incluirse en el siguiente.

But there are some possible disadvantages

- Loss of economy of scale
- 'Software breakage'

Una vez definidos los objetivos generales y un plan de tecnología abierta, la siguiente etapa es planificar los incrementos utilizando las siguientes pautas:

## The Outline Incremental Plan

*consisten en...*

- Steps ideally 1% to 5% of the total project
- Non-computer steps should be included
- Ideal if a step takes one month or less:
  - Not more than three months
- Each step should deliver some benefit to the user
- Some steps will be physically dependent on others

# Which Step First?

- Some steps will be pre-requisite because of physical dependencies
- Others may be in any order
- Value to cost ratios may be used  
*lo que me cuesta vs lo que lo valora el usuario.*
- V/C where
  - V is a score 1-10 representing value to customer
  - C is a score 0-10 representing value to developers



## V/C Ratios: an Example

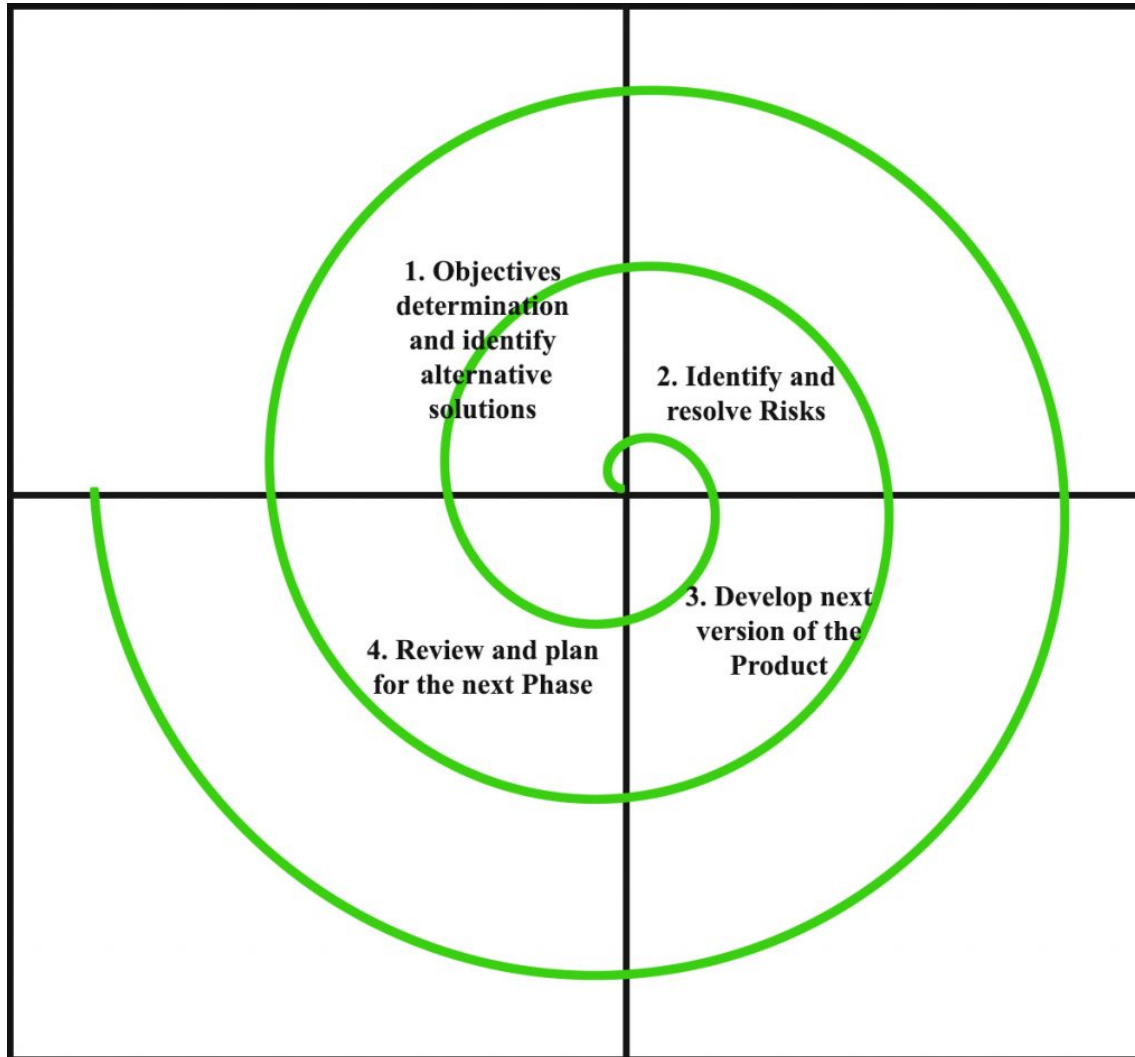
Step	Value	Cost	Ratio	Rank
Profit reports	9	1	9.00	2nd
Online database	1	9	0.11	5th
Ad Hoc enquiry	5	5	1.00	4th
Purchasing plans	9	4	2.25	3rd
Profit-based pay for managers	9	0	$\infty$	1st

*Of no modelo ...*

# The Spiral Model

- Spiral model is one of the most important *Software Development Life Cycle models*, which provides support for Risk Handling.
- It looks like a spiral with many loops
  - Number of loops unknown and depends on the project
  - Each loop as a **Phase** of the software development process.
- The Radius represents the cost of the project, and the angular dimension represents the *progress* made in the current phase.

# The Spiral Model



# Spiral Model as a Meta Model

- It subsumes all the other SDLC models
  - A single loop spiral actually represents the Iterative Waterfall Model
  - Uses the approach of Prototyping Model by building a prototype at the start of each phase as a risk handling technique.
  - Can be considered as supporting the evolutionary model – the iterations along the spiral can be considered as evolutionary levels through which the complete system is built.

# Advantages of Spiral Model

- Advantages
  - *Risk Handling* - unknown risks can be analyzed and handled at every phase
  - *Good for large projects* - recommended
  - *Flexibility in Requirements* - change in requirements can be incorporated at later phase
  - *Customer Satisfaction* - clients can see the development of the product at early phases

# Disadvantages of Spiral Model

- *Complex* - is much more complex than other SDLC models
- *Expensive*- not suitable for small projects as it is expensive
- *Too much dependable on Risk Analysis* - Without very highly experienced expertise, it is going to be a failure to develop a project using this model
- *Difficulty in time management* - As the number of phases is unknown

*Es mejor un producto acabado pero con fallos que no acabado y más limpio de fallos. Este modelo se realimenta con los clientes.*

*Me interesa una documentación completa... pero prefiero acabar antes.*

# 'Agile' Methods

*No queremos tardar. No queremos riesgo.*

Los métodos ágiles están diseñados para superar las desventajas que hemos observado con metodologías de implementación de peso pesado. Hay varios enfoques ágiles, que incluyen:

- 👉 Crystal technologies
- 👉 Atern (formerly DSDM) Feature-driven development
- 👉 Scrum
- 👉 Extreme Programming (XP)

Structured development methods have some perceived advantages

- Produce large amounts of documentation which can be largely unread
- Documentation has to be kept up to date
- Division into specialist groups and need to follow procedures stifles communication
- Users can be excluded from decision process
- Long lead times to deliver anything etc. etc

The answer? 'Agile' methods?

# 'Agile' Methods

- Crystal technologies
- Atern (formerly DSDM)
- Feature-driven development
- Scrum
- Extreme Programming (XP)



# The Agile Manifesto

- Individuals and interaction over processes and tools
- Working together over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

El Método de Desarrollo de Sistemas dinámicos (en inglés Dynamic Systems Development Method o DSDM) es un método que provee un framework para el desarrollo ágil de software, apoyado por su continua implicación del usuario en un desarrollo iterativo y creciente que sea sensible a los requerimientos cambiantes, para desarrollar un sistema que reúna las necesidades de la empresa en tiempo y presupuesto.

# Atern/Dynamic System Development Method (DSDM)

→ lo de antes.

SSADM (Método de diseño y análisis de sistemas estructurados)

- UK-based consortium
- *Arguably* DSDM can be seen as replacement for SSADM (*Structured SDM*)
- DSDM is more a project management approach than a development approach
- Can still use DFDs, LDSs etc!
- An update of DSDM has been badged as 'Atern'

Método de desarrollo de sistemas dinámicos (DSDM).

Como extensión del Desarrollo rápido de aplicaciones (RAD), DSDM se centra en los proyectos de sistemas de información que son caracterizados por presupuestos y agendas apretadas.

👉 estudio de viabilidad,  
👉 estudio de la empresa,  
👉 iteración del modelo funcional,  
👉 diseño e iteración de la estructura, e  
👉 implementación.

/uses de dsadm.

# Eight Core **Atern/DSDM** Principles

*muchas funcionalidades No a ponten venda de novo valor,*

*→ y hacen tardar mucho más.*

1. Focus on business need *tener límites temporales... mejor entregar algo que nada*

*→ En uno en cascada por ejemplo no existe límite temporal.*

2. Delivery on time – use of time-boxing

3. Collaborate *colaboración con stakeholders.*

*→ En cascada es en una única iteración (de toda la cascada). y cada paso requiere debes haber pasado por el anterior y completarlo.*

4. Never compromise quality *Nunca comprometer la calidad. No hay tiempo, ya se probará.*

5. Deliver iteratively *Entrega iterativa.*

6. Build incrementally

7. Communicate continuously

8. Demonstrate control

*demonstrar control.  
Por donde te llegas?*

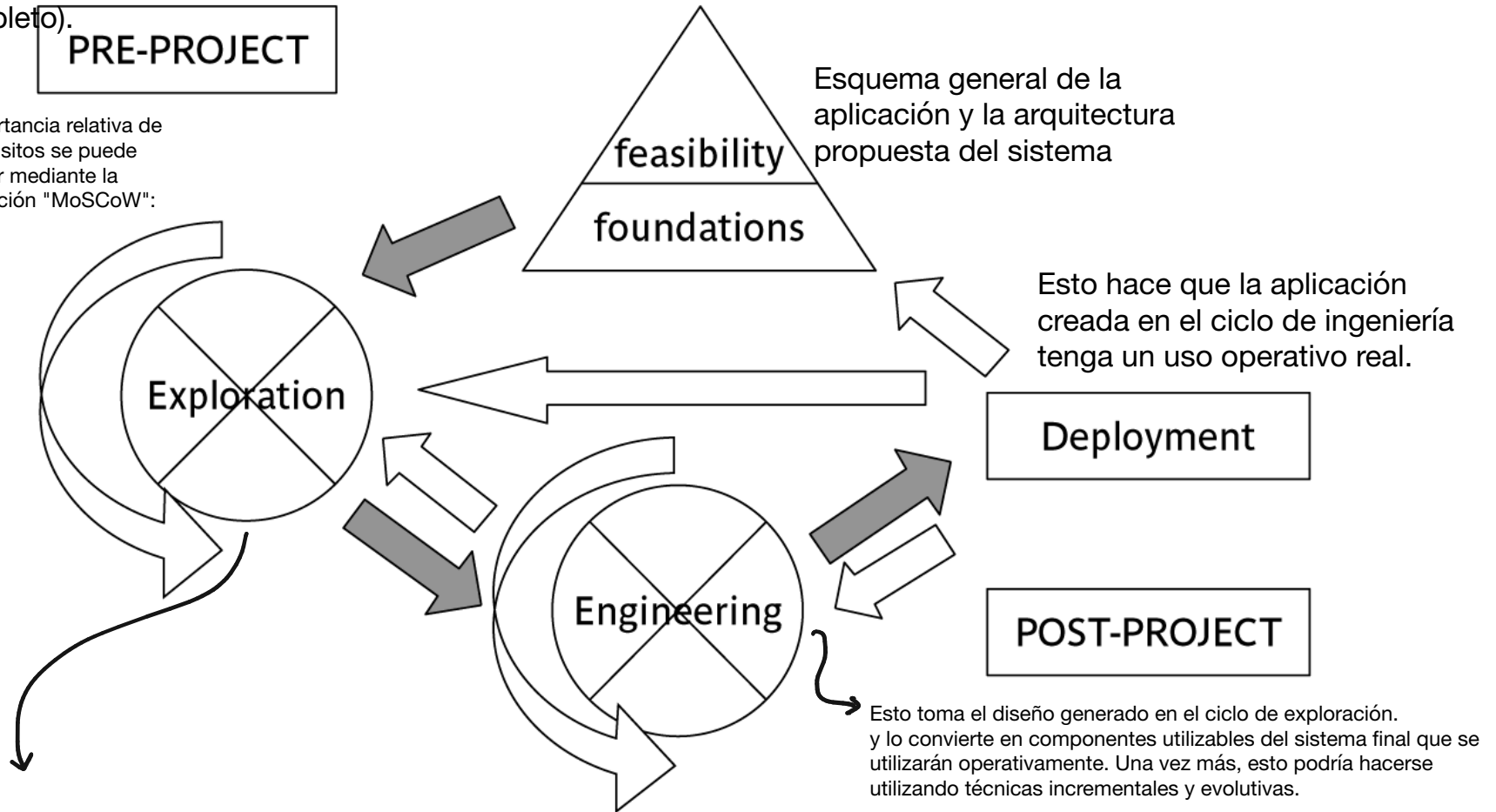
No solo puede haber iteraciones dentro de los ciclos de exploración e ingeniería, sino que un incremento podría involucrar la investigación de requisitos seguida de la construcción de la funcionalidad.

# Atern/DSDM Framework

Se recordará que para cumplir con el plazo impuesto por una caja de tiempo, la implementación de características menos importantes puede posponerse para incrementos posteriores (o incluso descartarse por completo).

**PRE-PROJECT**

La importancia relativa de los requisitos se puede clasificar mediante la clasificación "MoSCoW":



Esto investiga los requisitos comerciales. Estos requisitos se traducen en un diseño viable para la aplicación. Este podría ser un proceso iterativo que podría implicar la creación de prototipos exploratorios.

# Atern/DSDM: Time-Boxing

- **Time-box** fixed deadline by which something has to be delivered
- Typically 2 to 6 weeks time La importancia relativa de los requisitos se puede clasificar mediante la clasificación "MoSCoW":
- MoSCoW priorities
  - **Must have** - essential features
  - **Should have** - very important, but system could operate without
  - **Could have** - can be delayed with some inconvenience
  - **Want** - their delay to a later increment is accepted (but probably won't get!)

# Extreme Programming Core Values

## 1. Communication and feedback

Se argumenta que el mejor método de comunicación es la comunicación cara a cara. Además, la mejor manera de comunicar a los usuarios la naturaleza del software en producción es proporcionarles frecuentes incrementos de trabajo. Se evita la documentación formal.

- Best face-to-face

*incrementos frecuentes*

- Provide users with frequent working increments

## 2. Simplicity

Siempre se debe adoptar el diseño más simple que implemente los requisitos de los usuarios. No debe dedicarse ningún esfuerzo a tratar de satisfacer las posibles necesidades futuras que, en cualquier caso, es posible que nunca se materialicen.

- Simplest design that implements requirements
- Not spend efforts in future possible needs

## 3. Responsibility

Los desarrolladores son los responsables en última instancia de la calidad del software, no, por ejemplo, algunos grupos de control de calidad o pruebas del sistema. Los desarrolladores son los responsables en última instancia de la calidad del software, no, por ejemplo, algunos grupos de control de calidad o pruebas del sistema.

- Developers ultimately responsible for software quality


## 4. Courage

lo suyo es el coraje para tirar por la borda un trabajo en el que ya se ha invertido mucho esfuerzo, y empezar con un diseño fresco si eso es lo que se pide. También es el coraje para probar nuevas ideas; después de todo, si no funcionan, siempre pueden desecharse. Beck sostiene que es más probable que esta actitud conduzca a mejores soluciones.

- Throw away work done and start from scratch if needed

*Incremento como lanzamiento. (En XP).*

# Core Practices in XP

- El tiempo entre lanzamientos de funciones a los usuarios debe ser lo más breve posible.
- The planning exercise
  - Small releases
  - Metaphor
  - Simple design
  - Testing
  - Refactoring
- 

Anteriormente, cuando hablábamos de "incrementos", nos referíamos a componentes del \_ sistema que los usuarios realmente podían usar. XP se refiere a estos como lanzamientos.

Dentro de estas versiones, el código se desarrolla en iteraciones, períodos de uno a cuatro

semanas de duración durante las cuales se crean funciones específicas del software.

\_ Tenga en cuenta que normalmente no se trata de "iteraciones" en el sentido de que son versiones nuevas \_ mejoradas de la misma función, aunque es posible. El juego de planificación es el proceso mediante el cual las características que se incorporan en

se negocia el próximo lanzamiento. Cada una de las características está documentada en un

Breve descripción textual llamada historia que está escrita en una tarjeta. Se lleva a cabo un proceso similar al análisis de la relación valor-costos discutido anteriormente en la Sección 4.10 o la calificación MoSCoW de Atern para dar prioridad a las características. En el momento de la próxima publicación del código, se retendrá cualquier característica que no se haya completado, es decir, se empleará el recuadro de tiempo.

El sistema que se construirá será un código de software que refleje las cosas que existen y suceden en el mundo real. Una aplicación de nómina calculará y registrará los pagos a los empleados. Los términos utilizados para describir los elementos de software correspondientes deben, en la medida de lo posible, reflejar la terminología del mundo real; en un nivel muy básico, esto significaría usar nombres significativos para variables y procedimientos como "hourly\_rate" y "calculate\_gross\_pay".

# Core Practices in XP (ii)

- Pair programming *software más fiable si se programa dos personas mismo asunto.*

- Collective ownership 

Este es realmente el corolario de la programación por pares. El equipo en su conjunto asume la responsabilidad colectiva del código en el sistema. Una unidad de código no "pertenece" a un solo programador, que es el único que puede modificarla.

- Continuous integration 

Este es otro aspecto de la práctica de las pruebas. A medida que se realizan cambios en las unidades de software, las pruebas integradas se ejecutan con regularidad, al menos una vez al día, para garantizar que todos los componentes funcionen juntos correctamente.

- Forty-hour weeks
- On-site customers
- Coding standards



# Extreme Programming

*Realimentación al cliente temporal.*

- Increments of 1 to 3 weeks
  - Customer can suggest improvement at any point
- Argued that distinction between design and building of software are artificial
- Code to be developed to meet current needs only
- Frequent re-factoring to keep code structured

# Extreme Programming - (ii)

- Developers work in pairs
- Test cases and expected results devised before software design
- After testing of increment, test cases added to a consolidated set of test cases

# Limitations of Extreme Programming

- Reliance on availability of high quality developers
- Dependence on personal knowledge – after development knowledge of software may decay making future development less easy
- Rationale for decisions may be lost e.g. which test case checks a particular requirement
- Reuse of existing code less likely

El uso exitoso de XP se basa en ciertas condiciones. Si no existen, su práctica podría resultar difícil. Estas condiciones incluyen las siguientes.

= Debe haber fácil acceso a los usuarios, o al menos a un representante del cliente que sea un experto en el dominio. Esto puede resultar difícil cuando los desarrolladores y los usuarios pertenecen a organizaciones diferentes.

El personal de desarrollo debe estar ubicado físicamente en la misma oficina.

«Si los usuarios se enteran de cómo funcionará el sistema sólo al presentarles versiones funcionales del código, puede haber problemas de comunicación si la aplicación no tiene una interfaz visual.

m Para que el trabajo se secuencie en pequeñas iteraciones de trabajo, debe ser posible dividir la funcionalidad del sistema en componentes relativamente pequeños y autónomos.

a Los sistemas grandes y complejos pueden necesitar inicialmente un esfuerzo arquitectónico significativo. Esto podría excluir el uso de XP.

XP también tiene algunos problemas potenciales intrínsecos, particularmente con respecto a su dependencia de la experiencia y el conocimiento tácitos en contraposición al conocimiento externo en forma de documentación.

= Existe una dependencia de desarrolladores de alta calidad que hace que el desarrollo de software sea vulnerable si la rotación de personal es significativa.

w Incluso cuando la retención del personal es buena, una vez que se ha desarrollado e implementado una aplicación, el conocimiento tácito y personal del sistema puede decaer. Esto puede dificultar, por ejemplo, que el personal de mantenimiento sin documentación identifique qué partes del código modificar para implementar un cambio en los requisitos.

m Tener un depósito de datos de prueba completos y precisos y los resultados esperados puede no ser tan útil como podría esperarse si no se documenta la justificación de los casos de prueba particulares. Por ejemplo, cuando se realiza un cambio en el código, ¿cómo sabe qué casos de prueba deben cambiarse?

= Algunos entornos de desarrollo de software se han centrado en fomentar la reutilización de código como un medio para mejorar la productividad del desarrollo de software. Tal política parecería ser incompatible con XP.

Refuerza la idea  
equipo. No jerarquías.

# Scrum

cambia respecto  
del XP.

↳ Aquí los requeritos  
queden fijados en el Sprints y no  
se cambian.

- Named as an analogy to a rugby scrum – all pushing together
- Originally designed for new product development where 'time-to-market' is important
- 'Sprints' increments of typically one to four weeks
- Daily 'scrums' – daily stand-up meetings of about 15 minutes

Puesta al día de 15 min.

¿Cómo fue?

## Scrum (ii)

- Unlike XP, requirements are frozen during a sprint
- At the beginning of the sprint there is a sprint planning meeting where requirements are prioritized
- At end of sprint, a review meeting where work is reviewed and requirements may be changed or added to

Si la huer  
extensibility  
demande XP o Scrum  
noteable.

# Grady Booch's Concern

- Booch, an OO authority, is concerned that with requirements driven projects:

*'Conceptual integrity sometimes suffers because this is little motivation to deal with scalability, extensibility, portability, or reusability beyond what any vague requirement might imply'*

- Tendency towards a large number of discrete functions with little common infrastructure?

# Combinations of Approach

Construction	Ins	tala	tion
	One-shot	Incremental	Evolutionary
One-shot	yes	yes	<b>no</b>
Incremental	yes	yes	<b>no</b>
Evolutionary	yes	yes	yes

- One-shot or incremental installation – any construction approach possible
- Evolutionary installation implies evolutionary construction



# 'Rules of Thumb' about Approach to be Used

IF uncertainty is high  
THEN use evolutionary approach

IF complexity is high but uncertainty is not  
THEN use incremental approach

IF uncertainty and complexity both low  
THEN use one-shot

IF schedule is tight  
THEN use evolutionary or incremental

# Conclusion

- Examine each project carefully to see if it has characteristics which suggest a particular approach or process model
- Classic waterfall process model should lead to projects that are easy to control
- Prototyping may be able to reduce project uncertainties
- Incremental approach encourages the execution of a series of small, manageable 'mini-projects' but adds some cost