MDA 22-23: Parcial 1

Tema 1: Desarrollando software

1. ¿Qué es el software?

El software es un transformador de información, para ello: adquiere, gestiona, modifica, produce o trasmite información.

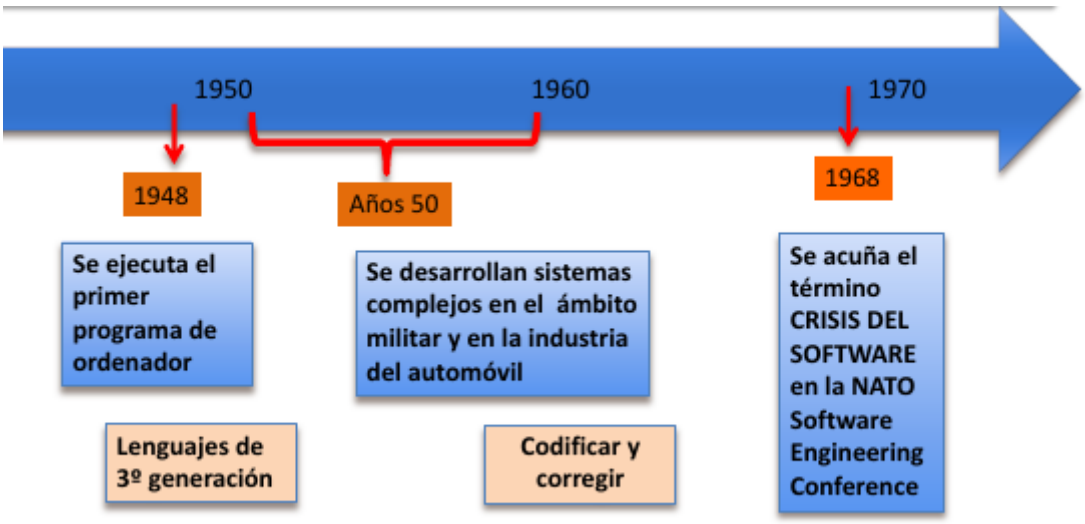
El software es el programa en la computadora, pero también es su código, los datos, o la documentación.

Problemas del software

- Mal funcionamiento (Calidad).
- Hay que mantener el volumen de software existente.
- Mantenimiento.
- Hay una demanda creciente de nuevo software.
- Adaptación a las nuevas tecnologías.
- Incremento de la complejidad.
- Requisitos incorrectos.
- ¿Cómo desarrollamos software?

Tendemos a construir los que no necesitamos - funcionalidades que como desarrolladores parecen buena idea acaban olvidadas por múltiples motivos: quizá no sean lo que el cliente quería, o son demasiado complicadas, o no encajan bien en el flujo de trabajo del programa.

2. Historia del desarrollo del software



La crisis del software

> La **Crisis del software** se refiere a los problemas que, desde sus inicios, ha ido experimentando el software, muchas veces problemas de gran magnitud, debido, principalmente, a la mínima eficacia que presentan una gran cantidad de empresas al momento de realizar un software. No fue hasta 1968 cuando en la primera conferencia elaborada por la OTAN, Friedrich L. Bauer habló por primera vez del conjunto de dificultades o errores ocurridos en la planificación, estimación de los costos, productividad y calidad de un software, o bien, lo que se conoce como la **crisis del software**. El término se le atribuyó a F. L. Bauer aunque ya había sido utilizado por Edsger Dijkstra en su libro *The Humble Programmer*. Para dar solución a los problemas que se presentaban en esta conferencia se creó una nueva rama de ingeniería, la **ingeniería de software**.
>
> Existen múltiples causas que originan la crisis del software. Una de ellas es que el desarrollo de un software es un proceso relativamente "nuevo", del cual no se tiene personal lo suficientemente capacitado, debido a una pobre implementación de los procesos más organizados. Por otro lado, debido a que el software es el conjunto de programas o instrucciones de una computadora, y que por lo tanto, no es un elemento de carácter físico; es poco probable que resulte exitoso el segundo primer intento de elaboración, ya que el personal encargado de su realización no posee total claridad de los requerimientos de su cliente, lo que a su vez, vuelve en exceso complicado hacer un diseño detallado de requerimientos, pues es importante mencionar que su calidad se mide con respecto a su funcionamiento.
>
> En muchos casos la automatización representa retribuciones económicas directas, pero para casos específicos puede ser difícil notar una diferencia ya que la retribución que se obtiene no es precisamente monetaria, sino que puede aumentar la velocidad de producción o volverla más flexible. En cualquier situación el desarrollo de software es una actividad capaz de añadir valores.

- El software es ineficiente.
- El software no satisface los requisitos.
- Los proyectos sobrepasan las estimaciones de tiempo y recursos.
- Los proyectos se vuelven inmanejables y el software imposible de mantener.

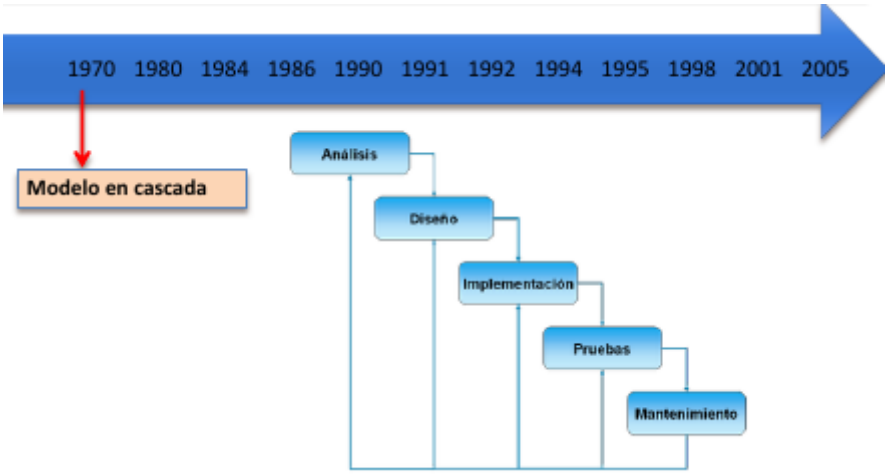
Soluciones?

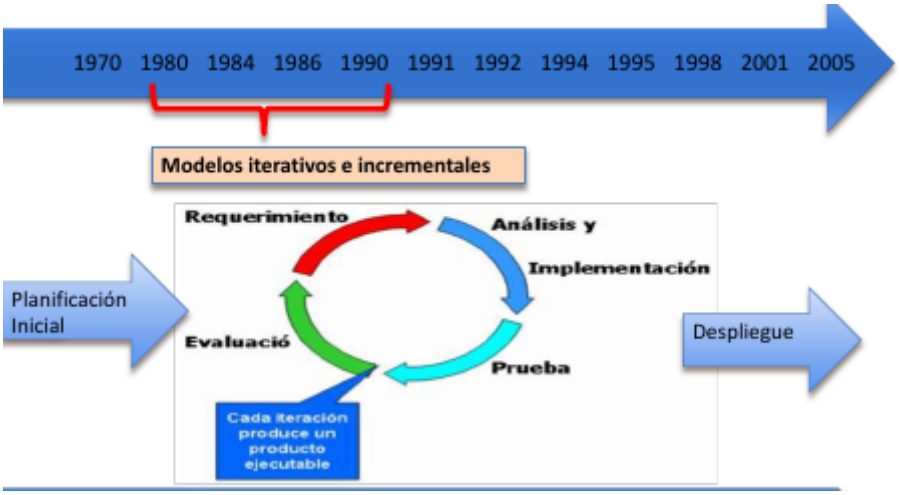
- **Gestión Predictiva de proyectos.**

    - Se considera que un proyecto de desarrollo de Software ha sido un éxito si se consigue la finalidad prevista, con el presupuesto y en la fechas que previamente se habían estimado.
    - **Gestión de proyectos "predictiva":**
        - Partir de la descripción detallada de lo que hay que hacer.
        - Realizar un plan detallado del proyecto (estimación).
        - Supervisar y coordinar la ejecución para evitar desviaciones del plan (seguimiento y medición).
- **Producción basada en procesos.**
    - **Proceso de desarrollo:** Define el marco de trabajo para la construcción racional del software (métodos, heurísticas, principios metodológicos y herramientas).

    - Los modelos de ciclos de vida del software dividen dicho desarrollo en unas etapas.
    - Distintos modelos: Etapas consideradas, posición relativa de las mismas y las tareas a realizar en cada una.
        - Secuenciales: Hasta que no se acaba totalmente una etapa no comienza la siguiente.
        - No secuenciales: El desarrollo se considera un proceso evolutivo en el que se parte de un software con funcionalidad mínima y se desarrolla progresivamente.
    - **Etapas principales:**
        - **Planificación**: Estimar el tiempo y los costes de desarrollo del software.
        - **Especificación de requisitos**: Análisis del problema a resolver. Documento en el que se dice qué debe hacer el sistema software.
        - **Diseño**: Búsqueda de la solución. Descripción de los componentes, sus relaciones y funciones que dan solución al problema.
        - **Implementación**: Traducción del diseño a un lenguaje de programación entendible por una máquina.
        - **Control de la calidad**: Debe realizarse durante todo el proceso de desarrollo. Revisiones de todo lo que se va obteniendo junto con la prueba del código.
        - **Mantenimiento o evolución**: Reparar fallos en el sistema cuando sean descubiertos o adaptar el software a los nuevos entornos.
- **Ingeniería del Software:**
    - Se necesita establecer y usar principios de ingeniería orientados a obtener software de manera económica, que sea fiable y funcione eficientemente sobre máquinas reales.
    - > La aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento del software; es decir, la aplicación de ingeniería al software.
      >
      > *--IEEE Standard Glossary of Software Engineering Terminology*
    - Proceso de aseguramiento de la calidad del proceso y del producto.

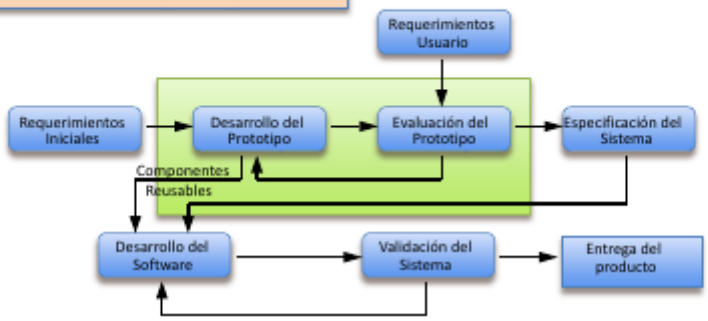

1970
1980
1990
2000
2010
Gestión Predictiva de Proyectos
• PMI (Proyect Management Institute)
Producción basada en procesos
• ISO (International Organization for Standardization)
• IEEE Computer Society
Ingeniería del Software
• SEI (Software Engineering Institute)


1970 1980 1984 1986 1990 1991 1992 1994 1995 1998 2001 2005
Modelo en cascada
Análisis
Diseño
Implementación
Pruebas
Mantenimiento


1970 1980 1984 1986 1990 1991 1992 1994 1995 1998 2001 2005
Modelos iterativos e incrementales
Requerimiento
Análisis y
Implementación
Planificación Inicial
Evaluació
Prueba
Despliegue
Cada iteración produce un producto ejecutable

1970 1980 1984 1986 1990 1991 1992 1994 1995 1998 2001 2005

Modelo basado en prototipos

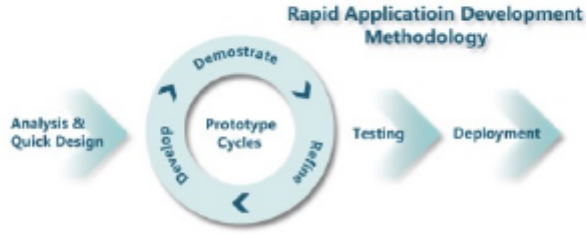


1970 1980 1984 1986 1990 1991 1992 1994 1995 1998 2001 2005

Modelo en espiral de Boehm

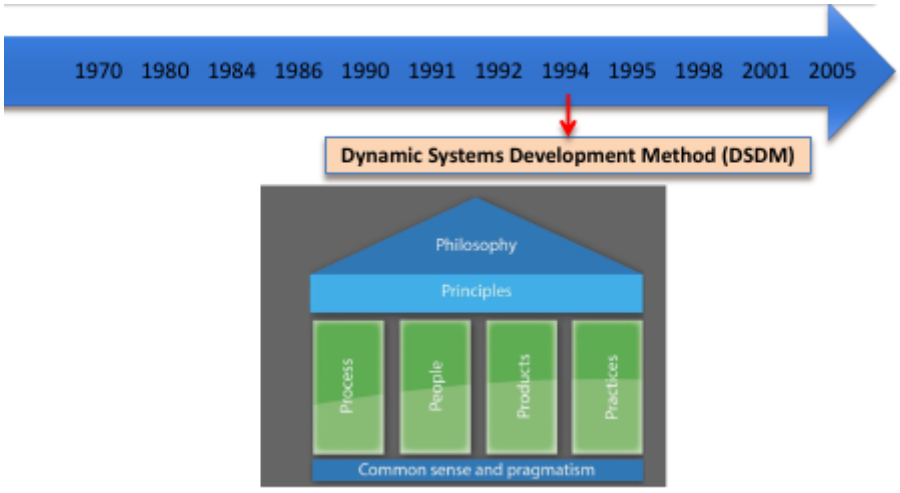
1970 1980 1984 1986 1990 1991 1992 1994 1995 1998 2001 2005

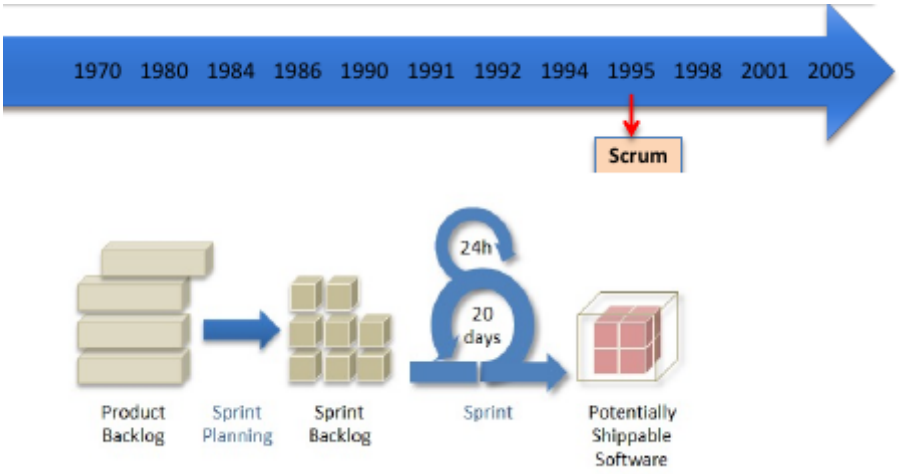
Desarrollo rápido de aplicaciones (RAD)

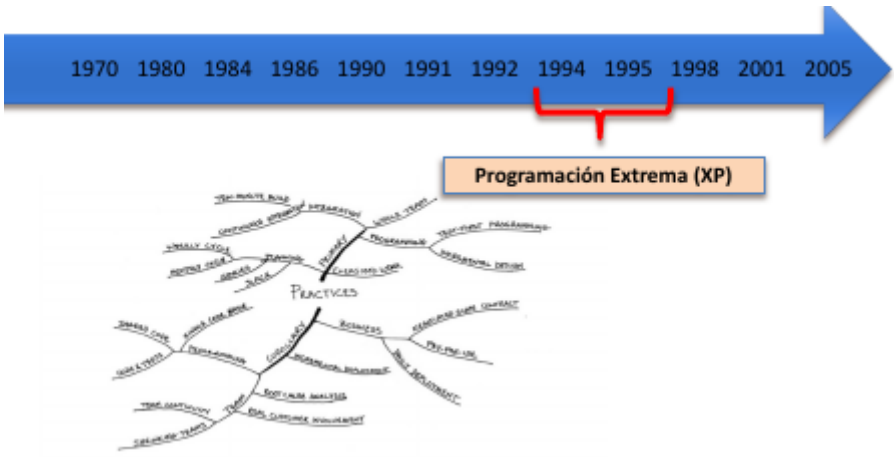


1970 1980 1984 1986 1990 1991 1992 1994 1995 1998 2001 2005

Crystal

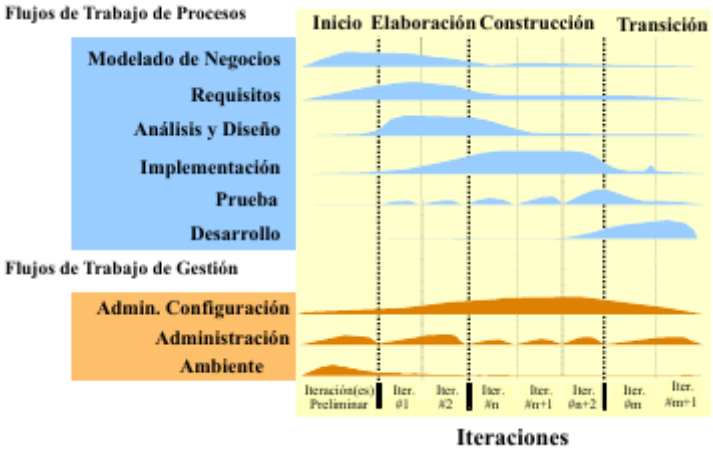




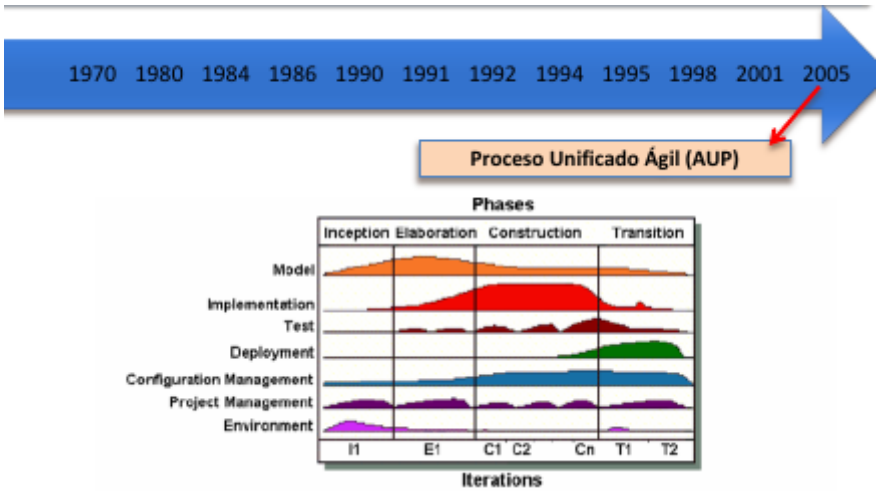


Proceso unificado RUP

- Publicado en 1999 por Ivar Jacobson, Grady Booch y James Rumbaugh.
- Marco de desarrollo de software adaptable caracterizado por:
    - Iterativo e incremental.
    - Dirigido por los casos de uso.
    - Centrado en la arquitectura.
    - Orientado a los riesgos.
    - Uso intensivo de UML.
- Iteraciones y fases del PU:



Principios del Manifiesto Ágil



3. Calidad en el software

Objetivo: desarrollar software de calidad, mejorando los procesos de desarrollo usados y el resultado final. Un software que no funciona...

- Solo cumple parcialmente lo que necesita el usuario, con errores y omisiones.
- Necesita un fuerte esfuerzo por parte del usuario para la adaptación al software.
- Su uso no satisface a sus usuarios.
- Es lento y poco eficiente.
- Es poco modificable, poco entendible y con problemas técnicos.

ISO 25000

Calidad del producto:

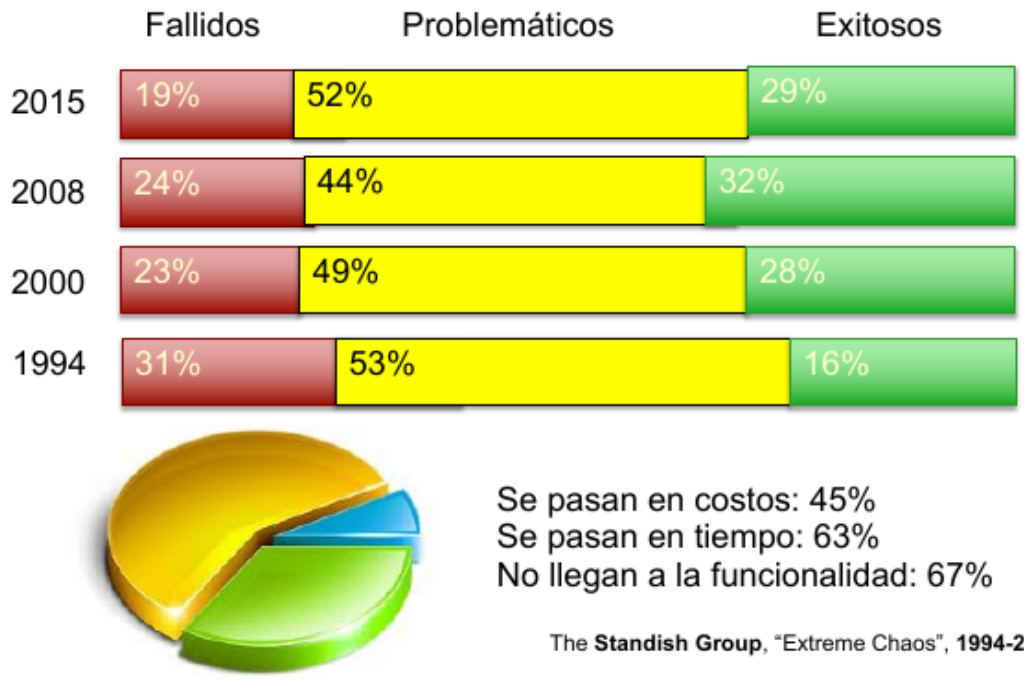
- Funcionalidad
  - Integridad funcional
  - Corrección funcional
  - Adecuación funcional
- Rendimiento
  - Comportamiento temporal
  - Uso de recursos
  - Capacidad
- Compatibilidad
  - Coexistencia
  - Interoperabilidad
- Usabilidad
  - Reconocimiento de la adecuación
  - Capacidad de aprendizaje
  - Operatividad
  - Protección de errores
  - Estética de la interfaz
  - Accesibilidad
- Fiabilidad

    - Madurez
    - Disponibilidad
    - Tolerancia a fallos
    - Capacidad de recuperación
- Seguridad
    - Confidencialidad
    - Integridad
    - No rechazo
    - Responsabilidades
    - Autenticidad
- Mantenibilidad
    - Modularidad
    - Reutilización
    - Analizabilidad
    - Testeabilidad
- Portabilidad
    - Adaptabilidad
    - Facilidad de instalación
    - Intercambiabilidad

Calidad en uso:

- Efectividad
- Productividad
- Seguridad
    - Riesgo de daño económico
    - Riesgo de salud
    - Riesgo ambiental
- Satisfacción
    - Cumplimiento del propósito
    - Confianza
    - Placer
    - Confort
- Contexto de uso
    - Flexibilidad
    - Cumplimiento de contextos de uso

El éxito es raro



	2011	2012	2013	2014	2015	2017
FALLIDOS	22 %	17 %	19 %	17 %	19 %	19%
PROBLEMÁTICOS	49 %	56 %	50 %	55 %	52 %	48%
EXITOSOS	29 %	27 %	31 %	28 %	29 %	33%

Causas del fracaso de un proyecto:

- Problemas en el alcance.
- Comunicación pobre.
- Falta de implicación de los implicados (stakeholders).
- Apoyo inadecuado del patrocinador del proyecto.

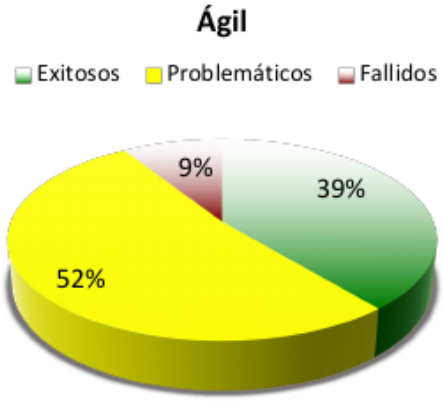
La inadecuada gestión de los requisitos es la culpable del fracaso de la mitad de los proyectos. (Según el 47% de los encuestados)

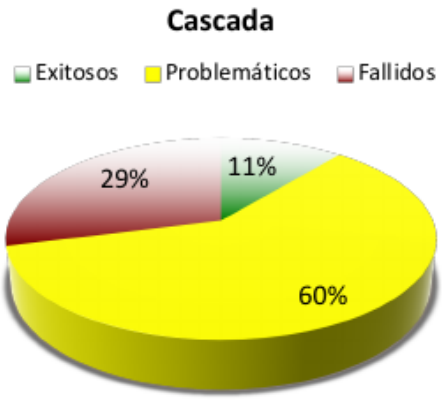
Puntos a mejorar:

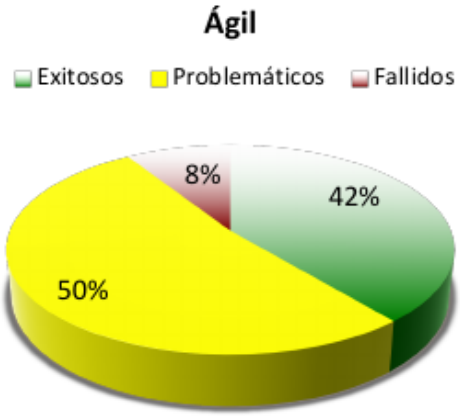
- Personas: Asignar los recursos necesarios y suficientes, así como reconocer y desarrollar las habilidades necesarias para desarrollar sus funciones.
- Procesos: Estandarizar y formalizar los procesos.
- Cultura: Crear un sentimiento en todos los niveles de la utilización de las prácticas elegidas.

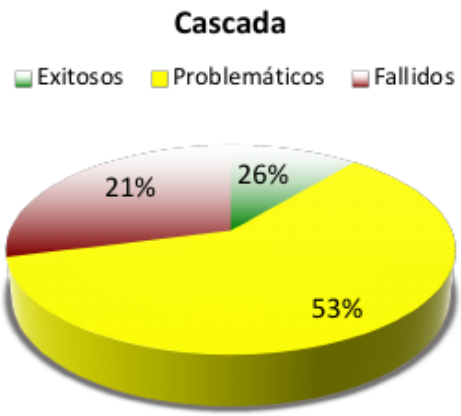
El tamaño del proyecto también afecta...

...así como la metodología de desarrollo. Comparativas entre 2011-2015 y 2013-2017

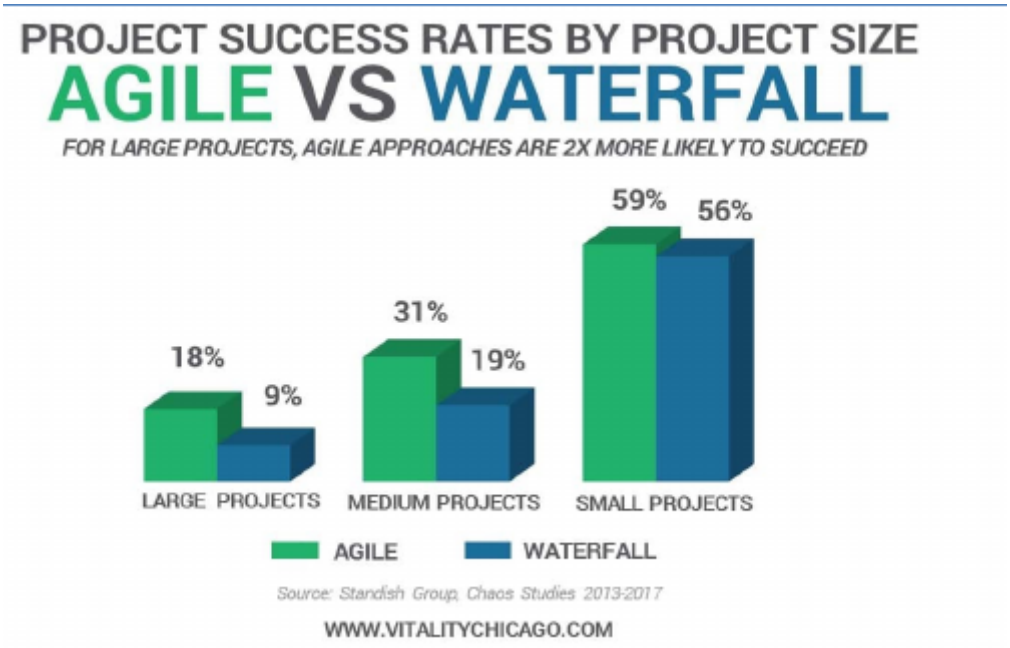


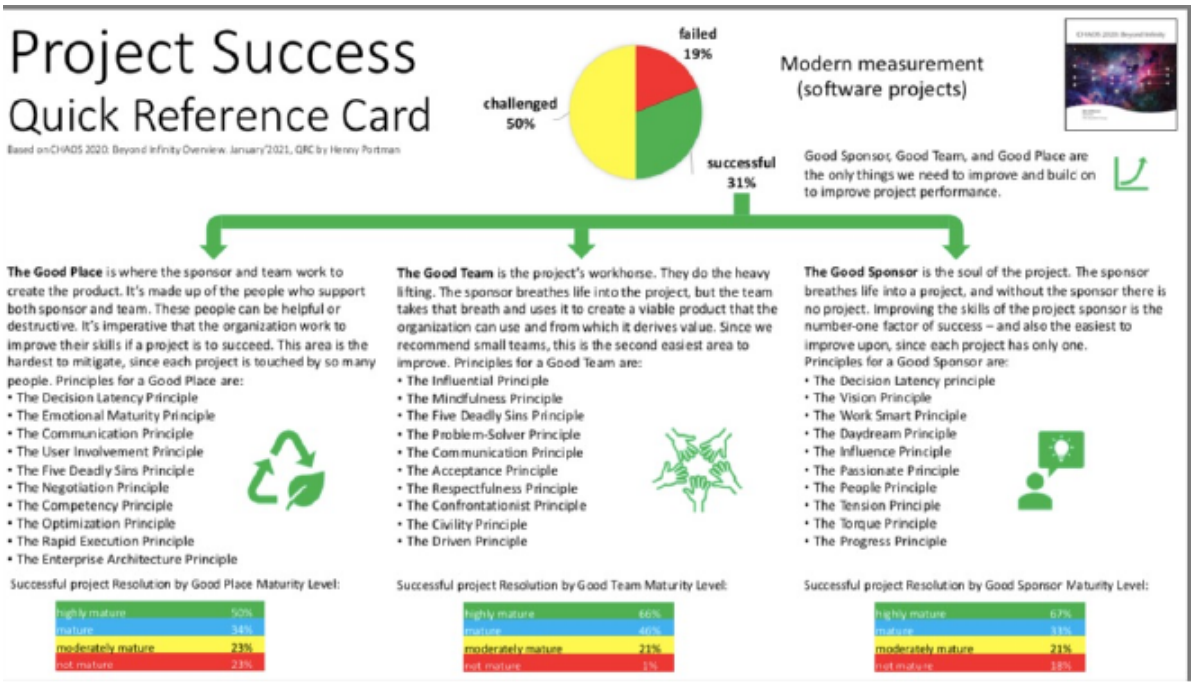






Relacionando tamaño de proyecto y metodología: en proyectos grandes, es dos veces más probables que las metodologías ágiles tengan éxito.





4. Problemas del desarrollo del software

Metodologías “predictivas”:

- Requisitos previos al desarrollo.
- ¿Qué desea el cliente? >> Contrato.
- Hay que evitar cambios en los requisitos.
- Los cambios son responsabilidad del cliente.
- Planificamos alcance + coste + tiempos.
- Gestionamos el proyecto.

No son ideales, porque...

- Detección de errores en etapas finales del proyecto.
- El cliente toma decisiones importantes en fases iniciales del proyecto.
- Proyectos difíciles de gestionar.
- Reducción de calidad.
- Comunicación pobre.

- Insatisfacción del cliente, proyectos inflexibles.
- Dificultad para implantar metodologías de desarrollo.

Entorno de desarrollo actual

- Requisitos cambiantes.
- Tiempo de desarrollo corto.
- Vida de las aplicaciones corta.
- Clientes y usuarios más exigentes.
- Medidas de la calidad y la productividad.

Es necesario desarrollar y construir el producto a la vez que se investiga y descubren los requisitos y hacerlo con una capacidad de adaptarse a los cambios dictados por el entorno (gestión de proyectos adaptable).

El cliente conoce la "visión" de su producto, pero por la novedad, el valor de la innovación, la velocidad de la tecnología y del negocio no puede detallar como será el producto final.

Tema 2: Principios y prácticas ágiles

Los métodos tradicionales (predictivos) no son buenos para entornos volátiles en los que los requisitos no son conocidos con exactitud (Incertidumbre y cambio).

- El cliente tiene que tomar muchas decisiones al principio (Costosas de cambiar y que no entiende).
- Los clientes piden más flexibilidad como solución a tomar decisiones pronto (Aumento de la complejidad del producto).
- El cliente toma decisiones con la información de las documentaciones técnicas.

¿Qué es más importante: excelencia técnica o satisfacción del cliente? ¿Acabar a tiempo y con el presupuesto adecuado, o maximizar el valor aportado por el cliente?

El cambio es inevitable

Cambios durante el desarrollo:

- Incertidumbre (en requisitos, en tecnología, etc.).
- Modificaciones al ver las primeras soluciones.

Cambios tras entregar el producto:

- Mantenimiento.
- Modificación y ampliaciones por cambios en el negocio.

Desarrollo Ideal:

- Los clientes saben lo que quieren y lo que necesitan.
- El equipo sabe cómo construirlo.
- Nada cambiará en el desarrollo.
- No hay restricciones (Tiempo y dinero para hacerlo).

Desarrollo Real:

- Los clientes descubren lo que necesitan.
- Los desarrolladores descubren cómo hacerlo.
- Muchas cosas cambian en el camino.
- Siempre hay más cosas que hacer que tiempo y dinero disponible.

1. El manifiesto ágil

> ¿Por qué tantos proyectos de desarrollo de software no se terminan a tiempo, cuestan más de lo presupuestado originalmente, tienen problemas de calidad serios y generan menor valor al negocio que el esperado? Se preguntaron 17 expertos, que después elaboraron el Manifiesto Ágil

Principales intenciones del manifiesto Ágil:

- El movimiento Ágil no es ir en contra de las metodologías (no es un “anti-método”).
- El modelado y la documentación es importante pero no para generar recursos inútiles o poco usados.
- Es importante planificar y estimar pero hay que reconocer los problemas y las necesidades de gestión en “entornos turbulentos”.

Principios del Manifiesto Ágil

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.
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.

¿Qué es la agilidad?

Según un diccionario de lengua española:

> "Habilidad de cambiar rápida y efectivamente la dirección de un movimiento ejecutado a velocidad".

> "Conjunto de mejores prácticas que permite la entrega rápida de software de alta calidad y un enfoque de negocio que alinea el desarrollo con las necesidades y los objetivos de las compañías"

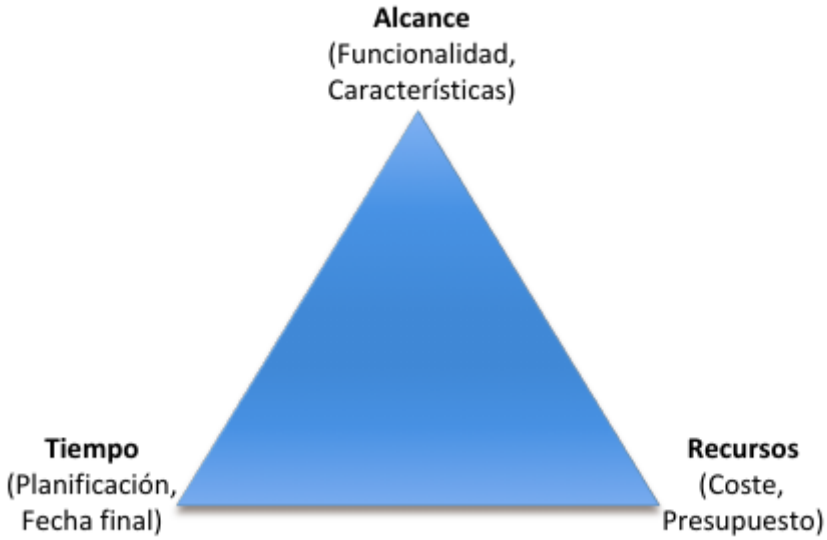
La insistencia en el cambio es el conductor primordial hacia la agilidad.

- Un equipo ágil es un equipo rápido que responde de manera apropiada a los cambios.
- Cambios en el software que se va a construir, cambios entre los miembros del equipo, cambios por las nuevas tecnologías, cambios de todo tipo que puedan influir en el producto o en el proyecto que crea el producto.

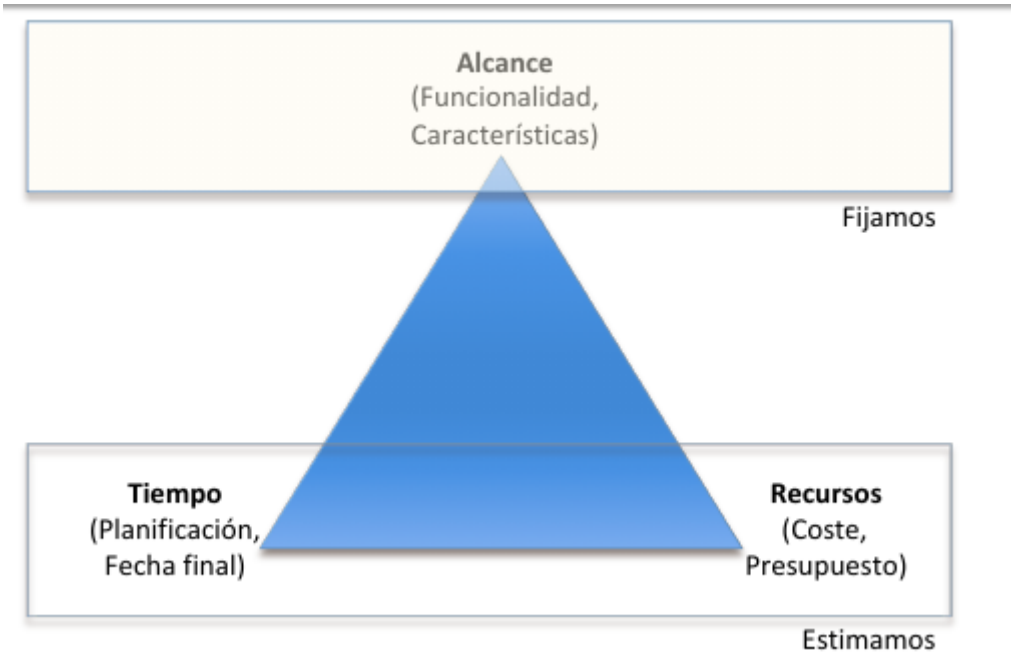
2. Gestión de un proyecto

¿Qué necesitamos para implementar la agilidad? Nuestro activo más valioso son las personas

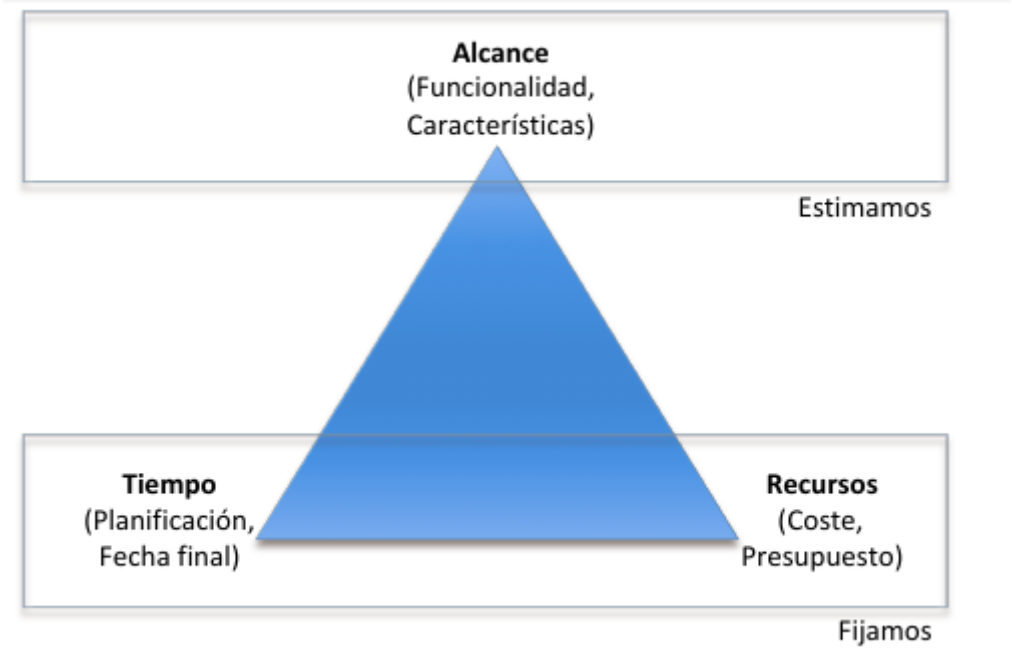
- Trabajo en equipo
- Priorizar por valor
- Aprendizaje
- Auto-organización
- Flexibilidad
- Excelencia técnica
- Comunicación cara a cara



Cuando seguimos un plan predictivo, fijamos el alcance y estimamos el tiempo y recursos:



Pero si nos dirigimos por el valor, dado unos recursos y un tiempo, estimamos unos resultados



El contrato con el cliente

Para cumplir estas estimaciones y demandas, es importante determinar qué tipo de contrato con el cliente vamos a tener.

El contrato es un acuerdo que establecemos con el cliente sobre el servicio que vamos a proporcionar (ganar-ganar). Un buen contrato aumenta las probabilidades de éxito.

- Los riesgos deberían ser compartidos.
- “Colaboración con el cliente frente a negociación contractual”.
- Algunos tipos…
    - Fijar todo.
    - Fijar algunas cosas + Colaboración.
    - Fijamos cosas de forma progresiva.
    - Contratamos para la visión + Otro contrato.
    - Cualquier funcionalidad puede cambiarse por otra de igual peso (…yo quito tu eliges).

- El cliente puede cancelar el proyecto si considera que no necesita más (pagando el 20% de lo cancelado).

El contrato es importante para evitar situaciones en las que es la palabra del cliente contra la mía.

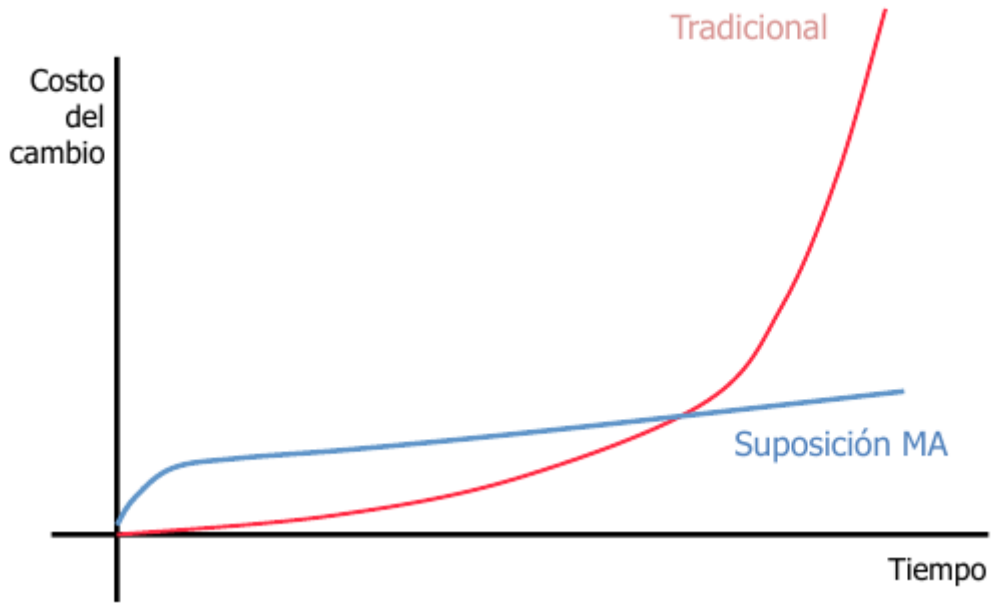
He said...	**... she said**
El sistema no funciona	El cliente cambió de idea
No podemos usarlo	Los usuarios no están formados
Nos deberían haber avisado	No nos hubieran escuchado
El sistema está lleno de bugs	Los datos están corruptos
Mal asesoramiento	Malas decisiones
Desarrolladores no cualificados	Interlocutores inadecuados
Mal proceso de desarrollo	Cliente no involucrado
El sistema no funcionó en el campo	El cliente no adaptó sus procesos
Entregaron tarde	Añadieron cambios

Seguimiento del proyecto

El seguimiento del proyecto debería estar basado en objetivos

- Los requisitos se desarrollaran priorizando por el valor aportado al cliente.
- El control y seguimiento del proyecto se basará en los requisitos completados.

El costo de los cambios en las metodologías ágiles se mantiene casi constante sin importar la etapa de desarrollo en la que estemos, mientras que en las tradicionales se dispara porque hay que volver atrás y rehacer el trabajo.



Valor real del producto

Dedicamos mucho tiempo a desarrollar objetivos que aportan poco valor. ¿Quién conoce el valor real de una funcionalidad?

De las funcionalidades desarrolladas:

- 7% se usan siempre.
- 13% se usan a menudo.
- 19% se usan pocas veces.
- 45% no se usan nunca.

En las metodologías ágiles es primordial la economización del trabajo, maximizar la cantidad de trabajo no hecho. Por eso es importante identificar las funciones que realmente aportan valor.

3. Equipos ágiles

- Competencia. Habilidades específicas sobre el software y el proceso a seguir.
- Enfoque común. Una sola meta. “Entregar al cliente un incremento de valor del producto software”.
- Colaboración. Comunicación y Trabajo en equipo.
- Toma de decisiones. Autonomía y Autoridad.
- Resolución de problemas confusos. Adaptación.
- Confianza y respeto.
- Auto organización.

¿Cuándo aplicar agilidad?

- Cuando tenemos un equipo pequeño y localizado en el mismo sitio.
- Cuando los requisitos son dinámicos.
- No tenemos un sistema crítico.
- Los clientes tienen disponibilidad para el proyecto y quieren colaborar.
- La empresa está abierta a nuevas formas de desarrollo de software.

Seminario 1: Las personas y los escenarios

1. Personas

- ¿Cómo son los grupos de usuarios?
- ¿Cómo se comportan?
- ¿Qué piensan?
- ¿Qué quieren lograr?
- ¿Por qué quieren alcanzar eso?

En 1999, Alan Cooper en su libro “The inmates are running the asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity” define el método Personas como una herramienta práctica de diseño de interacción. En su libro “About Face: the Essencials of Interaction Design” propone un método para crear personas.

- Cooper mantiene que el desarrollador tiene un gran control sobre el diseño de nuevas tecnologías y negocios, pero normalmente no se comunica con la gente para la que diseña
- Cooper usa las personas como solución a este problema, como forma de hacer que los equipos de desarrollo empatizaran con los usuarios

Una persona es una representación ficitica pero concreta del grupo de usuarios al que va dirigido un producto. Las personas son arquetipos de usuarios.

Consiste en identificar y conocer a los usuarios, personalizarlos y empatizar con ellos. Objetivos:

- Realizar una compresión clara de los objetivos y necesidades de los usuarios en contextos específicos de uso.
- Instrumento crítico para la ideación y validación temprana de soluciones o conceptos de diseño.

Guía de uso

- Las personas deben estar basadas en observaciones del mundo real.
- Es mejor centrarnos en las metas de los usuarios que en las tareas que realizan.
- No son personas reales, pero representan a usuarios reales durante el diseño y son descritos y usados con rigor.
- Evitan que el diseñador proyecte sobre el sistema sus propios objetivos e ideas.
- Usar un número manejables de personas. No describen el sistema completo.

Tipos de personas

- Principales: representan usuarios con necesidades específicas que sólo pueden ser satisfechas con una interfaz de usuario diseñada específicamente para ellos.
- Secundarias: son personas que también utilizan el sistema pero que pueden utilizar la interfaz diseñada para una persona principal.

> **Principio de diseño de Cooper: Cada persona principal debe tener una interface propia.**

Plantilla

No existe una única plantilla, pero deben incluir:

- Aspectos demográficos
- Aspectos relacionados con el producto a desarrollar (necesidades y objetivos)

PLANTILLA DE PERSONAJE		
Nombre		Foto
Edad		
Sexo		
Educación		
Contexto de uso		
Cuándo	¿Cuándo utiliza el ordenador?	
Dónde	¿Dónde?	
Tipo de ordenador	¿Qué tipo de ordenador utiliza?	
Misión		
Objetivo	¿Para qué quiere utilizar nuestra aplicación?	
Expectativas	¿Qué espera encontrar en ella?	
Motivación		
Urgencia	¿Para cuando quiere utilizarla?	
Deseo	¿Por qué quiere alcanzar ese objetivo?	
Actitud hacia la tecnología		
Tímido, agresivo, precavido		

Demographic characteristics. Age, gender, location, socio-economic status

Occupation experience. Current job title, years at the company, years of experience at that position, responsibilities, previous jobs and job titles

Company information. Company size, industry

Education. Degree, major, courses taken

Computer experience. Computer skills, years of experience

Specific product experience. Experience with competitors' products or other domain-specific products, usage trends

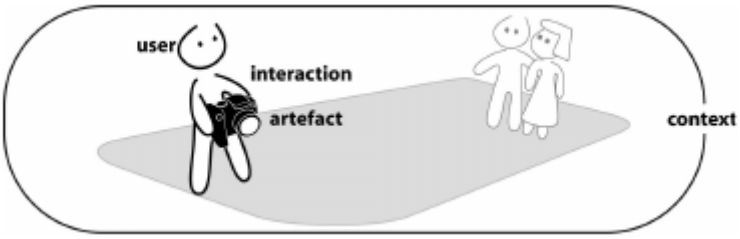
- **Tasks**. Primary tasks, secondary tasks
- **Domain knowledge**. The users' understanding of the product area
- **Technology available**. Computer hardware (monitor size, computing speed, etc.), software, other tools typically used
- **Attitudes and values**. Product preferences, fear of technology, etc.
- **Learning style**. Visual learner, audio learner, etc.
- **Criticality of errors**. In general, the possible consequences of a user's error.

2. Escenarios

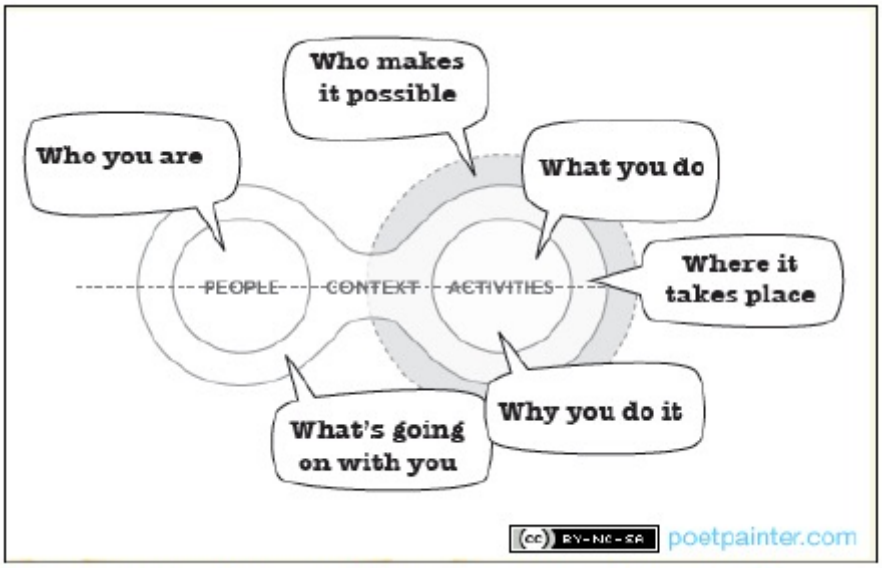
Diseño basado en escenarios. Cambiamos el objetivo funcional de la especificación, por describir como la gente usará un sistema para lograr realizar un conjunto de tareas o actividades determinadas. [Carroll-2002]

No se usan para describir las posibles iteraciones concretas de un usuario, sino que imaginamos/diseñamos/proponemos como podría ser la forma de interactuar con el futuro sistema y detectamos el contexto de uso y los objetivos de los usuarios.

Un Escenario es un descripción de una persona usando un producto para alcanzar un objetivo. Describe una instancia de uso de un producto o sistema en un contexto determinado. (Una o más tareas realizadas en una situación determinada). Se representa usando una narrativa que cuenta una historia.



> "Los escenarios son historias - historias sobre personas y sus actividades" John Carroll

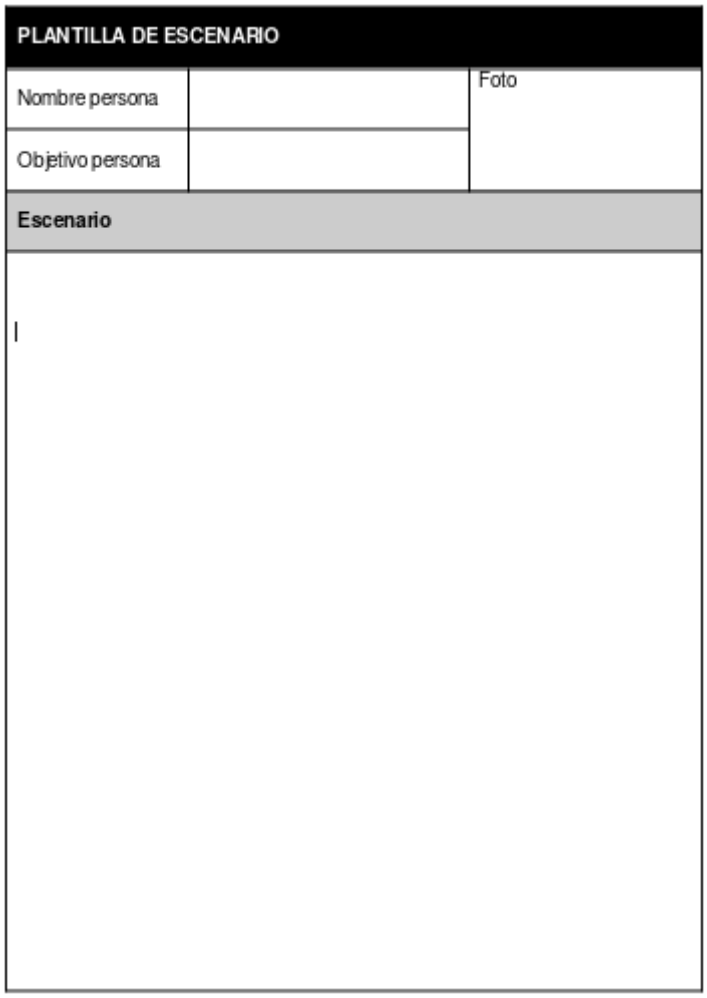


Información necesaria para describir un escenario

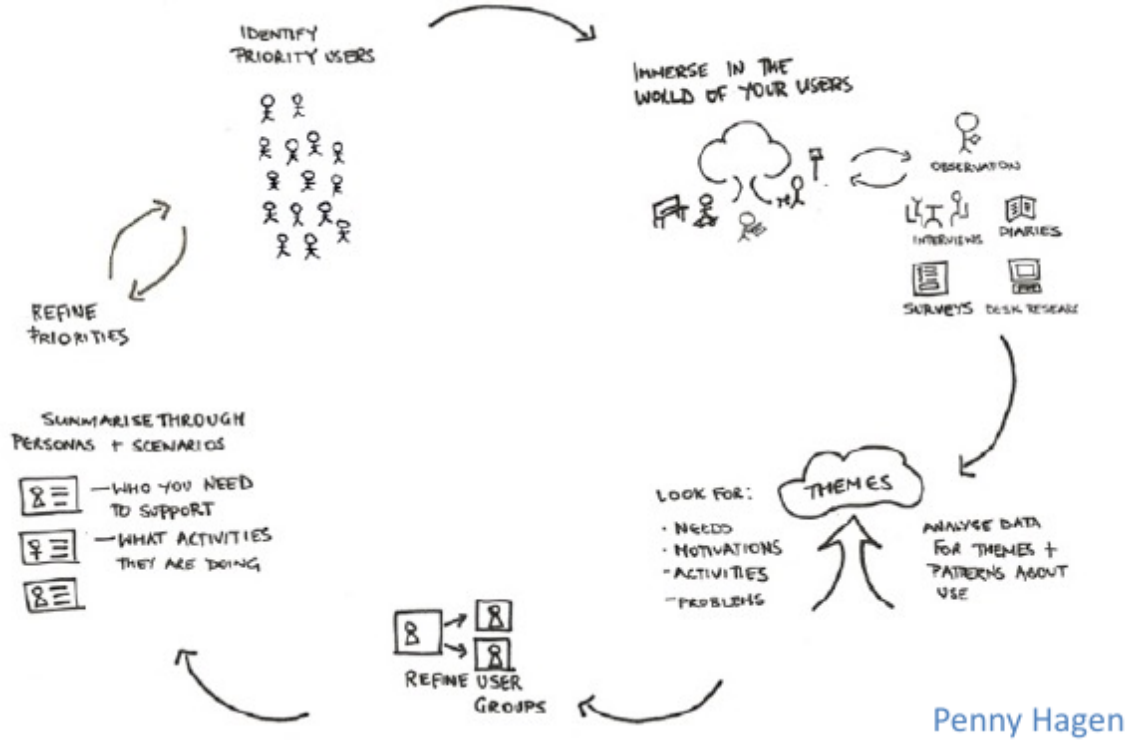
- Objetivos: ¿Qué quiere conseguir el usuario?, ¿Cómo afecta la acción del usuario a los objetivos de la organización?
- Procesos: ¿Qué pasos tiene que realizar el usuario?, ¿Qué información usa y como fluye entre los procesos?, ¿Con qué roles actúa el usuario en el proceso?
- Entradas & Salidas: ¿Qué información o material necesita el usuario para hacer de interface con el sistema?, ¿Qué necesita el usuario del sistema para continuar con sus objetivos?
- Experiencia: ¿Qué cosas similares ha realizado el usuario en el pasado?, ¿Cómo se podía alcanzar este objetivo en el pasado dentro de la organización?
- Restricciones: ¿Qué restricciones físicas , temporales o de negocio se tienen que imponer el usuario?
- Entorno físico: ¿Qué necesita el usuario para hacer las tareas asociadas al objetivo?, ¿Dónde va a realizar la actividad?
- Herramientas a usar: ¿Qué software y hardware usa actualmente el usuario?
- Relaciones: ¿Existe alguna relación entre el usuario que realiza las tareas y otros personas que se vean afectadas por las tareas?

Algunas guías de estilo al redactar escenarios

- Usar un lenguaje sencillo para describir las interacciones que son necesarias en el escenario.
- Evitar las referencias a la tecnología que no sean necesarias para el escenario.
    - [Por ejemplo: Es mejor decir el usuario se identifica, que el usuario introduce el PIN de 4 dígitos. A no ser que no sea negociable lo del PIN]
- Incluir referencias a aspectos relevantes del escenario aunque sean externos, como son los culturales, los sociales o de formas de actuar.
    - [Por ejemplo: Es importante reflejar que Marta es constantemente interrumpida por llamadas telefónicas]



3. Aplicación de personas y escenarios



1. Analizar a los usuarios e identificar a los prioritarios (principales)
    1. ¿Qué estrategia debemos seguir para diseñar para un público diverso?

        No debemos caer en la trampa de buscar una funcionalidad los más amplia posible para poder alcanzar las necesidades de la mayor cantidad de usuarios. Lo que debemos buscar es diseñar para tipos de usuarios específicos con necesidades muy particulares.
2. Introducirse en el mundo de los usuarios
    1. Observación de usuarios.
    2. Entrevistas, Cuestionarios y Encuestas.
    3. Diarios de Uso.
    4. Análisis Etnográfico.
    5. Indagación en el contexto.

CONTEXTUAL INQUIRY			
LOCATION 1	PEOPLE 2	CULTURE 3	VALUES 4
• *Describe direct location* • *Inside / outside* • *Problems* • *Safety issues* • *Environment (temp, humidity, conditions)* • *Lighting* • *Speed of action* • *Period of time - usage, frequency of use* • *Fixed location?* • *Sitting or standing?* • *etc*	• *Who is nearby?* • *Who else uses the product?* • *Who has an important relationship with the user?* • *Who is the boss?* • *Who are the subordinates?* • *Relatives?*	• *Describe the prevailing culture?* • *What are the beliefs of the users?* • *What are the working methods?* • *What are the customs?* • *What styles of communication prevail?* • *What are the decision making processes?* • *Why does this culture prevail?*	• *What values are important to the customers?* • *What does the customer like, dislike, hate, love, tolerate, desire etc?* • *What represents success and failure?*

3. Buscar “Patrones de comportamiento”.

1. Necesidades.
2. Actividades.
3. Motivaciones.
4. Problemas.

Nos enfocamos en analizar:

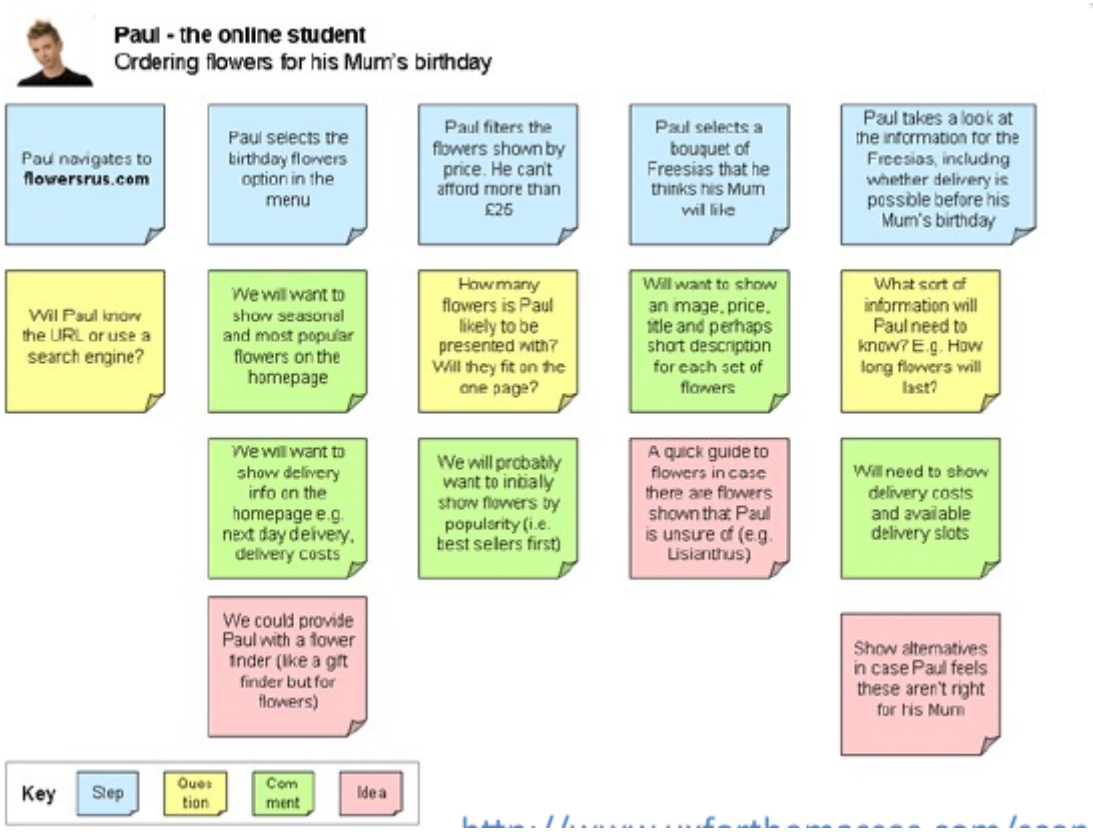
- Actvidades. Lo que el usuario hace; con qué frecuencia y en qué canadad.
- Actitudes. Qué piensa el usuario acerca del uso del producto y la tecnología.
- Aptitudes. Qué educación y entrenamiento tiene el usuario; cual es su habilidad para aprender.
- Motivaciones. Por qué el usuario está involucrado en el uso del producto.
- Habilidades. Habilidades del usuario relacionadas con el uso del producto y la tecnología.

Tipos de Escenarios:

- Escenarios actuales.
- Escenarios futuros.
- Escenarios de prueba.

  Todos los escenarios tienen que tener un objetivo determinado.

4. Redefinir los grupos de usuarios. Representante de un Rol.
5. Descripción de personas y escenarios de uso. Plantillas.



4. ¿Por qué usar personas y escenarios?

- Sirven para comunicar y explorar soluciones de diseño.
- Son concretos pero suficientemente flexibles para acomodar cambios y detalles según evoluciona el proyecto.
- Sirven para comprender el flujo de la experiencia y analizar el diseño propuesto.
- Ayudan para presentar y situar la solución propuesta.

- Ayudan a identificar problemas potenciales.
- Fáciles de entender por todos (stakeholders) ya que son como una “historia”.
- Incluyen información sobre el “contexto” (descripción de uso en una situación concreta).
- Ayudan a analizar si la solución es la apropiada.
- Ayudan a detectar factores sociales.

¿Cuándo podemos usarlos?

- Como herramienta para explorar posibles soluciones.
- Para averiguar cuestiones que han sido supuestas en el escenario.
- Durante el análisis de requisitos del software.
- Para validar soluciones y diseños y encontrar problemas junto con los usuarios.
- Como soporte a la documentación de especificaciones y diseños de las IU (wire-frames).
- Para generar las pruebas de los diseños finales (análisis de la usabilidad).
- Si hacemos un “análisis de competidores”, para comparar situaciones de uso entre diferentes sistemas.

Seminario 2: Historias de usuario

1. Historias de usuario

> "La historia es la unidad de funcionalidad en un proyecto de desarrollo. Demostramos el progreso entregando código testeado e integrado que implementa una historia. La historia debe ser entendible para los clientes, testeable por los desarrolladores, valiosa para el cliente y los suficientemente pequeña como para que se pueda desarrollar media docena en una iteración"
>
> -- Beck, K.; Fowler, M.: “Planning Extreme Programming”.

Una HU es una descripción simple de una tarea concreta que aporta valor al usuario o al negocio. Se pueden ver como una corta declaración de intención que describe algo que el sistema necesita hacer para el usuario. Suelen obtenerse de las reuniones con los “stakeholders”.

- Las historias tienen que ser simples, cortas, fáciles de entender, de aceptar y recordar, sin que tengan que escribirse todos sus detalles.
- Se describen usando tarjetas (descripción escrita en lenguaje del negocio que identifica y recuerda el requisito y ayuda a la priorización).

Objetivo de las HU

- El objetivo de las HU es principalmente, lograr la interacción con el equipo y con el usuario, por encima de documentar los requisitos que hay en el sistema.
- Hay que justificar el beneficio de todo lo que aparece en una HU.

¿Qué no es una HU?

No es un requisito del sistema (algo que el sistema deba hacer .....), hay que verlo más como algo negociable (necesitaríamos hacer algo similar a esto ....)

No son casos de uso

Concepto	Casos de uso	Historias de usuario

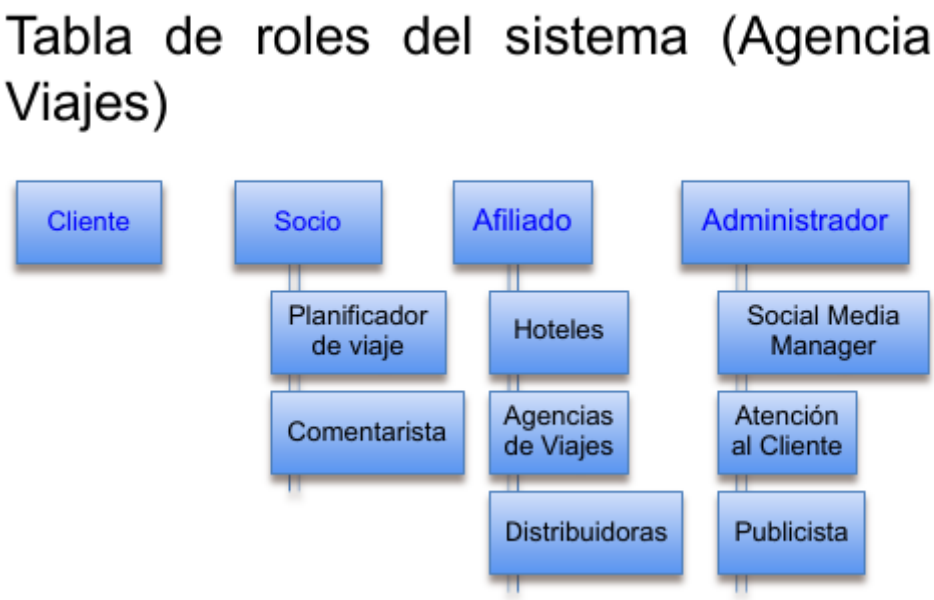
Concepto	Casos de uso	Historias de usuario
Objetivo	Modelar la interacción entre un actor y un sistema	Redactar una descripción breve de una funcionalidad tal y como la percibe el usuario
Estructura	Texto detallado donde se sigue una plantilla predefinida con conceptos técnicos (objetivo, resumen, actor, evento...)	Corta y consistente en una o dos frases escritas en lenguaje natural
Planificación	No se usan para planificar	Se usan para planificar
Agilidad	Requieren tiempo para análisis y la redacción de plantillas predefinidas	Se pueden escribir en pocos minutos
Comprensión	Difícil comprensión, incluso para los técnicos	Lenguaje natural
Mantenimiento	Pertenecen a documentos con cientos de páginas, difíciles de mantener	Muy fáciles de mantener
Comunicación	Modelo textual asociado a diagramas UML	Basadas en la comunicación verbal y orientada a la colaboración y discusión para clarificar detalles
Soporte	Escritos en documentos con el objetivo de que estos se archiven y actúen de referencia	Escritas en tarjetas (figurativas o reales) con el objetivo de ser usadas dinámicamente
Duración	Puede ser implementada en varias iteraciones	Debe ser implementada y probada en una sola iteración
Autores	Definidos por "intérpretes" (analistas, consultores...)	Posibilidad de que las definan usuarios y clientes
Pruebas	La definición de pruebas se redacta en documentación separada	Contienen prubas de aceptación en el reverso de la tarjeta
Contexto	Proporcionan una visión más general del sistema y la integración en él	Proporciona una visión menos obvia, por eso la pruebas y el feedback de los usuarios son tan importantes en la metodologías ágiles
Metodología	Asociado con el RUP	Asociado con la programación extrema

¿Quién escribe las HU?

- XP: El cliente.
- SCRUM: El Product Owner (con ayuda de clientes, stakeholders, equipo de desarrollo).
- En la realidad: Cualquier miembro del equipo de desarrollo con conocimiento suficiente en el dominio del problema.
- Los que pueden expresar las necesidades reales del sistema.
- Participar en las reuniones para detallarlas y priorizarlas.
    - (Customer Team) ---- (Stakeholders)
        - Clientes reales (usuarios).
        - Product Manager.
        - Probadores.
        - Diseñadores de interacción.

Hay que identificar y organizar a los usuarios del sistema. (Roles o personas)

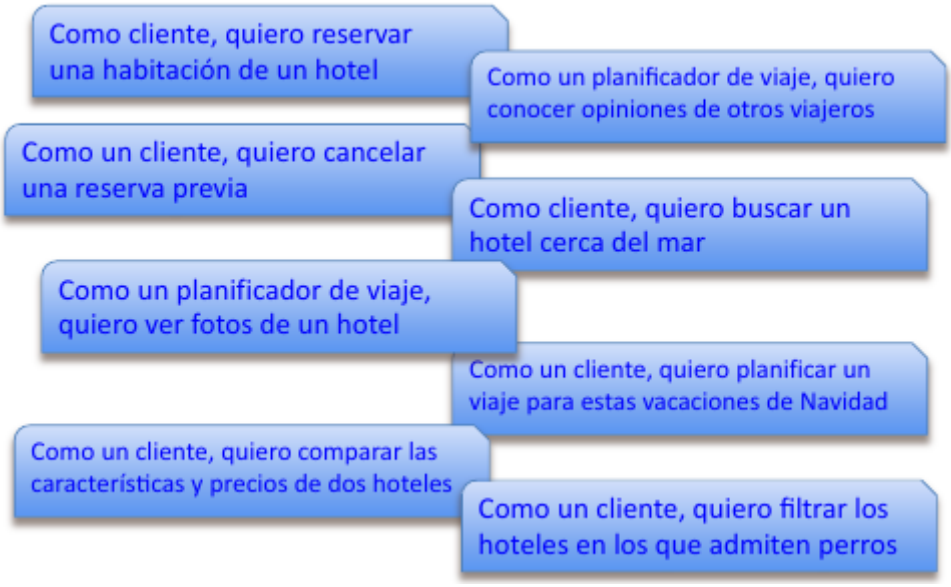
- Roles del sistema:
    - Podemos usar una sesión de “Brainstorming” para definirlos.
    - Definición de los roles. (Frecuencia de uso del software, objetivos que se plantean, forma habitual de acceso, etc.)



Formato de una HU

> Como [rol] quiero [característica, objetivo] para [valor de negocio, beneficio]

- [Cómo:] Quién es el usuario o grupo de usuarios que participan en la historia, normalmente es un “rol de usuario”, el que usará la funcionalidad” descrita por esta historia.
- [Quiero:] Es la actividad, el eje de la historia, qué hace el usuario en la historia.
- [Para:] Es el propósito de la historia, la meta que quiere alcanzar el usuario al ejecutar la historia.



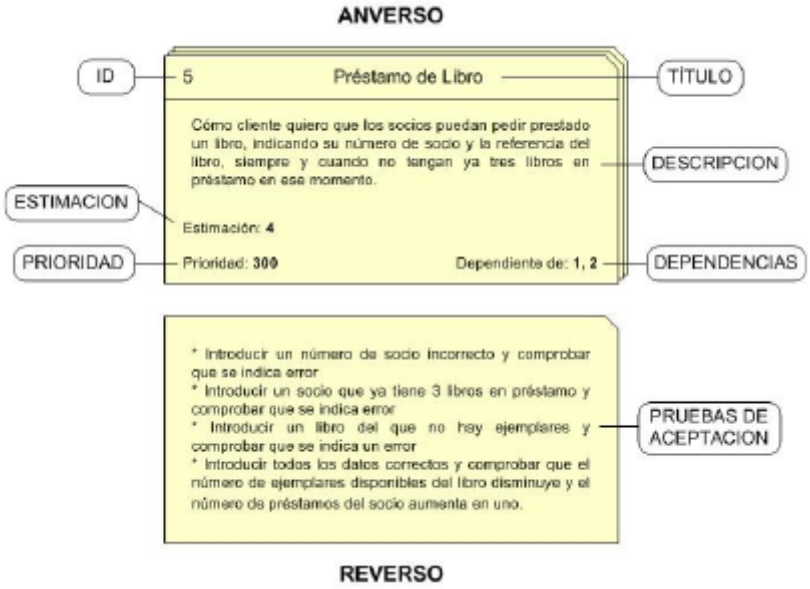
Elementos de una HU, tarjeta de HU

La información en una HU es una descripción corta que no incluye detalles. Los detalles se trabajan durante la etapa de "Conversación". Una tarjeta con demasiados detalles limita la conversación con el cliente.

- Tarjeta (Card): Representa las 2-3 sentencias usadas para describir la intención de la HU.
- Conversación: Detalles sobre la HU obtenidos de la conversación con el usuario o cliente.
- Confirmación: Cómo el equipo confirmará que la HU ha sido implementada (Test de aceptación).

¿Dónde están los detalles de las HU? Se describen en la conversaciones entre el equipo y los usuarios.

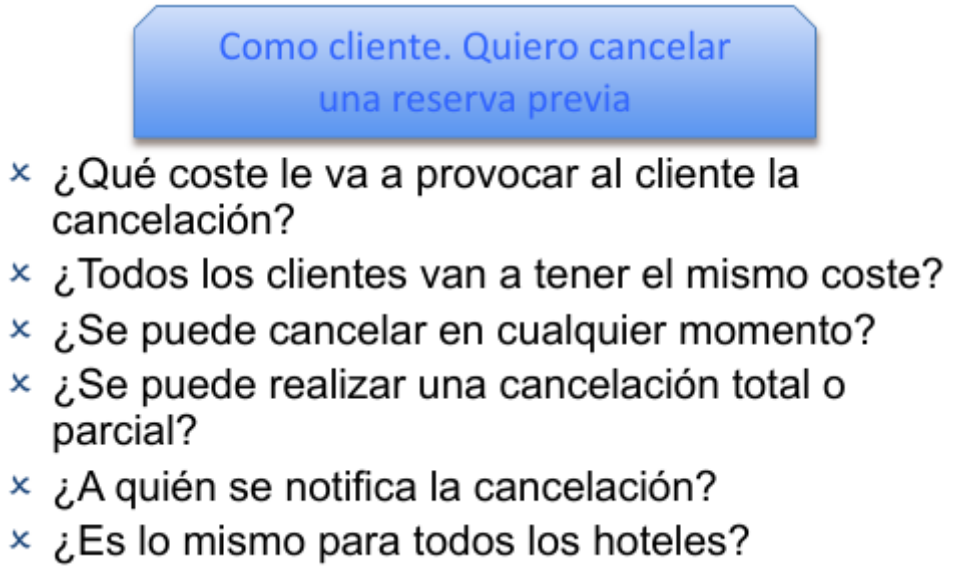
- Se delimita la parte a abordar por la HU usando las “pruebas de aceptación” [Confirmación]
    - Flujos alternativos en la actividad.
    - Límites de aceptación.
    - Pruebas de aceptación por parte de los clientes.



Historia de Usuario	
Número: 3	**Nombre:** Elaboración de Presupuesto
Usuario: Desarrollo Empresarial	
Modificación de Historia Nº: 0	**Iteración Asignada: 2**
Prioridad en Negocio: Media (Alta/Media/Baja)	**Puntos estimados:**
Desarrollador Encargado: Luis Ariel Castillo Estupiñan	
Descripción: Elaborar el presupuesto para la ejecución del evento. Depende de la disponibilidad presupuestal estipulada para la dependencia para el año vigente.	
Observaciones: El presupuesto está estipulado para la dependencia para el año, lo cual implica que para cada evento se tiene un máximo de recursos dependiendo de la naturaleza del mismo.	

Etapa de conversación

Buscar detalles en la HU



Etapa de confirmación

Se usan para expresar muchos de los detalles que se obtienen de la conversación entre clientes y desarrolladores. Se usan a dos niveles:

- Características que se añaden a la tarjeta de la HU.
- Pruebas que se usan para demostrar que la HU ha sido correctamente desarrollada.

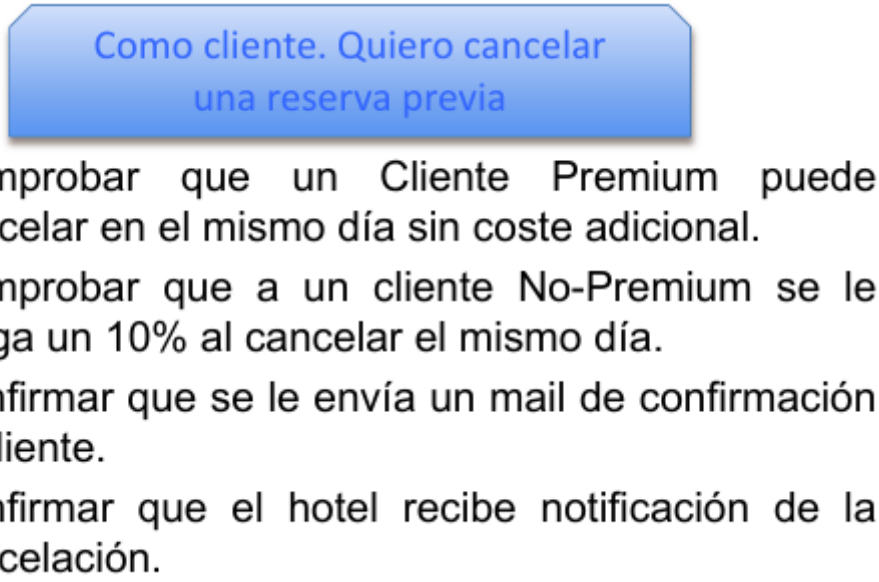
Los programadores pueden usarlos para mejorar el desarrollo ya que se describen antes de codificar nada. Las oruebas de aceptación:

- Se escriben en un lenguaje del ámbito de negocios.
- Aunque cualquiera puede escribir una prueba, el Product owner es el principal propietario de la prueba.
- Son pruebas de caja negra en el sentido de que sólo se verifican que las salidas del sistema cumplen las condiciones de satisfacción indicadas, sin preocuparse de cómo se consiguen los resultados.
- Se implementan en el transcurso de la iteración en la que se implementa la propia historia de usuario.
- Podemos añadir pruebas de aceptación mientras añadan valor y claridad a la HU.

- No hay que olvidar que el equipo de desarrollo realiza pruebas de unidad sobre los módulos que se desarrollen, independientemente de las pruebas de aceptación definidas.

Se podrían describir usando el formato (Behavior-Driven Development):

- **Dado** un sistema donde existe un usuario con nombre “idprueba”.
- **Cuando** alguien intenta registrarse con nombre “idprueba”.
- **Entonces** el sistema muestra un mensaje indicando que el nombre “idprueba” ya existe.



2. Características de las HU

- Son cortas y fáciles de leer, entendibles por los desarrolladores, interesados y usuarios.
- Representan incrementos pequeños de funcionalidad valorada, que puede ser desarrollada en un período de días o semanas.
- Son relativamente fáciles de estimar porque el esfuerzo de implementar la funcionalidad puede determinarse rápidamente.
- No se gestionan con documentos grandes o pesados, sino más bien en listas organizadas que pueden ordenarse más fácilmente y reordenarse a medida que se descubre nueva información.
- Necesitan poco o ningún mantenimiento y se pueden desechar con seguridad después de la implementación.
- Las historias de usuario, y el código que se crea a continuación, sirven como documentación, la cual también es elaborada de manera incremental.

HU terminada

La definición de “terminado” es un acuerdo de calidad entre los desarrolladores y los clientes. Se puede establecer una definición de “terminada” para todas las HU o un “terminado” específico para algunas HU.

Aspectos a considerar en la definición de “terminada”:

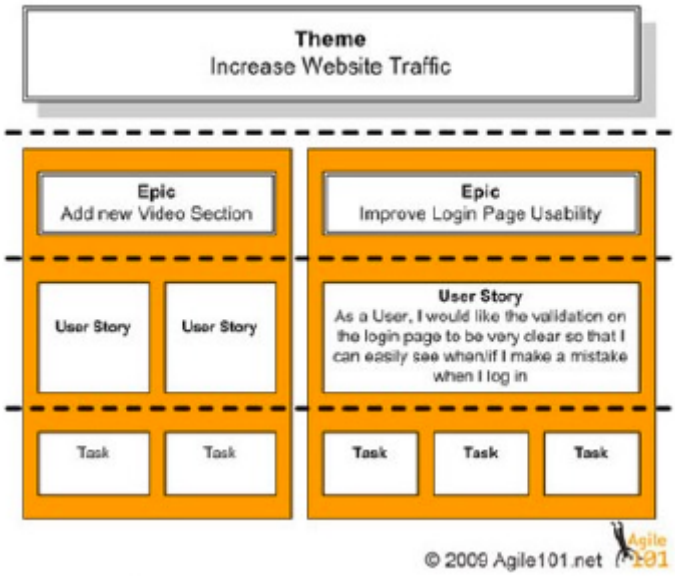
- Tipos de prueba y el grado de prueba.
- Nivel de calidad del producto software (métricas, etc.)
- Nivel de documentación.
- Entorno específico en que concluye el trabajo: en prepro, testing, producción.
- etc...

¿Porqué HU?

- Mantenemos una relación cercana con el cliente.
- Nos centramos en los aspectos relevantes actuales aplazando los detalles para cuando sean necesarios.
- Desarrollo oportunista (Requisitos emergentes).
- Facilita el trabajo iterativo.
- Buen tamaño para planificar y estimar.

Granularidad en HU

- Temas. Grandes proyectos, peticiones globales sin más análisis ni detalles. Subproductos.
    - *Planificador de viajes, gestor de ofertas de hoteles, sitio Web para socios.*
- Épicas. Súper historias de usuario, más concretas que los Temas. Conjunto de HU relacionadas entre sí.
    - *Sistema de búsqueda para hoteles, anotador de satisfacción de instalaciones y viajes, filtros aplicados a las búsquedas, comparador avanzado de ofertas.*
- Historias de Usuario. Una manera simple de describir una tarea concisa que aporta valor.
    - *Como cliente quiero buscar un hotel en una localidad determinada, como cliente quiero obtener un viaje una zona determinada y bajo un rango de precios, como socio quiero pagar la reserva de mi viaje usando una tarjeta de crédito.*
- Tareas. Las HU pueden ser divididas en diversas tareas por necesidades técnicas o de desarrollo.
    - *Crear la pantalla de presentación de resultados de la búsqueda de hoteles, crear los métodos de consulta a BBDD para que retornen resultados, mostrar mensaje si no se encuentran resultados con los criterios de búsqueda, probar integración del modulo de socios con el sitio Web.*



3. Apoyos a las HU

Requisitos no funcionales

Los RNF pueden tratarse como restricciones asignadas a cualquiera de los elementos que tratamos en las pilas de desarrollo (Temas, Épicas, HU y Tareas).

Modelo INVEST

¿Cuándo consideramos que una historia de usuario es buena?

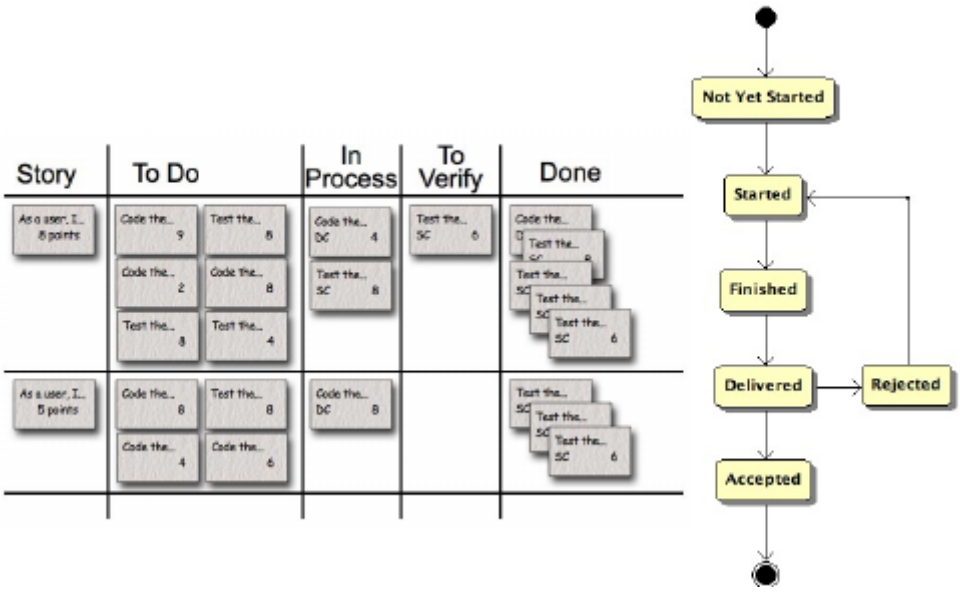
- Independiente: Una historia debería ser independiente de otras (lo más posible).
- Negociable: Las HU no son un contrato o un requerimiento que el software debe implementar.
- Valiosa: Tiene que tener valor para el cliente (para el usuario o para el comprador). El cliente las escribe, no son un contrato y son negociables.
- Estimable: Para permitir que se pueda priorizar y planificar la historia.
- Suscinta: Una buena historia debe ser pequeña en esfuerzo (2-3 personas/semana de trabajo).
- Testeable: Una historia necesita poder probarse para que ocurra la etapa de "Confirmación".

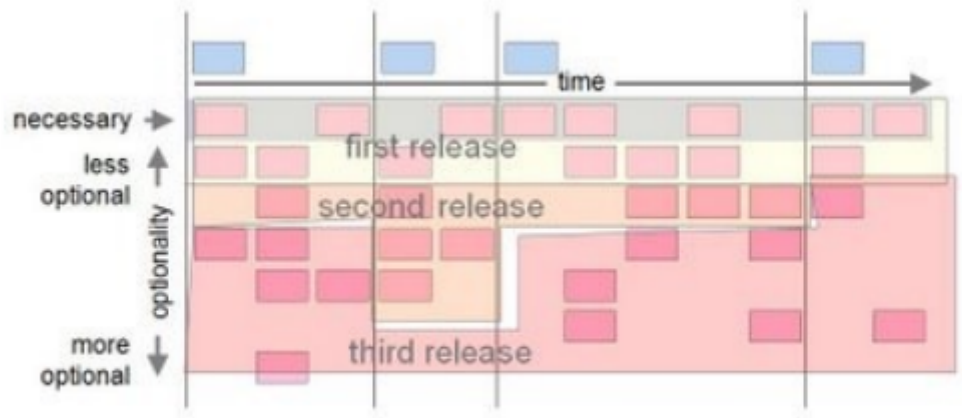
Pila de producto (*Product backlog*)

> "Lista priorizada de HU que contiene toda la funcionalidad deseable del producto".

- Prioridad. Valor que le da el cliente a la HU (Medida en la que permite satisfacer sus necesidades).
- Cubre todas las funcionalidades necesarias.
- Se revisa/actualiza frecuentemente.
- Es una herramienta de planificación y gestión.

Mapa de historias de usuario





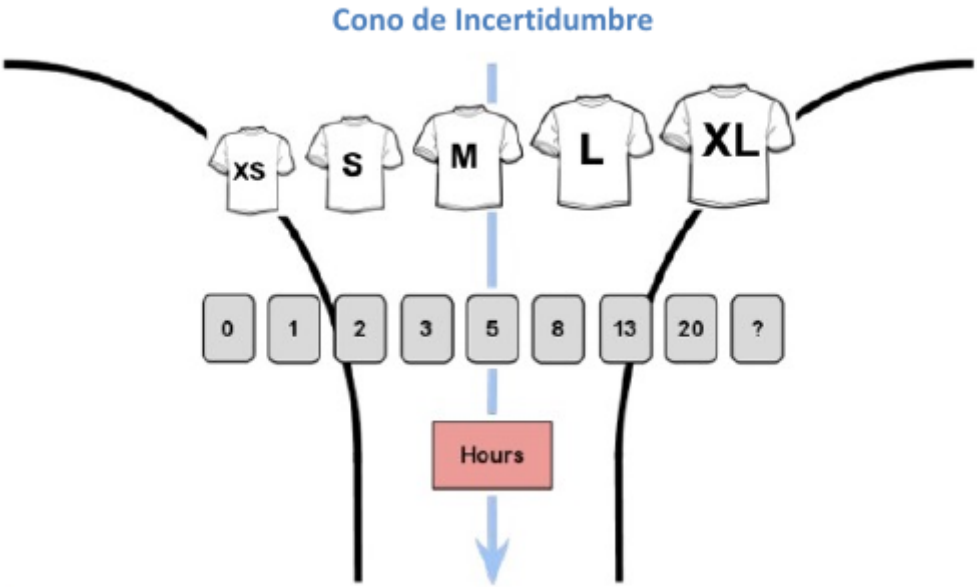
4. Proceso de definición de las HU

- Toma de requisitos en reuniones con usuarios/clientes (Entrevistas, cuestionarios, observación).
- Informar de la visión general transmitida por el cliente.
- Crear HU (reuniones de escritura de HU).
- Actuar sobre cada HU mediante un estudio funcional y técnico realizado en equipo (Conversación).
- Definir criterios de aceptación.
- Estimar el tamaño de las HU para priorizarlas.
- Descomponer las HU en tareas.

Estimar el tamaño

Estimación. “Puntos de historia” o “Días ideales”, estimados por el equipo de desarrollo.

- Puntos de Historia: Tamaño relativo de una historia comparado con otras historias estimadas por el mismo equipo.(esfuerzo necesario + complejidad + riesgo + ...)
- Días ideales: Un día de trabajo sin interrupciones.



Planning poker

- Técnica para realizar una estimación en base al consenso entre los estimadores.
- Un mazo típico contiene tarjetas mostrando la Secuencia de Fibonacci (0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89) o (0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100)
- Existe un moderador que no estima y gestiona la reunión.
- El desarrollador con más conocimiento de una determinada característica proporciona una breve introducción sobre la misma.
- El equipo tiene la oportunidad de hacer preguntas y discutir para aclarar los supuestos y riesgos.
- Cada persona coloca una carta del mazo boca abajo representando su estimación.
- Todo el mundo muestra sus cartas a la vez.
    - A las personas con estimaciones fuera del promedio (altas y bajas) se les da un tiempo para ofrecer su justificación para la estimación y la discusión continúa.
    - Se repite el proceso de cálculo hasta que se alcance un consenso.
    - Se asegura de respetar los tiempos asignados a cada estimación, permitiendo hacer de esta reunión un ejercicio productivo y eficiente.

Priorizar

1. Requeridas o críticas. Sin las cuales la solución no puede vivir (entregas primeras del proyecto).
2. Importantes. Sin las cuales el sistema puede vivir durante algún tiempo (entregas intermedias del proyecto).
3. Opcionales. Si hay tiempo y presupuesto se construyen. (entregas finales del proyecto).
4. No se construirán. Por restricciones de presupuesto o de tiempo o no tienen ningún valor para el negocio.

Las HU pueden ser negociadas con el cliente y van a evolucionar a lo largo del proyecto. Priorizar las HU en base al valor para el cliente (Valor de Negocio).

- Tener en cuenta el tamaño y los riesgos de cada objetivo.
- La prioridad puede cambiar, el tamaño no.
    - HU importante para un número alto de usuarios.
    - HU importante para un número pequeño de usuarios importantes.
    - HU muy relacionadas con otras HU priorizadas previamente.
- Método MOSCOW
    - M – Must, se debe completar este requerimiento para finalizar el proyecto.
    - S – Should, se debe completar este proyecto por todos los medios, pero el éxito del proyecto no depende de él.
    - C – Could, se debería completar este requerimiento si su implementación no afecta a la consecución de los objetivos principales del proyecto.
    - W – Would, se puede completar este requerimiento si sobra tiempo de desarrollo (o en futuras versiones del mismo).

Velocidad del equipo

Es la cantidad de puntos de historia que un equipo puede completar en una iteración. Se calcula por observación en varias iteraciones del proyecto.

Permite planificar las HU que se abordaran en una iteración.

- Utilizar datos históricos
- Ejecutar una iteración inicial y utilizar la velocidad de esa iteración.
- Hacer un pronóstico (“el cálculo de recursos”)

Planificación de iteraciones

Asignar HU a las iteraciones dentro de cada una de las entregas de software

Story	Story Points
Story A	3
Story B	5
Story C	5
Story D	3
Story E	1
Story F	8
Story G	5
Story H	5
Story I	5
Story J	2

Velocidad del equipo = 13

Iteration	Stories	Story Points
Iteration 1	A, B, C	13
Iteration 2	D, E, F	12
Iteration 3	G, H, J	12
Iteration 4	I	5

Iteration	Stories	Story Points
Iteration 1	A, B, C	13
Iteration 2	D, E, F	12
Iteration 3	G, H, Y	13
Iteration 4	J, Z	4

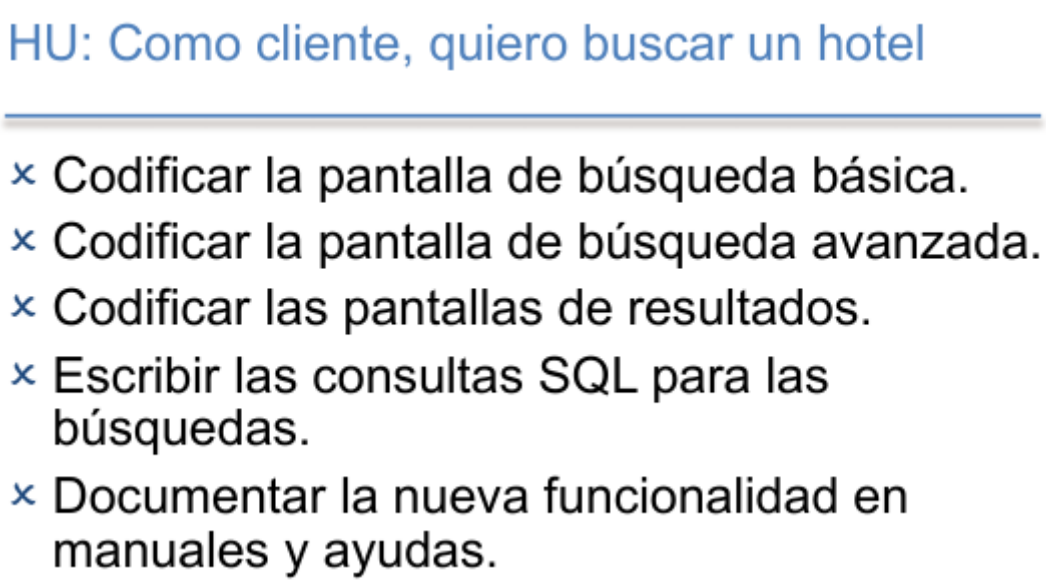
Descomponer en tareas

- Las HU hay que descomponerlas en tareas que puedan ser tratadas de forma individual por el equipo de desarrollo.
- La duración de una tareas debe estar entre medio día a 3 ó 4 días de trabajo.
- Una vez terminada la tarea se debe generar un producto entregable.
- No dedicar mucho tiempo a estudiar detalles de las tareas.
- Incluir tareas de prueba y automatización.

HU: Como administrador quiero poder realizar un mantenimiento de los clientes de la empresa.

- Construir la IU.
- Construir la lógica de negocio.
- Construir la capa de datos.

- Implementar añadir usuario.
- Implementar modificar usuario.
- Implementar eliminar usuario.
- Realizar pruebas de añadir/modificar/eliminar usuario.



5. Partir HU grandes

1.Pasos del flujo de trabajo.

Identificamos los pasos que realiza el usuario para hacer una actividad e implementamos esos pasos de una forma incremental.

Como un vendedor, quiero actualizar y publicar los nuevos programas de precios para los clientes.

- *… quiero publicar los programas de precios en la página principal.*
- *… quiero enviar un mensaje a cada cliente con los nuevos precios, usando el portal web.*
- *… quiero enviar la lista de precios actualizados con descuentos personalizados a cada uno de los clientes.*

2. Variaciones de las reglas de no negocio

Las HU pueden resultar sencillas si no tenemos en cuenta todas las posibles reglas de negocio que hay que aplicar.

Como un vendedor, quiero calcular el descuento que le vamos a aplicar a un cliente en todas sus compras.

- *… teniendo en cuenta las últimas compras que ha realizado.*
- *… teniendo en cuenta el potencial de venta que tiene en función de la demografía.*

3. El mayor esfuerzo de desarrollo

Algunas veces podemos partir una HU en varias partes de forma que las primeras tendrán un mayor esfuerzo de desarrollo y las siguientes van a consistir en añadir mayor funcionalidad.

Como un usuario, quiero poder cambiar la tarifa de electricidad que se me aplica usando un selector en la página de cliente.

- *… quiero usar la tarifa de medida de uso de electricidad.*
- *… quiero usar la tarifa de prepago.*
- *… quiero usar las tarifas de descuento.*

4. Sencillo / complejo

Si durante la descripción de la HU cada vez se va complicando y parece ser más larga, hay que parar y preguntarse, ¿qué es lo más simple que podría funcionar? Luego podremos añadir variaciones y complejidad sobre nuevas HU.

Como un usuario, quiero tener siempre el precio más bajo de un producto.

- *… teniendo en cuenta el proveedor que de el precio más bajo.*
- *… avisando de posibles ofertas que van a aparecer en el futuro próximo.*

5. Variaciones en datos

Las variaciones sobre los datos a usar pueden ser un origen de aumento de la complejidad.

Como un administrador, quiero enviar mensajes a los clientes.

- *… en inglés.*
- *… en español.*

6. Métodos de entrada de datos (Vista del IU)

Muchas veces la complejidad está en el IU más que en la funcionalidad. Podemos empezar con una funcionalidad básica y añadir posteriormente una interacción más rica.

Como un usuario, quiero ver la evolución de mis compras usando gráficos estadísticos.

- *… usando una gráfica de barras que compara las ventas mensuales.*
- *… usando una gráfica de barras con el criterio seleccionable de un conjunto definido.*

7. Niveles de calidad del sistema

Muchas veces las primeras implementaciones no son lo suficientemente eficientes como podrían ser. El equipo puede aprender mucho del desarrollo y posteriormente darle un mayor valor al cliente.

Como un usuario, quiero ver resultados en tiempo real de mis medidas.

- *… usando medidas interpoladas de las últimas medidas tomadas.*
- *… usando medidas obtenidas en tiempo real del sistema.*

8. Operaciones múltiples

Operaciones como gestionar, administrar, controlar pueden ser divididas en múltiples operaciones.

Como un usuario, quiero poder gestionar mi cuenta.

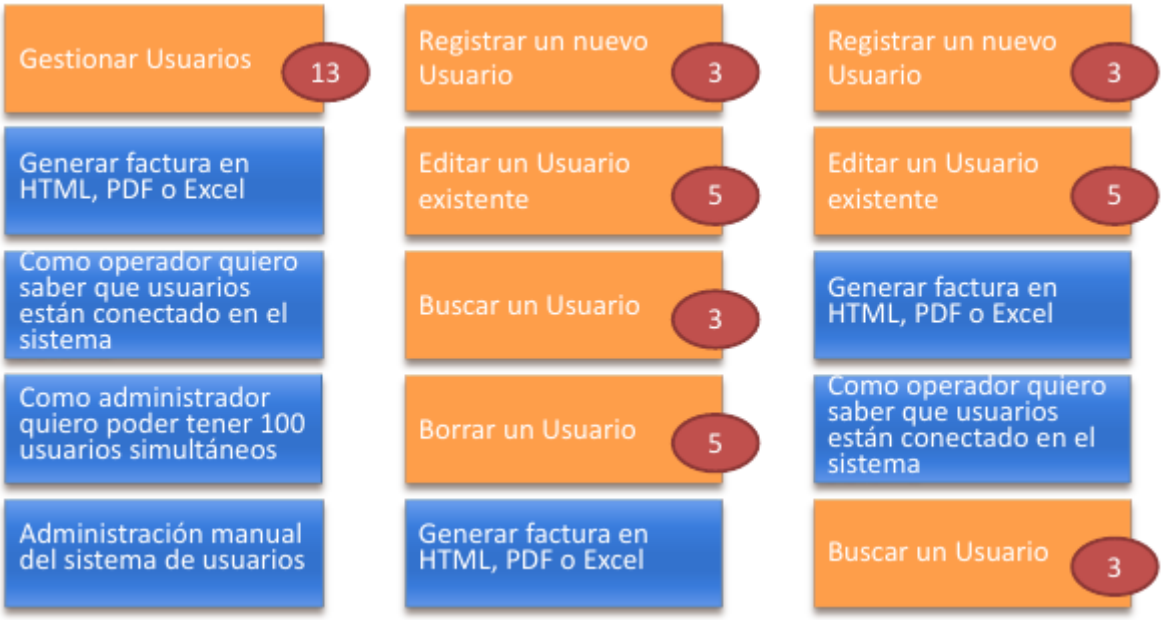
- *… darme de alta como cliente.*
- *… editar los parámetros de la cuenta.*
- *… cancelar la cuenta.*
- *… añadir más dispositivos a la cuenta que tengo asociada.*

9. Escenarios de un caso de uso

Si partimos de un estudio de casos de uso, podemos partir cada uno de ellos en escenarios individuales.

Como un usuario, quiero poder gestionar mi cuenta.

- *... darme de alta como cliente.*
- *... editar los parámetros de la cuenta.*
- *... cancelar la cuenta.*
- *... añadir más dispositivos a la cuenta que tengo asociada.*



Priority	Feature
1	Book search capability (title, author, subject)
2	Online shopping cart
3	Credit card processing integration
4	Book details
5	Order tracking
6	Book preview

Priority	Feature
1	Book Search by Title
2	Online shopping cart
3	Book Search by Author
4	Credit card processing integration
5	Book Search by Subject
6	Book details
7	Order tracking
8	Book preview
9	Book Search by ISBN

6. SPIKE

Tipo de HU usada para gestionar una dificultad técnica. Tiempo necesario para reducir un riesgo o una incertidumbre de un problema técnico o para aumentar la seguridad sobre una estimación de una HU.

- SPIKE Técnico: Analizar varias aproximaciones técnicas en el dominio de la solución del problema.
    - Decidir si comprar o desarrollar algo para la solución.
    - Evaluar el rendimiento de una solución propuesta.
    - Evaluar el impacto de carga de una nueva HU.
- SPIKE Funcional: Si hay una incertidumbre importante en cómo el usuario debe interactuar con el sistema.

Puede darse por:

- El equipo no tiene suficiente conocimiento sobre un dominio y usamos la HU para familiarizarnos con el dominio o una tecnología nueva.
- Tenemos una HU grande y la estimación necesita ser partida en trozos y analizar lo que implica.
- La HU tiene riesgos técnicos y el equipo necesita realizar un prototipo o algo de desarrollo para ganar confianza en el desarrollo o el diseño a realizar.
- La HU tiene riesgo funcional y no está claro como tiene que actuar el sistema con usuarios para alcanzar los objetivos planteados.

> EJEMPLO SPIKE
>
> Como un cliente, quiero ver el gasto que he tenido de electricidad en un histograma, de forma que pueda comprender el gasto realizado y el que se prevé que gaste.
>
> - SPIKE Técnico. Investigar cuánto tiempo se tarda en actualizar una pantalla personalizada con el uso actual y los requisitos de comunicación y de gestión que tendrá.
> - SPIKE Funcional. Realizar un prototipo de histograma en el portal web y obtener realimentación de varios usuarios sobre la forma de presentar la información.

Reglas de las SPIKE

- Se tratan como otras HU (las ponemos en el backlog, las priorizamos, las estimamos, etc.) Su resultado no es código, es información útil para otras HU.
- Debe generar un decisión, un prototipo, una demostración de un diseño, o una solución parcial de un problema.
- La solución de un SPIKE debe ser demostrable al equipo de desarrollo.
- Un SPIKE, como cualquier HU, debe ser aceptada usando criterios de aceptación definidos en la HU.
- Puede ser interesante dejar la resolución del SPIKE para una interacción diferente a la de la HU que lo creó.