Especificación

1.- El proceso de desarrollo

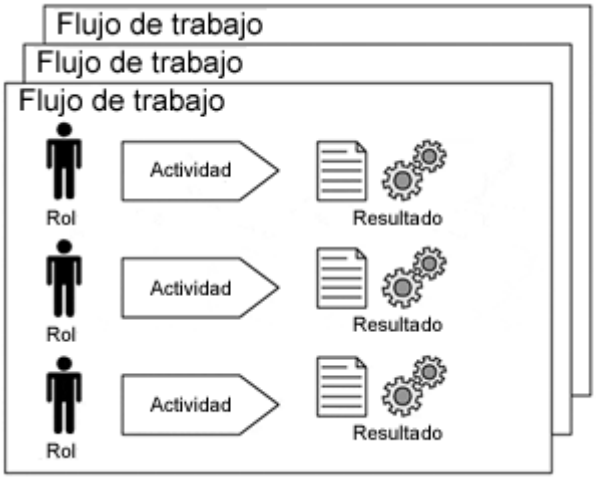
Contenidos

El desarrollo de aplicaciones es una tarea de gran complejidad, aunque el producto a desarrollar pueda parecer relativamente sencillo. Es necesaria una **metodología** (el proceso de desarrollo) que:

- **guíe** las actividades del equipo de desarrollo.
- defina los **resultados esperados** del proceso.
- dirija las **tareas** de los equipos.
- ofrezca criterios para la **monitorización** y medida del progreso del proyecto.

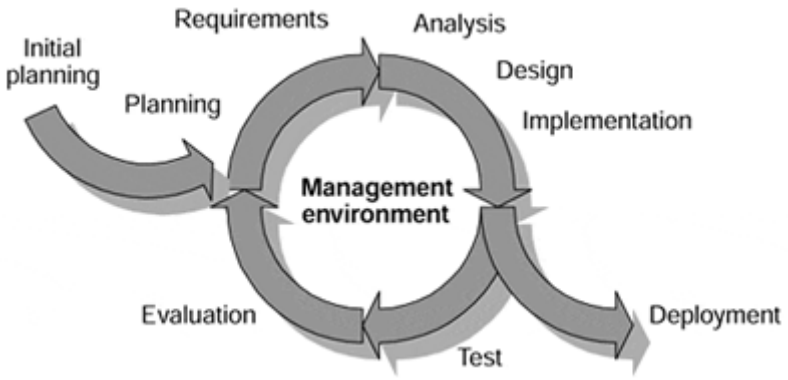
Por lo tanto, el proceso de desarrollo constituye la guía indispensable para la gestión del desarrollo de software. En este caso la base son los procesos de desarrollo Rational Unified Process [Kruchten98] y ICONIX Unified Process [Rosenberg99], adaptados para el caso de aplicaciones Web. El proceso define flujos de trabajo, actividades, resultados y roles:

- **Flujo de trabajo**: conjunto de actividades que producen un resultado.
- **Actividad**: las acciones que los individuos implicados en el flujo llevan a cabo para producir los resultados.
- **Resultados**: elementos de información producidos por el proceso.
- **Roles**: interpretados por los individuos participantes.



Componentes del proceso de desarrollo: flujos de trabajo, roles, actividades y resultados.

Los **flujos** de trabajo que componen el proceso de desarrollo son: ***Requerimientos***, ***Análisis***, ***Diseño***, ***Implementación***, ***Prueba***, ***Evaluación*** y ***Entrega***. Estos flujos se encadenan constituyendo pasos sucesivos encaminados a completar el desarrollo del sistema. Para evitar desviaciones excesivas en el producto desarrollado, se suelen desarrollar de forma **iterativa e incremental**. Cada ciclo acerca el producto resultante a los requerimientos iniciales pero se obtienen resultados antes con lo que se facilita la detección de errores, mal interpretaciones, aspectos poco definidos, etc. El objetivo es afrontar los elementos de riesgo del proceso cuanto antes mejor.



Rational Unified Process, iterativo e incremental, imagen de [Kruchten98]

Requerimientos

Un requerimiento es un enunciado sobre **lo que el sistema debe hacer**. El conjunto de todos los requerimientos es la **especificación de requerimientos**. El objetivo es expresar de forma inambigua **QUÉ** es lo que se espera del sistema a desarrollar. El CÓMO es lo que se responde en las subsiguientes flujos de trabajo: análisis y diseño. Una forma fácilmente comprensible de expresar los requerimientos es mediante **casos de uso**. Estos expresan de forma parcialmente formal escenarios de uso del sistema.

Análisis

Se analizan los requerimientos y se confecciona un **modelo conceptual del sistema**. Los resultados incluyen clases y colaboraciones, diagramas de secuencia, diagramas de estado y diagramas de actividad. Estos mismos resultados se reelaboran durante el diseño, la diferencia es que por ahora no se aplica ninguna arquitectura. Por lo tanto se describe el sistema **abstrayendo la arquitectura**. El objetivo es **evitar** por el momento **detalles técnicos** y por lo tanto facilitar la comunicación con expertos del dominio de aplicación y futuros usuarios.

Diseño

Se toman los resultados del análisis y se les **aplica la arquitectura**. El objetivo es hacer que el **modelo** del sistema sea realizable en software, es decir **implementable**. Se concretan por lo tanto los aspectos abstractos del modelo.

Implementación

Se traduce el modelo del sistema en **código**. Se pueden aplicar herramientas de desarrollo al modelo que parcialmente automatizan el mapeo del modelo a código ejecutable, herramientas CASE (Computer Aided Software Engeneering). Aunque las limitación de estas herramientas restringen el mapeo a los elementos más estructurales, dejando en manos del implementador los elementos dinámicos del sistema.

Prueba y Evaluación

Se evalúan los resultado de las pruebas de los ejecutables del sistema. Se desarrollan **tests para** comprobar que el sistema satisface tanto los **requerimientos funcionales como técnicos**. Para los primeros se utilizan la especificación de **casos de uso como guía** para determinar que se debe comprobar y cuáles son los resultados esperados. Para los requerimientos técnicos se somete el sistema a **pruebas de carga, rendimiento, seguridad,...**

Las pruebas se desarrollan en diferentes estadios del desarrollo:

- *Test de unidad*: llevado a cabo por los implementadores sobre las unidades desarrolladas por ellos.

- *Test de integración*: se evalúan las interfaces de los componentes y su funcionamiento conjunto. Se realizan cuando se conectan partes del sistema, antes del ensamblado del sistema completo.
- *Test de sistema*: comprobación de los requerimientos sobre el sistema completo.
- *Test de aceptación*: llevado a cabo por la comunidad de usuarios.

Las pruebas del sistema se planifican con antelación a la implementación del sistema y se desarrollan en paralelo.

Entrega

Corresponde a la **puesta en marcha** en "casa del cliente" de la aplicación. Este paso puede ser tan sencillo como instalar la aplicación en un ordenador, pero en muchos casos es algo más complejo. Si existen requerimientos de funcionamiento a prueba de fallos, alto rendimiento, etc. la instalación puede suponer la instalación en múltiples ordenadores, la **configuración** de módulos de balanceo de carga entre servidores,... También se debe considerar el **mantenimiento** de la aplicación.

Modelo

Para **facilitar la comunicación** entre los participantes, los resultados del proceso se encaminan a componer un modelo, un **representación abstracta del sistema a desarrollar**. El modelo se basa en un **lenguaje compartido de modelado**, en este caso **UML** (Unified Modeling Language) [Fowler03]. Cada **flujo** de trabajo genera ciertos **tipos de resultados**, predefinidos por el lenguaje de modelado, encaminados a la progresiva definición del modelo. Partiendo de resultados de un alto nivel de abstracción, **el modelo se concreta progresivamente** en el producto final.

Los resultados de mayor nivel de abstracción, con los que se inicia el modelo y que guían todo el proceso, son los casos de uso. Es por esto que el proceso de desarrollo se dice "dirigido por casos de uso".

Referencias

[Kruchten98]	Philippe Kruchten: "The Rational Unified Process: An Introduction". Addison Wesley, 1998
[Rosenberg99]	Doug Rosenberg, Kendall Scott: "Use Case Driven Object Modeling with UML: A Practical Approach". Addison Wesley, 1999
[Fowler03]	Martin Fowler: "UML Distilled: A Brief Guide to the Standard Object Modeling Language". Addison Wesley, 2003 http://safari.awprofessional.com/0321193687

2.- Especificación de requerimientos

Introducción

Un **requerimiento**, o restricción que el sistema debe cumplir, se expresa normalmente como un enunciado del estilo "El sistema debe...". Su propósito es expresar un **comportamiento o propiedad que el sistema debe tener**. Debe ser específico y claro, comprensible tanto para el equipo de desarrollo como para los clientes y usuarios. Una cualidad importante de un buen requerimiento es que el equipo de testeo pueda verificarlo en la aplicación final.

En general, los requerimientos se pueden categorizar como **funcionales o no funcionales**. Lo requerimientos **funcionales** expresan **acciones que el sistema debe realizar** y normalmente se definen como respuestas del sistema a estímulos externos. Son el tipo de requerimiento más común. Ejemplos de

requerimientos funcionales:

- "El sistema debe ser capaz de calcular cargos de entrega internacional para todos los productos ofrecidos."
- "El sistema debe producir automáticamente un informe con todas las ventas de la semana."

Los requerimientos **no funcionales** se dividen en:

- **Usabilidad:** los requerimientos de usabilidad se refieren a los aspectos generales de la interfaz entre el usuario y el sistema. Normalmente se refieren a estándares de interfaz de usuario. Para aplicaciones Web, los requerimientos de usabilidad deben incluir las funciones mínimas del navegador utilizado por los usuarios o los elementos HTML a usar.
  Por ejemplo: "La interfaz no debe usar frames HTML" o "El sistema debe ser accesible por cualquier navegador que soporte formularios".
- **Rendimiento:** los requerimientos de rendimiento describen el rendimiento de ejecución del sistema y normalmente están relacionados con el tiempo.
  Por ejemplo: "Las páginas Web no deben tardar más de 15 segundos en cargarse".
- **Robustez:** se define el grado de disponibilidad de la aplicación, idealmente siempre pero en la práctica definido como el tiempo máximo aceptable de no disponibilidad. También se refiere a cuestiones de copias de seguridad y almacenamiento.
  Por ejemplo: "El sistema debe dar acceso a datos en copia de seguridad semanal durante una franja de 2 horas".
- **Seguridad:** los requerimientos de seguridad tienden a especificar niveles de acceso al sistema y normalmente se mapean a roles de negocio desarrollados por los usuarios. También deben incluir el acceso al sistema por parte de otros sistemas externos.
  Por ejemplo: "El sistema asegurará que toda la información confidencial suministrada por los usuarios a través de Internet estarán encriptadas".
- **Hardware:** estos requerimientos se refieren al mínimo hardware necesario para implementar el sistema. Es importante recordar que no se refieren únicamente al hardware de servidor, también al de los clientes.
  Por ejemplo: "El sistema debe ser capaz de ejecutarse en la configuración estándar de los equipos de cliente de la compañía: Pentium II 800Mhz, 256MB RAM, SVGA 800 x 600 x 16".
- **Entrega:** describen como la aplicación llega a sus usuarios. Proporcionan restricciones sobre como se instala y mantiene la aplicación.
  Por ejemplo: "El software de cliente debe poder descargarse e instalarse desde el navegador sin necesidad de reiniciar el ordenador".

En teoría la especificación de requerimientos termina cuando se ha especificado completamente el sistema. La especificación constituye entonces el **contrato** que compromete al cliente a aceptar un sistema que cumpla los criterios especificados. En la práctica, seguramente algunos requerimientos quedarán sin especificar, otros serán demasiado restrictivos o incluso equivocados. Así, cierto grado de **flexibilidad** es necesario y los requerimientos pueden cambiar como parte plena del proceso iterativo. Pese a todo, los cambios en una buena especificación deberían de ser mínimos.

Recogida de requerimientos

La recogida de requerimientos se realiza en grupo. Como mínimo un equipo de requerimientos está constituido por un representante de los clientes o los futuros usuarios y otro del equipo de desarrollo. Cada requisito específico del sistema debe tener un identificador único que facilite su seguimiento.

Por ejemplo:

3. **Requerimientos de rendimiento:** Esta sección describe los requerimientos de funcionamiento del sistema. Estos requerimientos se relacionan generalmente con la velocidad de la ejecución y la capacidad de los componentes individuales del sistema.

3.1 **Rendimiento del servidor de Web:** Esta sección describe el funcionamiento previsto del servidor del Web.

3.1.1 Cada servidor del Web podrá manejar por lo menos 150 sesiones simultáneas del usuario.

3.1.2 El sistema requerirá no más de 3 segundos para mostrar una página estática al cliente.

Finalmente, es importante priorizar los requerimientos, por ejemplo con las etiquetas de prioridad "alta", "media" y "baja". Esto ayuda a resituar los requerimientos en el contexto global y a guiar los siguientes pasos del desarrollo.

3.- Casos de uso

La **especificación de requerimientos** es una buena manera de capturar y priorizar los requerimientos no funcionales, pero aporta poco a hora de detallar los **funcionales**. Constituye únicamente el primer paso. Una buena herramienta para formalizar los requerimientos funcionales es el análisis de "**casos de uso**"[Jacobson92, Cockburn97].

Los casos del uso son una manera formal de capturar y de **expresar la interacción entre los usuarios y el sistema**. Como usuarios también se entienden otros sistemas externos por lo tanto nos referiremos a ellos de forma más genérica como agentes. Los usuarios pueden jugar más de un rol cuando interaccionan con el sistema. Cada tipo de **rol** interpretado por algún usuario se denomina **actor**.
Por ejemplo, un agente puede desempeñar el rol de usuario anónimo, con ciertos derechos de acceso, registrarse y luego pasar a desempeñar el rol de usuario registrado. Los actores serían entonces "usuario anónimo" y "usuario registrado".

Por lo tanto, un caso del uso contiene una **descripción narrativa de un escenario específico de uso, en el cual un agente desempeña un rol, provee la entrada y el sistema exhibe una salida**. Un **caso del uso puede contener más de un escenario**. Existe un **escenario principal**, que describe el curso de acontecimientos más habitual, y **escenarios alternativos** que describen situaciones menos frecuentes o anómalas. El elemento común en todos los escenarios de un caso de uso es que el agente busca en **todos** ellos el **mismo objetivo inicial**.
Por ejemplo, cuando el usuario anónimo se registra, el curso de acontecimientos principal es el que acaba añadiendo al usuario anónimo al registro. Situaciones anómalas que habría que considerar como escenarios secundarios podrían ser que el identificador del usuario ya está registrado o que alguno de sus datos, p.e. el NIF, no es válido.

Por cada caso de uso detallar:

- **Identificador**: facilita el seguimiento.
- **Actores**: detalla el actor principal, el que inicia la interacción con el sistema, y los actores secundarios, para éstos el sistema es quien inicia la interacción.
- **Objetivo**: enunciado que resume el objetivo del caso de uso.
- **Descripción**: texto descriptivo del comportamiento de los escenarios, posiblemente con referencias a otros casos de uso más básicos que son incluidos por este (son invocados por él):
    - **Escenario principal** / **Escenarios alternativos**.
- **Precondiciones**: condiciones necesarias antes de que se realice el caso de uso.
- **Postcondiciones**: condiciones que deben satisfacerse al finalizar el caso de uso, concretar para escenario principal y alternativos.
- **Requerimientos satisfechos**: lista de los identificadores de los requerimientos que este caso satisface.
- **Autores**: miembros del equipo que han contribuido directamente al texto del caso de uso.
- **Prioridad**: la prioridad del caso a partir de las definidas en la especificación de requerimientos.

- **Riesgo**: probabilidad de que algo pueda ir mal durante el desarrollo del caso de uso. Por ejemplo por la inexperiencia del equipo en las tecnologías que implica o por la intervención de sistemas externos. Un alto riesgo conlleva que se afronte antes en el proceso de desarrollo. El riesgo junto con la prioridad ayudan a planificar el desarrollo.
- **Cuestiones pendientes**: cosas a resolver antes de que el caso pase a la fase de análisis y diseño. Esta sección se utiliza solamente durante su formalización y debe estar en blanco antes de pasar a la siguiente fase de desarrollo.

Metodología

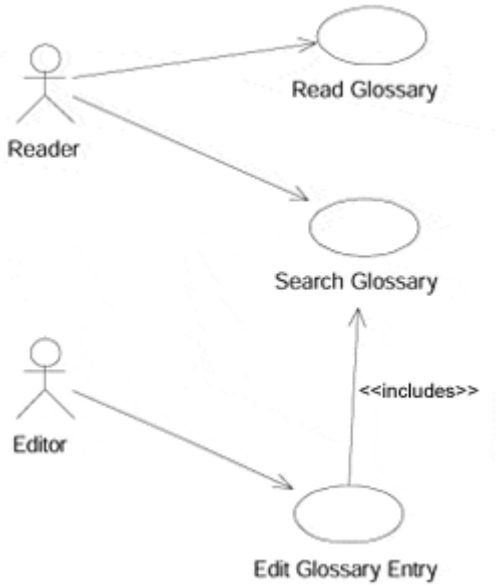
Para redactar los casos de uso, trabajar sobre todo el sistema (todos los requerimientos) incrementando paulatinamente el nivel de detalle [Cockburn00]:

1. Detectar todos los actores principales.
2. Para cada actor identificar sus objetivos. Para cada objetivo detectar los requerimientos satisfechos.
3. Redactar la descripción del escenario principal para cada objetivo. Añadir precondiciones y postcondiciones.
4. Detectar las condiciones que llevan a escenarios alternativos.
5. Redactar los escenarios alternativos. Añadir las postcondiciones particulares de esos escenarios.

Diagrama de casos de uso

Los diagramas UML de casos de uso facilitan el trabajo con los casos de uso mediante una **representación gráfica**. Se representan los casos de uso, las relaciones entre ellos (etiquetadas como <<includes>>) y con los actores del sistema (flechas de los actores principales a casos de uso y de casos de uso a actores secundarios). Existen otros tipos de relaciones entre casos de uso: <<uses>> que se puede considerar equivalente a includes y <<extends>> que no utilizaremos.

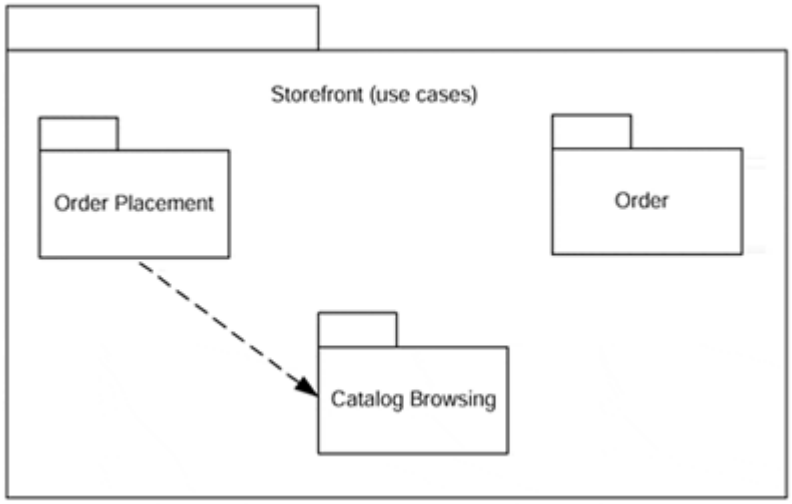
Las relaciones entre casos de uso permiten **reutilizar casos de uso**. Si un caso de uso requiere el comportamiento formalizado en otro caso de uso, se establece una relación <<includes>> que parte del caso de uso que reutiliza el comportamiento hacia el caso de uso reutilizado.



Ejemplo de diagrama de casos de uso

Incluso aplicaciones relativamente pequeñas representan un número significativo de casos de uso. Para facilitar su gestión se acostumbran a organizar en paquetes. Cada paquete contiene un conjunto de casos de uso o incluso otros paquetes. Un paquete es un mecanismo UML para romper un modelo en piezas más

manejables.



Organización de casos de uso en paquetes

En un diagrama de paquetes, estos se muestran como carpetas. Los paquetes incluidos en otros aparecen en su interior y flechas señalan la dependencia entre paquetes si alguno de los casos de uso del paquete dependiente incluye algún caso de uso del paquete objetivo. Los paquetes permiten, una vez completados los casos de uso, distribuir el trabajo de los siguientes pasos entre diferentes grupos de trabajo del equipo de desarrollo. La coordinación entre ellos será especialmente importante en el caso de paquetes con interdependencias.

Conclusiones

Los casos de uso serán a partir de ahora la guía del proceso de desarrollo y se utilizarán en las siguientes fases de desarrollo (análisis, diseño, implementación y prueba), para la comunicación con el cliente y para la confección del manual de usuario.

Referencias

[Jacobson92]	Ivar Jacobson, Magnus Christerson, Patrik Jonsson and Gunnar Övergaard: "Object-Oriented Software Engineering: A Use Case Driven Approach". Addison-Wesley, 1992
[Cockburn97]	Alistair Cockburn: "Structuring Use Cases with Goals". Journal of Object-Oriented Programming, Sept-Oct 1997 - Nov-Dec 1997
[Cockburn00]	Alistair Cockburn: "Writing effective use cases". Addison-Wesley, 2000 (extract)