2. Introducción a la ingeniería de requisitos

Qué es la Ingeniería de Requisitos

- Es la *rama de la ingeniería del software* que se ocupa de la primera etapa en el proceso de desarrollo del software: la *comprensión y formalización de las necesidades* que debe satisfacer un sistema informático
- "Es el desarrollo sistemático de los requisitos a través de un proceso *iterativo* y *cooperativo* en el que se analiza el problema, se *documenta* el resultado en diversos formatos de representación, y se *comprueba* la exactitud de la comprensión alcanzada" (Locopoulos & Karakostas. *Systems Requirements Engineering*. McGraw-Hill, 1995)
- Se pueden distinguir *dos fases* en el proceso:
  - *Captura*: interacción cuidadosa con todos aquellos interesados en la aplicación o sistema informático; *adquisición de información*, "requisitos en bruto"
  - *Análisis*: estudio cuidadoso de la información adquirida para lograr una verdadera *comprensión de los requisitos* y estructurarlos adecuadamente; expresar los requisitos de forma concreta y detallada; refinamiento, depuración, estructuración, "requisitos depurados"
- Obtener los requisitos correctos es un proceso difícil
  - Adivinar los deseos y necesidades que habitualmente el cliente no es capaz de describir más que en forma confusa, incompleta y desordenada
  - Los requisitos *no se descubren, se inventan*; los requisitos no están ahí esperando que alguien los descubra, sino que son creados, construidos o inventados en un proceso interactivo entre el cliente y el ingeniero
  - La mayor parte de los *defectos en el software* entregado tienen su origen en el análisis de requisitos, y son en general los más difíciles de reparar
  - El éxito en el producto requiere *colaboración y comunicación* fluida entre clientes y desarrolladores: cuanto más completo y menos ambiguo sea el conjunto de requisitos, más probabilidades de éxito
- El *resultado del proceso* es el "documento de requisitos", que es uno de los productos (o *artefactos*) del proceso de desarrollo de software: modelos de diseño, código fuente, pruebas, manuales...

Un caso práctico

- Agenda de compromisos: "Necesito algo para organizar mejor mis actividades, una agenda para llevar al día mi horario, mis compromisos, etc."
- ¿Cuánto se tardaría en desarrollar esta aplicación?
  - "¡Esto me lo curro yo en una semanita!"
- En el momento de la entrega de la agenda el cliente no queda satisfecho
  - Quiere más, quiere otra cosa
  - Colores, sonidos, funciones, copiar compromisos de un día a otro…
  - La divertida semanita se convierte en una pesada carga
  - ¿Hasta qué punto me he comprometido?
- Antes de entregarla, quiero probarla…
  - ¿qué pruebo?
  - ¿qué porcentaje de funcionalidad he alcanzado?
- La ingeniería del software no trata de "programar bien", sino de:
  - Cumplir plazos, presupuesto y expectativas
  - Gestionar riesgos y recursos
  - Transformar la producción artesanal en industria
- ¿Termina el ciclo de vida del software en la entrega del producto? Grave error: mantenimiento.

Necesidad de la Ingeniería de Requisitos

- Para construir algo, antes hay que entender qué es ese "algo"
- Si la aplicación funciona, pero no satisface las necesidades del cliente... ¿de qué sirve?
- En general, los requisitos expresan *qué* debe hacer una aplicación, sin decir *cómo* debe hacerlo: expresan el punto de vista del cliente, que sabe lo que quiere (en el mejor de los casos), pero no tiene por qué saber cómo conseguirlo
    - A menudo el cliente ni siquiera sabe lo que quiere
    - Tarea del analista es ayudar a que el cliente se exprese
- Ejemplo:
    - El sistema permitirá al usuario consultar el saldo de su cuenta (sí)
    - Los saldos de clientes se almacenarán en una tabla llamada Saldo en una base de datos Access (no)
- Excepciones: puede ocurrir que usar Access sea un requisito
    - Existen distintos niveles de abstracción en los requisitos, debido a que suelen provenir de distintas fuentes (distintos interlocutores con distinto nivel de conocimientos tecnológicos)
    - Ejemplo: el informático de la empresa nos habla de los sistemas con los que deberá interactuar el nuestro, mientras que el jefe de contabilidad nos proporciona una mejor visión sobre el conjunto de procesos del área

Dos niveles en los requisitos

- ¿Del cliente o del sistema, del usuario o del software?
  - Puente a medio camino: *del* cliente *para* el desarrollador
- Primer nivel: *requisitos del usuario* (o del cliente)
  - Deseos y necesidades del cliente, expresados en lenguaje comprensible por él
  - Audiencia primaria: cliente – Audiencia secundaria: desarrollador
- Segundo nivel: *requisitos del software* (o del sistema, del desarrollador, detallados...)
  - Forma estructurada y específica, carácter mucho más técnico
  - Audiencia primaria: desarrollador – Audiencia secundaria: cliente
- La finalidad última es la misma. La distinción entre los dos niveles no es muy clara:
  - Forma: no estructurada / estructurada
  - Audiencia: cliente / desarrollador
  - Contenido: mayor o menor nivel de detalle
  - Texto / Texto + Diagramas
  - Requisitos en bruto → Requisitos depurados
- Distintas nomenclaturas y clasificaciones, misma idea de fondo:
  - Clásica/IEEE:
    - Una única fase: Análisis de Requisitos
    - Único documento *Especificación de Requisitos* con los requisitos C y D
  - USDP/ESA:
    - Dos fases: Requisitos + Análisis
    - Dos documentos: *Requisitos del Usuario+ Requisitos del Software*

Por qué es necesario escribir los requisitos

- Es obvio incluso para el novato, pero con demasiada frecuencia se ignora o deja pasar
- Tarea descuidada durante mucho tiempo: trivial, "literaria", poco tecnológica...
- ¿No quedan bien expresados en el código fuente? No, no funciona.
- Sin requisitos escritos, el equipo de desarrollo:
  - No sabe cuál es su objetivo
  - No puede inspeccionar su trabajo
  - No puede probarlo
  - No puede analizar su productividad
  - No puede reflexionar sobre sus prácticas
  - No puede predecir el tamaño y esfuerzo del siguiente trabajo
  - No puede satisfacer a sus clientes
  - Brevemente, *no hay ingeniería profesional sin requisitos escritos*
- Una forma de NO escribirlos es no mantener las versiones, o no publicarlas
- Cada uno de los requisitos debe ser...
  - Expresado con propiedad
  - Fácilmente accesible para todo el personal involucrado
  - Numerado de forma unívoca
  - Acompañado por pruebas que lo verifiquen
  - Tenido en cuenta en el diseño y el código
  - Probado aisladamente y con otros requisitos
  - Validado por las pruebas al finalizar la construcción de la aplicación
- Sin requisitos escritos sería imposible hacer todo esto
- The Chaos Report (1994, 1996, 1998, 2000, 2003)
  - Exitoso: termina bien (tiempo, dinero, requisitos)
  - Completo pero deficiente: termina y funciona (+ tiempo, + dinero, – requisitos)
  - Cancelado: no termina
- La adecuada gestión de los requisitos es el factor más importante de éxito en un proyecto, por encima de las pruebas, el diseño, la programación...

Dificultades de la Ingeniería de Requisitos

- El precio pagado: un defecto en los requisitos es 20-50 veces más caro de reparar si se desliza sin corregir en el resto del proceso de desarrollo de software (sin contar con el daño que sufre el prestigio del desarrollador)
- La inversión es tanto más valiosa cuanto mayor sea el proyecto
- Si el beneficio es tan grande, ¿por qué tan a menudo se omite el análisis de requisitos?
  - Los clientes normalmente no tienen claro lo que quieren al principio, sólo una idea vaga, y esto convierte el análisis de requisitos en una tarea más difícil
  - Solución: sucesivas iteraciones Requisitos-Diseño-Implementación
- Es una necesidad, no un lujo (voy despacio porque tengo prisa, no puedo permitirme el lujo de gastar tan poco)
- Sin requisitos, no es posible realizar pruebas (que sí se consideran esenciales)
- Muchas organizaciones no logran escribir los requisitos: esto no significa que no los usen, sino que *sólo existen en las mentes* de los ingenieros. Otro peligro: la *rotación de personal*. No es extraño que muchos proyectos nunca terminen.
- Un problema más sutil: *escribir sólo la primera versión* de los requisitos, pero sin mantenerlos al día en sucesivas iteraciones. Para poder actualizar el documento de requisitos es necesario que esté bien organizado. Gestión de requisitos cambiantes.
- La adecuada comprensión de los requisitos es la base del acuerdo sobre la aplicación y de la relación contractual.
- *Escribir es pensar*. Escribir código prematuramente es como echar cemento sin saber cómo va a ser el puente ni dónde tiene que ir. Documentar no es sólo registrar decisiones, es *pensar por escrito*.
- El rechazo o desgana para escribirlos no es porque sea una tarea trivial e inútil, sino porque es demasiado difícil. La profesionalidad lo exige.

La Ingeniería de Requisitos en el contexto del proyecto

- Los requisitos tienen *valor contractual* y establecen los límites de la aplicación en cuanto a la funcionalidad que deben satisfacer
- Contenido del pre-proyecto
  - definición de requisitos
  - modelado
  - prototipado
  - estimación de costes
  - análisis de riesgos
- Pueden estar sujetos a normas de *confidencialidad* y propiedad intelectual
- Los requisitos son la base para los *estudios de viabilidad*:
  - ¿Merece la pena realizar el proyecto?
  - Implementaciones parciales (prototipos) o simulaciones
  - Ej: videojuego por internet, [automatrícula por internet]
  - Prototipos y simulaciones son mini-proyectos: requisitos, documentación, etc.
- La obtención de requisitos afecta a la *planificación* del proyecto. El proceso puede ser iterativo, pero con ciertos límites:
  - El cliente quiere saber el coste antes de empezar
  - El desarrollador no quiere comprometerse al menos hasta congelar los requisitos
- La mejor comprensión de los requisitos facilita refinar las *estimaciones* de esfuerzo necesario: equilibrio planificación-presupuesto-requisitos
- El conjunto de documentos del proyecto está vivo, varios se ven afectados en cada fase, gestionarlos no es fácil
  - Números de versión de secciones y subsecciones del documento de Requisitos

3. Obtención y descripción de requisitos del usuario

Las fuentes de los requisitos

- Restricciones naturales (físicas, químicas, biológicas, sociológicas...)
- Documentos (leyes, documentos organizativos de la empresa...)
- Personas: deseos y necesidades
- Expertos del dominio
- Otras aplicaciones en uso en el dominio

Plan de trabajo para obtener los requisitos

- Identificar al usuario (usuarios posibles, seleccionar entre ellos)
- Entrevistar al usuario (antes, planificar la entrevista)
- Escribir / describir los requisitos del usuario (sucesivas versiones)
- Revisar los requisitos con el usuario
- Repetir los pasos hasta que el usuario firme el "conforme" del documento

Identificación de *stakeholders* (interesados: usuarios, clientes, propietarios, etc.)

- ¿Quiénes tienen interés en el producto? (Ejemplo: sitio web de comercio electrónico)
  - Los usuarios (pueden ser millones... y saber mucho más que nadie del tema)
  - Los propietarios
  - Los administradores
  - Los desarrolladores, incluso (negociar requisitos para protegerse)
- ¿Quién es el cliente?
  - La persona que te contrata
  - El departamento de Marketing, que diseña un producto para un cliente ideal
  - Aplicaciones a medida, aplicaciones genéricas (*product lines*)
- Los distintos *interesados* pueden tener intereses contrapuestos y generar requisitos contradictorios (¿puede un profesor ver el expediente académico de un alumno?)
  - Ejemplo: El cliente de Aula Global es la Universidad, pero los usuarios principales son los profesores y alumnos, que no fueron consultados...
- Negociar el equilibrio requisitos-presupuesto-planificación: tarea del director del proyecto, que requiere habilidades de gestión, personales, negociadoras y políticas
- El cliente confía en el ingeniero de requisitos para aclarar sus necesidades (como en un arquitecto para definir la casa que quiere construir)
- Trabajo conjunto para determinar los deseos del cliente: idea general, estudio del problema, descubrir matices, tomar decisiones clave

El punto de vista del cliente

- El gran desafío es *averiguar y expresar claramente* lo que los clientes necesitan
- A veces bastan las *palabras*, pero otras veces las *figuras* o tablas son de gran ayuda
- El cliente suele tener una idea vaga, inconsciente e incompleta de lo que espera de su aplicación (*punto de vista del cliente*)
- Diferentes personas pueden concebir de modo muy distinto lo que implica un sistema
- Ejemplo: una "aplicación meteorológica" puede significar:
  - Una utilidad para presentar gráficamente información recibida en bruto del servicio meteorológico
  - Un sistema en tiempo real para predecir el tiempo
  - Una aplicación para alertar a los usuarios de anomalías climáticas

Cómo llevar adelante una entrevista

- La entrevista es un recurso caro:
  - consume un tiempo significativo de más de una persona
  - → requiere una planificación cuidadosa
- Es importante escuchar con atención, pero no basta con escuchar pasivamente
  - Hay que saber preguntar, el cliente necesita ayuda
  - Ganarse la confianza del cliente
  - El resultado es producto del trabajo conjunto
- Escribir los requisitos y validarlos por escrito – continuar las reuniones hasta que se alcance el acuerdo
- Antes de la entrevista
  - Enumerar y priorizar los entrevistados
  - Planificar la hora de comienzo y finalización de cada entrevista
  - Enumerar los puntos que es necesario tratar en la entrevista
- Durante la entrevista
  - Asistir al menos dos personas del equipo de desarrollo: es preferible que haya dos entrevistadores, uno sólo tiene a descuidar algunos puntos
  - Usar cinta grabadora (pedir permiso)
  - Concentrarse y escuchar
  - No sea pasivo: preguntar y animar
  - Insistir hasta entender los deseos y las necesidades
  - Utilizar diagramas (si son útiles y según la formación de los entrevistados)
  - Tomar notas detalladas
  - Concretar la siguiente entrevista
- Después de la entrevista
  - Redactar el borrador de requisitos
  - Revisar entre los dos entrevistadores
  - Enviar a los clientes para comentar y aprobar

Técnicas para la *obtención* y *descripción* de requisitos del usuario

- Textuales: relativamente accesibles a un cliente sin formación específica
  - Texto en prosa común y corriente, tablas, etc.
  - Texto estructurado, lenguaje técnico
  - Casos de uso
- Gráficas: requieren un cierto grado de formación técnica en el cliente; tienen el peligro de convertir el análisis de requisitos en diseño
  - Diagramas de flujo de datos
  - Diagramas de estados
  - Diagramas de actividad
- Prototipos: no confundir con diseño, valorar la inversión
  - Interfaces de usuario
  - Prototipado rápido

Casos de uso

- Describir los *requisitos* como una *interacción* entre la aplicación y un agente externo
- Expresan el *punto de vista del usuario* de cómo debe funcionar la aplicación
- Más adecuados para obtener requisitos funcionales que no-funcionales

Diagramas de flujo de datos

- Para requisitos que se describan de modo *natural* como un flujo de datos (flechas) entre elementos de procesamiento (nodos)
- La utilidad de los DFD's depende del tipo de aplicación

Diagramas de estados

- Dividir la aplicación en estados de modo que siempre se encuentre exactamente en uno de ellos
- Útiles para *aplicaciones dirigidas por eventos*

Interfaces de usuario

- Diseño de la interfaz de usuario: ¿requisitos o diseño?
- Visualizar la GUI ayuda a los clientes a concebir y describir la aplicación
- Al ver los bocetos, el cliente se da cuenta de que *necesita más o quiere algo diferente*
- Tarea de un profesional especializado (especialmente el diseño detallado)
- Once pasos para desarrollar la GUI (ver Braude, pp. 151-157)

Prototipos

- Prototipado rápido
  - Implementación parcial, con un componente GUI importante
  - Muy útil para elicitar requisitos del cliente y para identificar y eliminar partes arriesgadas de un proyecto
  - Una comprensión más profunda ahorra trabajos futuros de corrección
- El beneficio del prototipo depende de su coste y de su valor para el proyecto
- ¿Debe transformarse el prototipo en la aplicación? Decisión planificada, no accidental, por que, en caso contrario, puede darse un empeoramiento de calidad
  - contruidos rápidamente, mal documentados (no será necesario mantenerlos)…
  - implementados en lenguajes que rápidamente obtienen resultados, pero tal vez inadecuados para la aplicación final: seguridad, fiabilidad...
  - desventaja: el cliente puede impresionarse y pensar que estamos cerca del final
  - ¿transformarías un prototipo de casa con balas de paja en la casa final?
- Beneficios: extraer requisitos, ensayar soluciones

Los Requisitos del Usuario en el Estándar ESA

- Lo que dice cada uno de los documentos
  - ESA software engineering standards (PSS-05-0)
  - Guide to applying the ESA SE-Std to projects using OO Methods (BSSC98-1)
  - Guide to the software requirements definition phase (PSS-05-03)
- Nociones importantes
  - Alcance del software: objetivo general que se persigue con la aplicación
  - Entorno operacional: contexto en el que debe operar el software
  - Requisitos de *capacidad*: funciones y operaciones requeridas por los usuarios para resolver un problema o alcanzar un objetivo; describe una operación, o secuencia de operaciones, que el software debe ser capaz de realizar
  - Requisitos de *restricción*: restricciones impuestas por los usuarios sobre la manera en que el problema es resuelto o el objetivo es alcanzado; restringe la manera en que el software es construido o funciona, sin alterar o describir las capacidades del software
  - Propiedades de cada requisito: identificador, descripción, necesidad (esencial, conveniente, opcional), estabilidad, fuente (personas, grupos, documentos...)
- Contenido del documento
- Importante: definir bien el alcance del software, el entorno operacional, los tipos de usuarios, etc. (esto marca una diferencia importante con los requisitos del software)

5. Modelado de casos de uso

Introducción

- Propósito: *describir las interacciones* entre la aplicación y los agentes externos, con el fin de *obtener los requisitos* de la aplicación.
  - Normalmente, las personas encuentran más fácil dar *ejemplos de la vida real* que *descripciones abstractas*. Pueden comprender y criticar un escenario de cómo podrían interactuar con un sistema software.
  - Los ingenieros de requisitos pueden utilizar la información obtenida de esta discusión para formular los requisitos reales del sistema.
  - Los *escenarios* son descripciones de ejemplos de sesiones de interacción.
  - A pesar de enorme número de *ejecuciones potenciales*, la mayoría de las aplicaciones se conciben en términos de un número relativamente pequeño de *interacciones típicas*. La descripción de interacciones cuasi-lineales, o usos típicos, ayuda a entender los requisitos funcionales del sistema, aunque el sistema final, con toda seguridad, no funcionará de forma cuasi-lineal.
- Definición: un caso de uso es la descripción de un *uso típico* del sistema o aplicación, que proporciona un *resultado valioso* para el usuario.
  - Alternativa: un caso de uso es la especificación de una funcionalidad o *servicio* ofrecido por la aplicación, *descrito en forma de interacción* usuario-sistema

Casos de uso y extracción de requisitos

- Los casos de uso expresan el *punto de vista del usuario* sobre cómo debe funcionar la aplicación: el cliente explica de manera sencilla lo que espera del sistema, por medio de la descripción de una interacción con el mismo, incluyendo la información suministrada, los cambios observables en el sistema, y la respuesta esperada.
- En esta descripción se revela cómo piensa el usuario o cliente acerca del sistema, y cuáles son los conceptos fundamentales del dominio.
- Por su forma de "historia" son útiles para comunicarse con los clientes (y *describir* o *descubrir* requisitos): proporcionan una vista muy reveladora de la aplicación
  - Los conceptos empleados sirven para descubrir las clases de análisis
  - La descripción de historias (escenarios) es útil para *ilustrar* usos típicos, pero es bastante limitada para *especificar* requisitos de modo completo
- Carácter hipotético de los casos de uso
  - Ejemplo: agencia de viajes por internet.
  - Formular hipótesis: patrón de comportamiento, objetivo de la interacción
  - Contrastar estas hipótesis con el cliente
- El requisito no es la interacción, sino el objetivo
  - Lo que el usuario realmente requiere del sistema (el verdadero requisito que hay que extraer) no es la interacción, sino el resultado observable, u objetivo
  - Describir la *historia* debe ser un medio para llegar a determinar el *objetivo*
  - La historia sirve para *entender* el requisito, pero no para *especificarlo*
- Más adecuados para obtener requisitos funcionales que no-funcionales
- Los casos de uso son una buena forma de *descubrir y organizar* requisitos de usuario
- Pero no son exactamente lo mismo: el requisito es el objetivo, no la interacción
  - El objetivo del caso de uso es un requisito de usuario
  - Los escenarios o interacciones no son requisitos
- Hay requisitos de usuario que no son (objetivos de) casos de uso, pero normalmente puede establecerse una relación de subordinación entre ellos (muchos a muchos)

El modelo de casos de uso

- Ejemplo: feria de subastas
- Diagrama de casos de uso
- Actores: no identificar los actores con "categorías de usuarios"
- Casos de uso
- Actores cooperativos: aplicación a la feria de subastas
- Include y Extend: aunque mucha gente las usa, su significado está poco claro

Especificación textual de casos de uso

- Nombre, actores y objetivo
- Escenario básico, o escenario principal con éxito (*main success scenario*)
    - Frases simples en la medida de lo posible
- Escenarios alternativos
- Pre- y post- condiciones
- Dos niveles de detalle según las necesidades
    - Requisitos del usuario: nombre, actores, objetivo y escenario básico
    - Requisitos del software: escenarios alternativos, pre- y post- condiciones, etc.

Casos de uso y operaciones del sistema

- No confundir:
    - las operaciones del sistema son las que responden a los *eventos externos*
    - un caso de uso es un *uso coordinado de operaciones del sistema*
- Típicamente, un caso de uso contiene más de una operación del sistema (diálogo)
- Ejemplo: acceso al sistema (*login*), ¿operación del sistema o caso de uso?
    - ¿cuál es el objetivo del usuario?
- Los pasos de un escenario se pueden agrupar en *bloques de acciones* que corresponden a operaciones del sistema (petición-validación-cambio de estado-respuesta)
- Las operaciones del sistema se pueden especificar con contratos más detallados

Especificación gráfica de casos de uso

- Diagramas de actividad

Casos de uso y procesos de negocio

- Un proceso de negocio (*business process*) es una secuencia de acciones conjuntas en las que participan uno o más agentes, de acuerdo con ciertas reglas de negocio (*business rules*)
- Los agentes pueden ser una o varias personas, y uno o varios sistemas informáticos
    - Entre todos ellos cooperan para sacar adelante el proceso
    - Cada uno de ellos tiene un rol distinto en el proceso, su propia responsabilidad
- Ejemplo: comprar una casa con hipoteca
    - Localizar la casa (vendedor, comprador, web inmobiliaria)
    - Obtener una hipoteca (comprador, banco, sistema informático del banco)
    - Formalizar la compra (vendedor, comprador, representante del banco, sistema informático del banco, notario, sistema informático del registro de la propiedad)
- Un caso de uso es una forma de describir la contribución (el rol, la responsabilidad) de un sistema a una acción conjunta (*proyección de la acción conjunta sobre el sistema*); describe el comportamiento del sistema en cada parte del proceso de negocio
- Aplicación al ejemplo: identificar los casos de uso de la web inmobiliaria, del sistema informático del banco, y del sistema informático del registro de la propiedad

- Consecuencia: el caso de uso sólo describe acciones del sistema
  - *Entradas de información* desde los actores (parecen acciones de los actores)
  - *Salidas de información* hacia los actores
  - *Acciones internas* cuyos efectos pueden ser observados o inferidos por los actores (qué, no cómo – sin detalles innecesarios)
  - Prohibido: acciones internas de otros actores o sistemas, comunicación de los actores con otros sistemas, etc.
- En cada caso de uso pueden participar uno o varios actores (y alguno puede representar en realidad otro sistema informático)

6. Requisitos del software: tipos de requisitos

Qué son los Requisitos del Software

- Es el único lugar donde se pone por escrito la naturaleza exacta de la aplicación
  - Se elaboran a partir de los requisitos del usuario
  - Son la base para el diseño y la implementación
- Diversas denominaciones: del software, del desarrollador, detallados, específicos...
- El nivel de detalle debe ser completo, pero no redundante
  - Lista completa de propiedades específicas y funcionalidad de la aplicación
  - A cada requisito se le sigue la pista hasta la implementación
- Diferencia con los requisitos del usuario: punto de vista del cliente, del desarrollador
- Audiencia primaria, el desarrollador. El cliente puede estar interesado en ellos e incluso entenderlos y hacer observaciones
- Anéctoda: en 1999 la NASA perdió un satélite porque se asumió que ciertos datos de control estaban en unidades métricas en lugar de anglosajonas. El defecto se identificó pocos días después del desastre... (podía haberse identificado durante el análisis)
- No es una tarea estúpida: organizar *todos* los requisitos bien detallados es una tarea difícil que requiere saber organizar personas y documentación

Clasificación de requisitos del software

- *Requisitos inversos*: lo que la aplicación no debe hacer (son infinitos...)
  - Pueden aclarar posibles malentendidos, el alcance del sistema
  - Seleccionar aquellos que sirven para aclarar los verdaderos requisitos
  - Ejemplo: la aplicación no *asesorará* al usuario sobre...
  - ¿Dónde? Alcance del software (RU/RS). Anotación a un requisito concreto
- *Requisitos funcionales* (derivados de los requisitos de capacidad)
  - Especifican la funcionalidad o *servicios* que la aplicación debe proporcionar
    - Ejemplo: un tiempo de respuesta no es un requisito funcional
  - Acompañados de un *modelo de análisis* (Platform Independent Model - PIM)
    - *Modelo conceptual*: requisitos de información
    - *Modelo de casos de uso*: requisitos de operación
    - Modelos de comportamiento que sean necesarios
- *Requisitos no funcionales* (derivados de los requisitos de restricción)
  - Imponen *restricciones*: en el producto desarrollado, en el proceso de desarrollo
  - Características distintivas:
    - *cómo* debe realizar algo el software, NO *qué* debe realizar
    - cortan *transversalmente* a los requisitos funcionales
    - son *negociables* hasta cierto punto, dentro de límites aceptables
    - la *medida de satisfacción* (o verificación) no es todo/nada, sino gradual
  - A veces denominados *requisitos de calidad*
    - "La funcionalidad se presupone, la calidad satisface (o no)"
  - Ejemplos: consumo de recursos, rendimiento, fiabilidad y disponibilidad, seguridad, portabilidad, mantenibilidad, modificabilidad, reusabilidad, interfaces, amigabilidad, manejo de errores, uso de estándares, verificación...
  - Habitualmente peor analizados y documentados
    - Descritos de modo vago, informal, ambiguo (dificultan la verificación)
    - Repetidos en muchos lugares (transversalidad)
    - Peor contemplados en los modelos
- Separación de incumbencias (*separation of concerns*) en los requisitos
  - Definir por separado requisitos funcionales y no funcionales
  - Componer los requisitos F y NF mediante tablas de referencias cruzadas

Consumo de recursos (*Resource expenditure*)

- Especifican la cantidad de recursos que la aplicación requiere
- Ejemplos:
  - Uso de memoria (volátil / permanente; o primaria / secundaria)
  - Capacidad de tráfico, líneas de atención simultánea, etc.

Rendimiento (*Performance*)

- Especifican restricciones temporales que la aplicación debe cumplir
- Son críticas en los sistemas de tiempo real
  - Ejemplos: velocidad, tiempo de respuesta, etc.

Fiabilidad y disponibilidad (*Reliability and availability*)

- En estos dos factores se reconoce que las aplicaciones difícilmente son perfectas, pero sí se exige un nivel de calidad mínimo
- Fiabilidad: cuantifica los errores permisibles y su gravedad
  - Ejemplo: el sistema no sufrirá más de dos fallos de nivel uno al mes
- Disponibilidad: cuantifica el grado de disponibilidad de la aplicación para los usuarios
  - Ejemplo: la aplicación no estará disponible como máximo durante 10 horas en cualquier periodo de 30 días, y como máximo durante 2 horas seguidas

Manejo de errores (*Error handling*)

- Cómo debe responder la aplicación a *errores de su entorno*, o a *errores internos*
  - Ejemplo: cómo reaccionar ante un mensaje en formato no reconocido
- No se debe abusar del manejo de errores internos, que no deben existir (no debemos cubrir nuestros errores con un código interminable de manejo de errores)
  - Ejemplo: determinar lo que hay que hacer si una función es llamada con parámetros equivocados; esto sólo se debe hacer si es preferible manejar el error a la terminación de la aplicación

Requisitos de interfaz (*Interface requirements*)

- Describen el formato con el que la aplicación se comunica con su entorno
  - Ejemplo (*comunicación con usuarios*): El coste del envío se mostrará en la esquina inferior derecha
  - Ejemplo (*comunicación con otras aplicaciones*): Los pedidos se transmitirán en formato XML utilizando la plantilla DTD detallada en el anexo IV

Restricciones (*Constraints*)

- Describen límites o condiciones sobre cómo diseñar o implementar la aplicación
- Estos requisitos no deben suplantar el proceso de diseño, simplemente especifican condiciones impuestas en el proyecto por el cliente, el entorno, u otras circunstancias
  - Ejemplo (exactitud, precisión): las tarifas aplicadas se medirán en diezmilésimas de euro
  - Ejemplo (lenguaje, herramienta): se empleará el lenguaje Fortran, el gestor de base de datos SQLServer...
  - Ejemplo (arquitectura): se emplearán tecnologías de intranet y cliente-servidor
  - Ejemplo (estándares): los informes generados se ajustarán al estándar XX-123
  - Ejemplo (plataformas): la aplicación deberá funcionar sobre Windows 95

Seguridad (*Security / Safety*)

- *Security*: seguridad del sistema frente a amenazas externas (confidencialidad, integridad, disponibilidad, etc.)
- *Safety*: seguridad de las personas frente a fallos del sistema

7. Requisitos del software: propiedades y atributos

Propiedades deseables de los requisitos del software

- Propiedades que deben tener todos los requisitos para estar bien especificados
- Al *inspeccionar* los requisitos deben cuestionarse estas propiedades
  - Utilizar *cuestionarios de validación* para inspeccionar requisitos
- Dos tipos:
  - Propiedades *globales*: completitud, consistencia
  - Propiedades *individuales*: tamaño, claridad, comprobabilidad, condiciones de error, trazabilidad (funcionales, no funcionales)
- No confundir con los *atributos* de los requisitos: toman un valor distinto en cada caso
  - Necesidad, prioridad y riesgo

Tamaño

- Para manejar con mayor facilidad un requisito, deberá tener un *tamaño adecuado*:
  - ni tan grande que sea inmanejable
  - ni tan pequeño que no valga la pena seguirle la pista por separado
- Es posible aplicar los principios de *modularidad* y *anidamiento* a los requisitos
- Nivel de *granularidad*: la misma cantidad de información puede repartirse en un número grande/pequeño de requisitos (grano fino/grueso)
- Nivel de *detalle*: los requisitos contienen más detalles, globalmente más información

Completitud

- Significa que *no hay omisiones* que comprometan la integridad de los requisitos
  - No faltan requisitos (propiedad global)
  - No faltan detalles en la especificación de cada requisito (propiedad individual)
- Es una propiedad *difícil de determinar* (tan sólo podemos alcanzar una aproximación)
  - Contrastar con el cliente
  - Comparar con proyectos semejantes
- Buscar la *visión de conjunto*, detectar huecos o partes infra-especificadas
  - Una manera de comprobarlo: para cada requisito, comprobar que están presentes los demás requisitos relacionados
- La buena *organización* facilita la detección de faltas
  - Ejemplo: *organización por tipos de requisitos*
- Técnicas estadísticas para estimar el número de requisitos aún no descubiertos
  - Ritmo temporal de creación/modificación de requisitos
  - Ajustar la "nube de puntos" (tiempo-requisitos conocidos) a una curva monótona creciente acotada (¿hipérbola?)

Consistencia (o coherencia)

- Significa que *no hay contradicciones* entre requisitos (ni acoplamientos-redundancias)
- Contradicción ≠ Ambigüedad, pero las ambigüedades difultan detectar contradicciones
- Es más difícil de comprobar si el número de requisitos crece
- Una buena organización facilita la detección de contradicciones
- Ejemplo: *tabla de referencia cruzadas*, que además facilita la detección de requisitos afectados por la modificación de uno dado (distinta de la tabla de composición F-NF)
  - Conflicto: contradicción, no se pueden satisfacer simultáneamente
  - Acoplamiento: hablan de lo mismo (si cambia uno, puede afectar al otro)
  - Redundancia: dicen lo mismo (sobra uno de los dos)
  - Independencia
- En la versión final no puede haber Conflictos ni Redundancias, sí Acoplamientos

Claridad

- Significa que *no hay ambigüedad* en la especificación de cada requisito
- Utilizar un *vocabulario controlado*, y tabla de términos equivalentes (*sinónimos*)
- Para cualquier labor de escritura
    - Tenga siempre a mano *diccionarios* (normal, sinónimos, estilo, idiomas, corrector ortográfico y sintáctico)
    - Escribir, corregir, escribir, corregir… y hacerlo *entre varios* (uno escribe, otro corrige)
    - Respetar *normas* ortográficas, sintácticas, gramaticales, estilísticas… no es un capricho: lo que no está bien escrito no se entiende
    - Estructurar bien, proceder con orden, proporcionar las referencias necesarias
    - Sintetizar, resaltar ideas importantes, resaltar más lo menos obvio

Comprobabilidad

- Incluye dos tipos distintos de defectos que se desea *descubrir y eliminar*:
    - *Validación*: defectos de interpretación (*do the right thing*)
    - *Verificación*: defectos de implementación (*do the thing right*)
- Muchos defectos se pueden descubrir y eliminar mediante *pruebas*, pero salvo que se usen métodos formales, las pruebas no garantizan que *todos* los defectos desaparecen
- Atención: las pruebas de verificación no sirven como pruebas de validación
- Para validar y verificar un requisito es necesario que éste sea comprobable (*testable*). Un requisito no comprobable o ambiguo tiene escaso valor y debe ser rechazado.
- Ejemplos:
    - El sistema mostrará la diferencia entre el valor observado y la media mundial (*no comprobable*)
    - El sistema mostrará la diferencia entre el valor observado y el valor publicado por Naciones Unidas (¿cuándo? todavía es *ambiguo*)
    - El sistema mostrará la diferencia entre el valor observado y el último valor publicado por Naciones Unidas en su página web.
- Conviene asegurar que no es ambiguo, ni siquiera si se interpretase mal a propósito
- Esbozar una prueba específica que establecería la satisfacción
- La definición de pruebas ayuda a clarificar el sentido de cada requisito

Condiciones de error

- Para estar completa, la especificación del requisito debe tener en cuenta las condiciones de error, es decir, qué ocurre con el requisito en una situación con errores
- Las condiciones de error son especialmente importantes para realizar las pruebas, ya que al probar un requisito se fuerzan condiciones de error, y es necesario prever qué debe ocurrir en estos casos. Caso típico: datos de entrada incorrectos
    - *Ejemplo*: clasificar un triángulo a partir de las coordenadas de sus tres vértices
- No confiar en que no se producirán condiciones de error: esto aumenta la dependencia entre módulos (un módulo confía en la detección de errores de otro módulo)
    - Pero recordar que no se debe abusar del manejo de errores internos
- Redundancia para promover la seguridad
- Determinar lo que hay que hacer en situaciones no previstas (*prever lo imprevisto*)
- Prevenir posibles errores de programación en lugares críticos

Trazabilidad de requisitos funcionales

- Es la *correspondencia* entre cada requisito del software y...
  - uno o más requisitos del usuario, u otras fuentes (trazabilidad hacia atrás)
  - una o varias partes del diseño o la implementación (trazabilidad hacia adelante)
- La relación RU-RS es típicamente uno-a-pocos. Todo RU debe ser desarrollado por al menos un RS, pero puede haber un RS cuya fuente no sea un RU (PSS-05-03, 3 y 47):
  - No confundir con el caso de nuevos RU que se descubren durante el desarrollo de los RS, y que deberán ser validados por el usuario: nueva versión URD
  - Verdadero RS sin RU: todo RS debe tener una buena razón para existir, con mayor razón si no existe un RU correspondiente: el cliente no querrá pagar por cosas que no ha encargado
- Es necesaria para poder *comprobar* que la aplicación satisface los requisitos
- Una forma de conseguirla es relacionar cada requisito funcional con una función específica en el lenguaje destino
  - Esta técnica es poco generalizable y limitada; produce excesivo acoplamiento entre la estructura de los requisitos y la estructura de la implementación
  - En OO *no es trivial* preguntarse qué clase es responsable de qué función
    - Función con un solo parámetro: F(x) → x.F( )
    - Función con dos o más parámetros: G(x, y) → x.G(y) / y.G(x)
- La transición Análisis-Diseño no es sencilla ni tiene por qué serlo
- Es muy necesaria para afrontar que los requisitos cambian, y gestionar su *evolución*
  - Si la gestión de la trazabilidad es difícil y lleva demasiado trabajo, los desarrolladores tenderán a evitar la actualización del documento de requisitos e irán directamente a hacer cambios al código: en consecuencia el documento de requisitos se hace inútil para las pruebas y la validación
  - Si además el documento no es fiable por culpa de algunos miembros del equipo menos responsables, entonces ni siquiera los más responsables se tomarán la molestia de mantenerlo al día
  - Por tanto, el sistema para hacer corresponder requisitos con diseño y código debe ser claro y concreto
  - Ejemplo: *matrices de trazabilidad de requisitos*
    - hacia atrás / hacia delante
    - distintas de la tabla de referencias cruzadas entre requisitos, para gestionar la consistencia
- Un requisito puede corresponderse con varias partes del código (funciones), que a su vez pueden implementar varios requisitos cada una
  - Por tanto, antes de cambiar el código para adaptarse a un cambio en un requisito, hay que comprobar que otros requisitos no quedan comprometidos
  - La correspondencia Requisito-Función es muchos-a-muchos; si no puede ser *uno-uno*, al menos puede intentarse que sea *pocos-pocos*
  - El *diseño* juega un papel fundamental en establecer esta correspondencia

Trazabilidad de requisitos no funcionales

- La correspondencia con elementos del diseño y de la implementación es mucho más difícil, ya que depende en mayor medida de la *colaboración entre varios elementos*
- La *validación* de estos requisitos es más complicada, normalmente requiere un mayor nivel de integración del sistema (no es fácil validar NFRs en componentes aislados)
- Ejemplo: *rendimiento*.
  - Identificar los elementos que consumen más tiempo o memoria.
  - La mayor parte de los recursos son consumidos por un número reducido de elementos, de modo que esta tarea es provechosa para mejorar el rendimiento.
  - Bucles, gráficos y comunicaciones suelen ser los que más tiempo consumen.

ATRIBUTOS

Necesidad y prioridad

- Para negociar entre funcionalidad, planificación, presupuesto y nivel de calidad, es conveniente que los requisitos funcionales tengan asignado un nivel de *necesidad* (tres o cuatro categorías: *esencial, deseable, opcional*...), así como un nivel de *prioridad* temporal (dos o tres categorías: *alta, baja*...)
- La necesidad de un requisito hace referencia al *interés* de los usuarios/clientes en que la aplicación lo realice, y hasta qué punto estarían dispuestos a pasarse si él
- La prioridad de un requisito hace referencia al *orden temporal*: indica en qué fase de construcción del sistema se incluirá la funcionalidad que realice el requisito
- Si el 20% de la aplicación proporciona el 80% de su valor... no más del 20% de los requisitos deberían ser "esenciales"

Riesgo

- Cada requisito debería ir acompañado de una estimación del riesgo que supone realizarlo (relacionado con la dificultad de implementarlo). Ej.: *alto, moderado, bajo*
- Seguir la pista con especial atención a los de mayor riesgo

Para escribir un requisito: resumen

- Enunciado breve del mismo, numerado
- Explicación detallada
- Atributos y propiedades tal como se han discutido hasta aquí, por ejemplo
  - Necesidad, Prioridad y Riesgo
  - Condiciones de error
  - Pruebas de validación y verificación requeridas
- Tabla de composición de requisitos funcionales-no funcionales
- Tabla de referencias cruzadas entre requisitos: conflicto, acoplamiento, redundancia
- Matrices de trazabilidad hacia requisitos de usuario y hacia diseño/implementación
- Usar un formato que permita distintos *niveles de visibilidad*
  - sólo enunciado
  - sólo enunciado y descripción
  - enunciado, descripción y propiedades...