Apunte de clase

Introducción a la Ingeniería de Requerimientos

Maximiliano Cristiá

Licenciatura en Ciencias de la Computación

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Universidad Nacional de Rosario

Rosario – Argentina

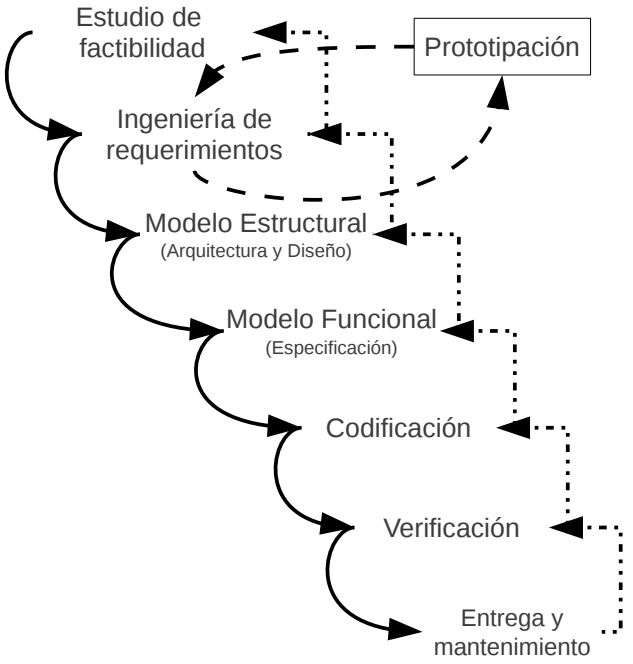
© Maximiliano Cristiá – 2018-2024 – Todos los derechos reservados

Índice

1. Ingeniería de Requerimientos

La Ingeniería de Requerimientos es la segunda fase estipulada en el ciclo de vida de cascada, como se muestra en la Figura 1. Según este modelo de ciclo de vida, es necesario contar con los requerimientos para poder definir el Modelo Estructural del sistema (es decir, la arquitectura y el diseño). Sin embargo, como analizaremos en la sección 1.2, usualmente es imposible contar con todos los requerimientos del sistema en un tiempo razonable, por lo que en la mayoría de los desarrollos se comienza a definir el Modelo Estructural teniendo en cuenta solo algunos requerimientos. Habiendo ubicado la Ingeniería de Requerimientos en el proceso de construcción de un sistema de software, definimos más formalmente esta fase.

Figura 1 Esquema del ciclo de vida de cascada. La Ingeniería de Requerimientos es la segunda fase.



Definición 1 (Ingeniería de Requerimientos)**.** *El proceso para establecer los servicios que el sistema debería proveer y las restricciones bajo las cuales debería operar y ser desarrollado, se llama Ingeniería de Requerimientos [Som10].*

La Ingeniería de Requerimientos es, por tanto, una actividad esencialmente de interacción con los interesados[1] en el sistema. Es incorrecto y extremadamente riesgoso que los Ingenieros de Software establezcan los requerimientos del sistema (a menos que la estrategia de la empresa sea forzar a los potenciales clientes a adecuarse al sistema tal cual está, como es el caso de los programas de uso masivo) sin haber mantenido numerosas reuniones con diferentes representantes del cliente, sin haberles mostrado prototipos del sistema, sin haberles hecho una y otra vez las mismas preguntas, sin haber comprendido el negocio del cliente.

[1] Interesados es nuestra traducción del término inglés *stakeholders*.

Pero, ¿qué es un requerimiento? Si bien no es vital dar una definición muy formal de este concepto, creemos que las siguientes pueden ayudar a comprender el tema.

Definición 2 (Requerimiento o requisito)**.** *Un requerimiento es una característica del sistema o una descripción de algo que el sistema es capaz de hacer con el objeto de satisfacer el propósito del sistema [Pfl01], lo que ha sido apropiadamente documentado y validado por el solicitante [BPKR09]. Los requerimientos tratan exclusivamente sobre los fenómenos del dominio de aplicación y no sobre la máquina que los implementa [Jac95].*

Es importante resaltar algunos puntos de la definición anterior:

- No es lo mismo un pedido o deseo de un usuario o cliente que un requerimiento. No todos los pedidos o deseos de un usuario o cliente se convierten necesariamente en requerimientos, pero sí todos los requerimientos se originan en un pedido o deseo de un usuario o cliente. Para que un pedido o deseo de un usuario o cliente se convierta en requerimiento, este debe ser documentado apropiadamente y el solicitante debe validarlo.
- Los ingenieros de software no originan los requerimientos; su función es convertir pedidos de los usuarios o clientes en requerimientos. Luego deben proveer un sistema que los implemente.
- Los requerimientos siempre están en el entorno del sistema que se va a desarrollar, nunca dentro de él. Por consiguiente palabras tales como tabla, XML, clase, subrutina, etc. no pueden formar parte de un requerimiento.

A continuación introducimos las dos grandes categorías de requerimientos.

Definición 3 (Requerimientos funcionales)**.** *Un requerimiento funcional describe una interacción entre el sistema y su ambiente. Los requerimientos funcionales describen cómo debe comportarse el sistema ante un estímulo [Pfl01].*

Algunos ejemplos de requerimientos funcionales son los siguientes:

- Sistema de control de ascensores.
  - Una persona que desee utilizar el ascensor para ir del piso n al $n+m$ (con $m > 0$) debe pulsar el botón *Arriba* en el piso n, en cuyo caso el ascensor eventualmente debería detenerse en dicho piso.
  - Se deben abrir las puertas del ascensor y deben permanecer así por un tiempo suficiente como para que el usuario pueda ingresar.
  - Las puertas nunca deben abrirse a menos que el ascensor esté detenido en un piso.
- Sistema de cajas de ahorro de un banco.
  - Cada caja de ahorro se identifica por un número de cuenta.
  - Cualquier persona puede efectuar un depósito de una cantidad positiva de dinero en cualquier caja de ahorro.
  - Solo el titular o los co-titulares de una caja de ahorro pueden extraer una cantidad positiva de dinero de ella.
- Editor de textos gráfico.
  - El usuario debe poder seleccionar un área de texto, lo que se representará gráficamente pintando el fondo del área con un color diferente al fondo que el área tenga en ese momento.

    - El usuario debe poder poner en negrita cualquier cadena de caracteres de longitud mayor o igual a 1 tal que ninguna de sus subcadenas ya lo esté. Para lograr esto se deberán proveer los siguientes mecanismos:
        1. Si la cadena está seleccionada se pondrá en negrita todo su contenido.
        2. Si el cursor se encuentra entre dos letras de una palabra, se pondrá en negrita dicha palabra.
    - El usuario deberá poder ejecutar la función negrita desde un botón ubicado en una de las barras de herramientas, mediante la combinación de teclas que se le haya asignado a dicha función en ese momento y mediante la opción de menú Formato-Negrita.
- Sistema de facturación.
    - El usuario deberá poder seleccionar el tipo de factura a emitir.
    - Las facturas de tipo A requieren los siguientes datos obligatorios: razón social de quien emite la factura, CUIT de quien emite la factura, razón social de quien solicita la factura, CUIT de quien solicita la factura, fecha en la que se emite la factura, número de la factura.
    - La razón social de quien emite la factura será incluida automáticamente.
    - El CUIT de quien emite la factura será incluido automáticamente.
    - El sistema deberá buscar, en la lista de clientes de la empresa, y completar automáticamente la razón social del destinatario del sistema a medida que el usuario la tipea.
    - Una vez ingresada la razón social del solicitante, si corresponde a un cliente de la lista de clientes de la empresa, el sistema completará automáticamente el CUIT del solicitante.

Los requerimientos funcionales incluyen [IEE98]:

1. Chequeos de validación de la entrada
2. Secuencia exacta de las operaciones
3. Respuestas a situaciones anormales, incluyendo:
    a) Desborde
    b) Comunicaciones
    c) Manejo de errores y recuperación
4. Efecto de los parámetros del sistema
5. Relación entre las salidas y las entradas, incluyendo:
    a) Secuencias de entrada/salida
    b) Fórmulas para convertir entrada en salida

Si bien los requerimientos funcionales, o la función o funcionalidad del sistema, son esenciales para poder construir el sistema correcto, existen ciertas cualidades o atributos que los usuarios esperan del sistema que no tienen una relación simple con la funcionalidad que desean. A estas cualidades o atributos se los llama *requerimientos no funcionales*.

Definición 4 (Requerimientos no funcionales o atributos de calidad o cualidades del sistema)**.** *Un requerimiento no funcional es una restricción sobre el sistema o su proceso de producción [Som10, Pfl01].*

Algunos ejemplos de requerimientos funcionales son los siguientes:

- El sistema debe ejecutar sobre Linux Ubuntu 10.4 LTS 10.04 (Lucid Lynx) y Windows 7.
- El sistema debe impedir que usuarios no autorizados accedan a información crítica.
- El sistema debe estar en funcionamiento 24/7.
- El sistema debe poder procesar 100.000 transacciones por hora.
- El sistema debe ser implementado en Java.

Lamentablemente, es un problema recurrente en el desarrollo de sistemas de software que los requerimientos funcionales no solo ocupen la primera butaca sino la única en el proceso de desarrollo. Esto es la causa de innumerables modificaciones *extensivas* puesto que la mayor parte de las veces los sistemas de software no se modifican porque sean funcionalmente incorrectos, sino porque son difíciles de mantener, portar, escalar, o son muy lentos o vulnerables. Claramente, un sistema inseguro no se torna seguro modificando dos o tres funciones, normalmente es necesario introducir muchos cambios más o menos importantes en diferentes partes del sistema.

Conviene mencionar que no siempre es simple determinar si un requerimiento es funcional o no funcional.

1.1. Relación de los requerimientos con otros documentos

¿Qué relación tienen los requerimientos con el resto de las descripciones técnicas o actividades del proyecto? En particular, ¿cuál es la relación entre los requerimientos, el diseño, la especificación, el programa y el testing?[2] Como muestra la Figura 2 el diseño y la especificación del sistema se obtienen a partir de los requerimientos. A su vez, el diseño y la especificación se utilizan para implementar el programa. Finalmente, los requerimientos son el documento esencial para llevar adelante el testing de aceptación. De estas relaciones surge claramente que la calidad de los requerimientos tiene una influencia enorme sobre el éxito o fracaso del proyecto, puesto que de ellos dependen los documentos claves del desarrollo. En consecuencia, cuanto mayor calidad tengan los requerimientos mayores son las posibilidades de que el proyecto sea un éxito. De esta observación podría concluirse que cuanto más tiempo y recursos se dediquen a esta etapa, mejor será para el proyecto. Sin embargo, como veremos en la siguiente sección existe un límite práctico que se debe tener en cuenta cuando se planifica el proyecto.

1.2. El límite de la Ingeniería de Requerimientos

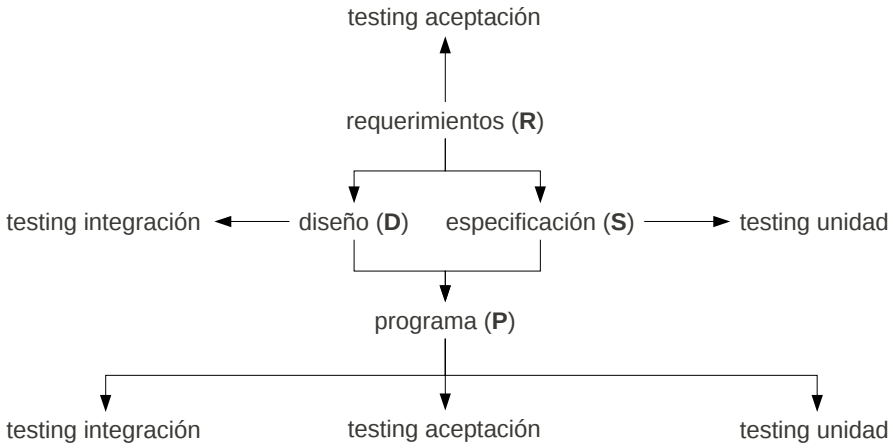
Existe un límite fundamental, y sobre todo inherente a la construcción de software, que afecta a la Ingeniería de Requerimientos. Este límite puede expresarse de la siguiente forma: es virtualmente imposible iniciar la construcción de un sistema de software teniendo la lista completa y consistente de todos los requerimientos, en un tiempo razonable.

Entre las razones que justifican esta hipótesis tenemos las siguientes:

- Usualmente se requiere que los grandes sistemas de software constituyan una superación del estado del arte en cierto dominio de aplicación.

[2] Entendemos por diseño al documento técnico que describe los elementos de software en que se ha dividido el sistema, las relaciones entre ellos y la función de cada uno de ellos.

Figura 2 Relación entre requerimientos, diseño, especificación, programa y testing.



- Los grandes sistemas normalmente son utilizados por una comunidad de usuarios muy diversos. Esto implica que cada grupo de usuarios planteará expectativas y requerimientos diferentes sobre el sistema.
- En general la gente que paga por el sistema y quienes lo usan no son los mismos, lo que genera conflictos entre lo que se puede pagar y lo que se necesita.
- La introducción de un sistema de software importante en una organización es tan disruptiva que suele producir cambios igualmente importantes en la organización lo que, a su vez, genera nuevos requerimientos sobre el sistema.
- El entorno en el cual se instala y opera el sistema cambia por lo que el sistema debe cambiar también.
- El negocio de la empresa productora del software suele cambiar por lo que sus ejecutivos solicitan nuevas funciones o cualidades al sistema.

En nuestra opinión este es un límite real y propio de la naturaleza del software y de la tecnología informática en general. Es tan irreal pensar que existe alguna forma de tener todos los requerimientos en un tiempo razonable, como peligroso no pensar en los requerimientos futuros. Esta es una regla del juego que juegan los Ingenieros de Software y por lo tanto hay que crear y aplicar principios, metodologías, técnicas y herramientas para ganar el juego sin ser penalizados por esa regla.

Los principios y metodologías que permiten minimizar los efectos nocivos de este límite los estudiaremos en las secciones 1.8 y 2.1, respectivamente. Las técnicas que aplicaremos son las siguientes:

1. Prototipación rápida desechable,
2. Escenarios de atributos de calidad, y
3. Diseño basado en ocultación de información.

La primera la estudiaremos en la sección 2 y la segunda en la sección 3, en tanto que la tercera queda fuera del alcance de este curso.

1.3. La importancia de los requerimientos

Si bien la importancia de los requerimientos es bastante obvia creemos que es conveniente resaltar algunas cuestiones específicas por lo que reproducimos aquí, casi textualmente, el cuadro destacado 4.1 de [Pfl01].

En 1994 el Standish Group realizó un estudio sobre 8.000 proyectos de software con el objetivo de determinar el grado de éxito de cada uno de ellos. Los resultados fueron bastante desalentadores: el 31% de los proyectos fueron cancelados antes de ser terminados, en las grandes compañías solo el 9% de los proyectos fueron entregados dentro del plazo y presupuesto predefinidos y el 16% en las pequeñas y medianas. A raíz de estos descubrimientos, un año más tarde Standish Group realizó otro estudio para determinar las causas de semejante fracaso. Los resultados fueron los siguientes:

- Requerimientos incompletos (13,1%)
- Falta de compromiso del usuario (12,4%)
- Falta de recursos (10,6%)
- Expectativas no realistas (9,9%)
- Falta de soporte ejecutivo (9,3%)
- Requerimientos y especificaciones cambiantes (8,7%)
- Falta de planeamiento (8,1%)
- Fin de la necesidad del sistema (7,5%)

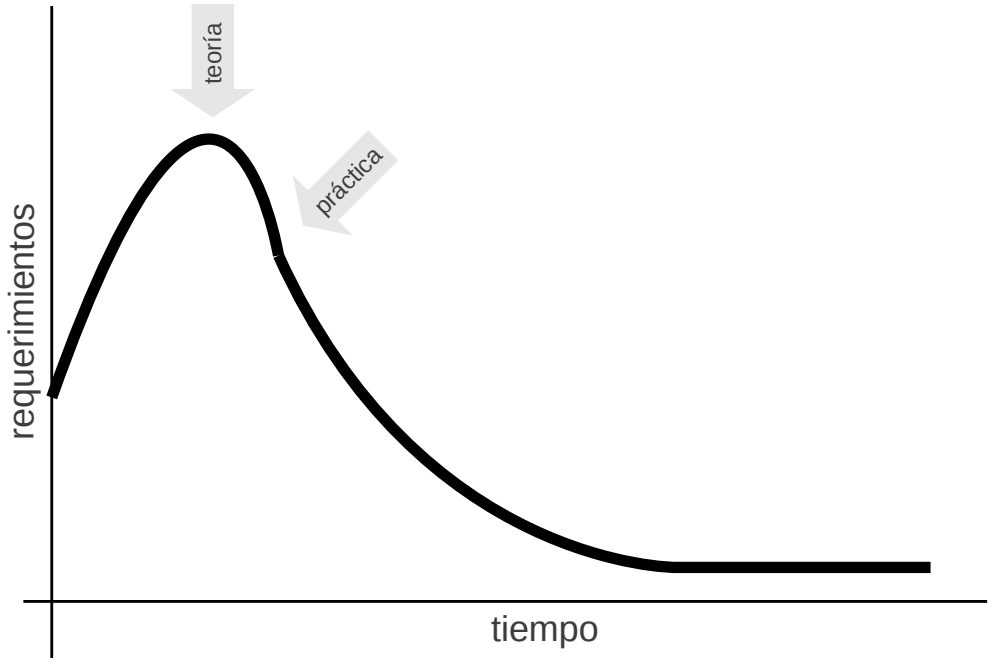
Notar que la Ingeniería de Requerimientos participa en varias de las causas mencionadas. Por otro lado, Boehm y Papaccio [BP88] calcularon que si cuesta $1 localizar y subsanar un error durante la Ingeniería de Requerimientos, puede costar $5 repararlo durante el diseño, $10 durante la programación, $20 durante el testing de unidad y hasta $200 después de la entrega del sistema. En consecuencia, podemos "gastar" bastante más que lo habitual durante la Ingeniería de Requerimientos porque de lo contrario es muy probable que gastemos mucho más en las fases siguientes.

Finalmente, el documento de requerimientos (ver sección 1.4) debería ser la base sobre la cual se planifica y cotiza el desarrollo del sistema. Sin contar con una primera versión del documento de requerimientos, cualquier cálculo de costos y tiempo es como mínimo temerario.

1.4. El proceso y los productos de la Ingeniería de Requerimientos

La Figura 4 muestra las principales etapas y productos de la Ingeniería de Requerimientos. El proceso comienza por la obtención de requerimientos y finaliza con la redacción del documento de requerimientos. En todos los casos el término "requerimientos" refiere tanto a requerimientos funcionales como no funcionales. Tal vez la etapa más importante de todo el proceso es la validación de requerimientos pues, cuando se la lleva a cabo correctamente, permite ahorrar mucho dinero a la empresa desarrolladora y aumenta la confianza del cliente en el éxito del proyecto. La figura muestra claramente que la validación debe realizarse durante todo el proceso de la Ingeniería de Requerimientos. Por otro lado, observar que las etapas se suelen ejecutar de forma circular e iterativa, siendo imposible en la mayoría de los casos pasar de una a la otra sin jamás volver hacia atrás.

Figura 3 Curva típica que representa la aparición de requerimientos con respecto al tiempo en el período que media hasta la primera entrega.



Según el modelo de desarrollo de cascada (Figura 1) a la fase de Ingeniería de Requerimientos le sigue la de Arquitectura y Diseño. Cabe preguntarse, entonces, en qué momento de la ejecución de la primera es conveniente iniciar la segunda. Se podría iniciar la fase de Arquitectura y Diseño (o la etapa de **Definición de requerimientos**) cuando se desacelera la cantidad de requerimientos *validados* que pide el cliente. Esto ocurre cuando cuando el cociente entre la aparición[3] de requerimientos y el tiempo es bajo durante un período de tiempo razonable. Podemos expresarlo matemáticamente de la siguiente forma:

$$\lim_{\Delta t \to 0} \frac{\Delta R(t)}{\Delta t} = 0$$

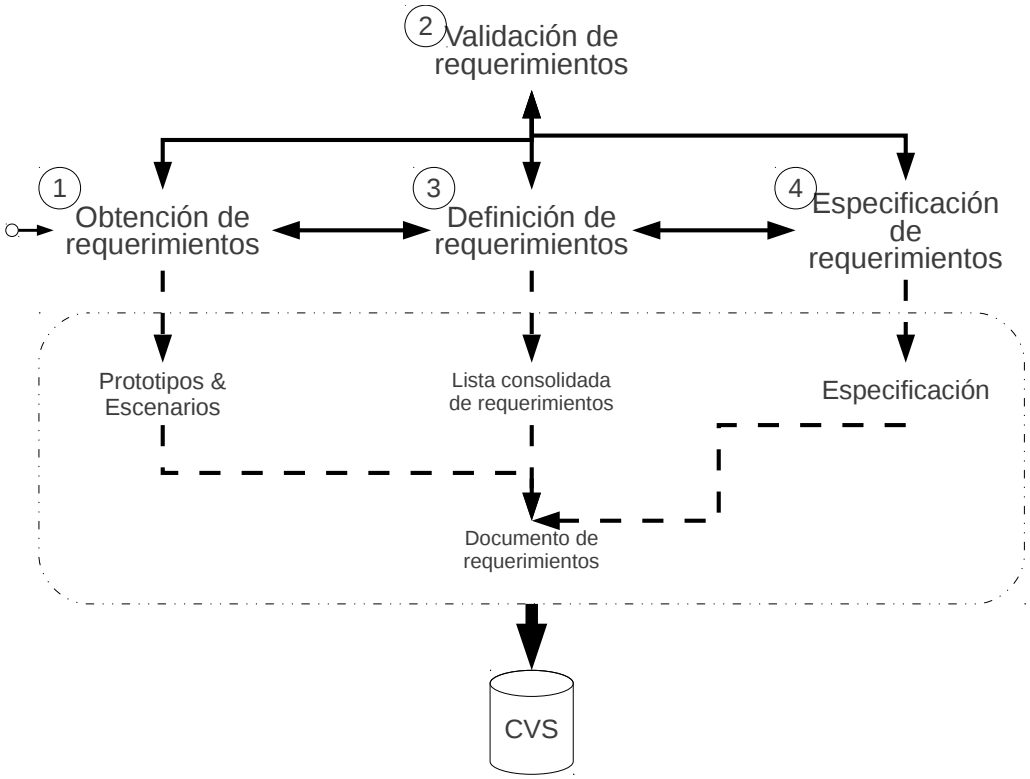
donde R es la función que cuenta la cantidad requerimientos *validados* en cada instante de tiempo. La Figura 3 muestra gráficamente los cambios de fase teórico y práctico de acuerdo a la fórmula propuesta. El cambio de fase práctico difiere del teórico en tanto los ingenieros no tienen forma de saber que se ha alcanzado el pico de aparición de requerimientos hasta que ha pasado un lapso razonable de tiempo durante el cual la cantidad de requerimientos por unidad de tiempo se ha ido reduciendo.

A continuación se describen muy brevemente cada una de las etapas del proceso.

Obtención de requerimientos. En esta etapa se obtienen los requerimientos del sistema a través de la observación de sistemas existentes y del entorno donde se instalará el sistema, reuniones con los interesados, generación de prototipos, definición de escenarios, etc. Estas técnicas deben aplicarse una y otra vez hasta tanto finalice la etapa.

[3] Modificación, inclusión y/o eliminación.

Figura 4 Proceso y productos de la Ingeniería de Requerimientos. Los números indican la secuencia más común entre las etapas.



Es crucial que antes de comenzar esta etapa el cliente tenga claro lo que se denomina *el negocio detrás del sistema*[4]. Si el negocio no está claro es muy probable que la construcción del sistema sea un fracaso más que nada para el cliente (lo que en muchos casos termina por afectar al desarrollador). Por lo tanto, es una responsabilidad de los Ingenieros de Software lograr que el cliente pueda definir con precisión el negocio detrás del sistema. Veremos más sobre esto en la sección 1.6.

Resulta igualmente importante que los ingenieros de requerimientos tengan un conocimiento del dominio de aplicación suficiente como para poder comprender el vocabulario del cliente, caso contrario el proceso de Ingeniería de Requerimientos puede tornarse largo, costoso y el cliente puede perder confianza en la empresa desarrolladora.

La tercera condición imprescindible para el éxito de esta etapa es que los ingenieros de requerimientos determinen con la mayor precisión posible la lista de interesados en el sistema. Todos los interesado participarán de esta etapa y/o de la etapa llamada **Validación de requerimientos**. Esta lista de interesados puede ir creciendo a medida que la empresa desarrolladora va conociendo a su cliente y el proyecto, y en particular la Ingeniería de Requerimientos, se va desarrollando. Es muy difícil determinar a priori la lista exhaustiva de interesados por lo que no debe sorprendernos que la vayamos ampliando a medida que el proyecto avanza.

Validación de requerimientos. Los requerimientos solo pueden ser validados por el cliente. Por lo tanto, una vez que se ha obtenido un requerimiento (preguntando al cliente) se le debe preguntar si ese es el requerimiento que expresó. Esto no necesariamente es o da lugar a una tautología. El "cliente" no es una sola persona, como veremos en la sección 1.5, ni lo que un ingeniero de software escribe es exactamente lo que el "cliente" dijo. Por lo tanto, la validación de requerimientos trata esencialmente de preguntar a un representante del cliente si lo que un ingeniero de requerimientos puso por escrito, a instancias de otro o el mismo representante del cliente, es correcto o no.

En este contexto poner por escrito tiene varios significados: desarrollar un prototipo, transcribir un requerimiento expresado verbalmente, describir un escenario de un atributo de calidad, etc.

Veremos más sobre validación de requerimientos en las secciones 2 y 3.

Definición de requerimientos. En esta etapa la información colectada en la etapa anterior se vuelca en un documento consolidado, organizado y estructurado. En efecto, como durante la **Obtención de requerimientos** participan varios ingenieros, numerosos representantes del cliente y se aplican diversas técnicas (en particular se generan varios prototipos y escenarios) la información recabada tiende a estar dispersa y desorganizada, por lo que es necesario consolidarla y organizarla.

Durante esta etapa se prioriza cada uno de los requerimientos apoyándose en el negocio detrás del sistema (sección 1.6) según la siguiente escala [Pfl01]:

1. Requerimientos que deben ser absolutamente satisfechos;
2. Requerimientos que son muy deseables pero no indispensables; y
3. Requerimientos que son posibles, pero que podrían eliminarse.

[4]Traducción de *the business case for the system*, concepto tomado de [Rob04].

Atributo	Comentario
Cohesivo	El requerimiento habla de una única cosa.
Completo	El requerimiento está enunciado en un único lugar y no falta información.
Consistente	El requerimiento no contradice ningún otro requerimiento y es consistente con cualquier otra información relevante.
Correcto	El requerimiento apunta a soportar el negocio detrás del sistema como fue enunciado por el cliente.
Actualizado	El requerimiento no se volvió obsoleto por el paso del tiempo.
Observable externamente	El requerimiento habla de una característica del sistema que es observable o experimentada por el usuario. Los "requerimientos" que hablan de la arquitectura, diseño o implementación no son, en realidad, requerimientos.
Factible	El requerimiento puede implementarse dentro de los límites del proyecto.
No ambiguo	El requerimiento está enunciado de forma clara y concisa utilizando el vocabulario del dominio de aplicación, habiéndose definido con precisión todos los términos técnicos y siglas. El enunciado tiene una única interpretación. Sustantivos, adjetivos preposiciones, verbos y frases subjetivas que resulten vagas o poco claras deben ser evitadas. Se prohíbe el uso de oraciones negativas o compuestas.
Validable	La implementación del requerimiento puede validarse a través de inspección, análisis, demostración o prueba (test).

Tabla 1: Atributos que ayudan a determinar la calidad de un requerimiento. Fuente: `http://en.wikipedia.org/wiki/Requirement`.

Además, se establece la *calidad* de cada uno de los requerimientos. La calidad de un requerimiento refiere en general a qué tan perdurable en el tiempo será tal y como está descripto en este momento. Por ejemplo, un requerimiento es de mala calidad si el cliente no puede expresarlo claramente o directamente duda o desconoce qué es lo que realmente necesita o quiere. La calidad de un requerimiento está directamente relacionada con el grado de validación que aun necesita. Los atributos listados en la Tabla 1 permiten determinar más objetivamente la calidad de cada requerimiento la cual debe resumirse según la siguiente escala:

1. El requerimiento no necesita más validación;
2. El requerimiento debe ser validado por al menos un interesado más; y
3. El requerimiento debe ser validado por todos los interesados.

Especificación de requerimientos. En pocas palabras en esta etapa la **Lista consolidada de requerimientos** se escribe desde la perspectiva del desarrollador. Esta etapa puede incluirse en la fases denominadas **Modelo Estructural** o **Modelo Funcional** en el modelo de ciclo

de vida de cascada (Figura 1) como veremos en capítulos posteriores. En consecuencia no profundizaremos aquí sobre esta etapa.

Los productos del proceso de la Ingeniería de Requerimientos son los siguientes:

Prototipos. Nunca se enfatizará lo suficiente la importancia crucial de usar inteligentemente prototipos para obtener los requerimientos funcionales. En nuestra opinión debe ser la forma de interacción casi excluyente entre los ingenieros de requerimientos y el cliente, durante esta fase. Nos referimos concretamente a *prototipos desechables*[5]. Un prototipo desechable es un programa que imita al menos algunos de los requerimientos funcionales pero que luego de que los requerimientos se han validado, el programa se descarta. Trataremos en profundidad y con mucho detalle este tema en la sección 2.

Escenarios. Es muy complejo o costoso, y muchas veces imposible, prototipar los atributos de calidad de un sistema (tales como disponibilidad, seguridad, testeabilidad, etc.). En consecuencia, en lugar de prototipos, se utilizan *escenarios de atributos de calidad* [**?**] como técnica efectiva para determinar con suficiente precisión cuál es el significado que el usuario le asigna a las cualidades del sistema. Veremos más sobre escenarios de atributos de calidad en la sección 3.2.

Lista consolidada de requerimientos. Es un documento que consolida, organiza y estructura la información recolectada durante la etapa de **Obtención de requerimientos**.

El vocabulario y las notaciones que se utilicen para escribir esta lista deben ser estrictamente las del cliente o las del dominio de aplicación (ver más en sección 1.7): esencialmente es un documento *para* el cliente. Es decir, en general, este documento no debería incluir términos tales como: tabla, registro, campo, lista enlazada, función, proceso o procedimiento (en su acepción informática), variable, base de datos, archivo, directorio, sistema operativo, módulo, objeto, clase, hilo de control, servidor, HTTP, webservice, etc. Frases tales como "el sistema deberá implementar un módulo..." o "un webservice tomará los datos de la tabla `personas` y los enviará vía SOAP al servidor Web..." están totalmente fuera del alcance de este documento. Frases tales como "el sistema deberá implementarse en C++..." o "el sistema deberá procesar 10.000 transacciones por minuto..." pueden estar dentro del alcance de este documento, en tanto requerimientos no funcionales.

El documento se genera fundamentalmente a partir de los prototipos, escenarios de atributos de calidad y las entrevistas y observaciones sobre entorno del sistema que han efectuado los ingenieros de requerimientos.

Este documento debe incluir al menos los siguientes contenidos (para una descripción detallada de la estructura de este documento ver la Sección 5):

- El negocio detrás del sistema. Una descripción clara y lo más objetiva posible sobre el negocio detrás del sistema (ver sección 1.6).
- Descripción global de los requerimientos. En esta sección se describe el sistema que se debe construir en términos generales sin identificar requerimientos específicos.
- La lista estructurada de requerimientos. Es una lista dividida en secciones, subsecciones, etc. en cada una de las cuales se incluyen los requerimientos relacionados con una funcionalidad específica del sistema, en tanto sea posible dividirlos de esta

[5] Traducción de *throw-away prototypes*.

forma. El objetivo de estas subdivisiones es organizar la lista de manera tal que los interesados puedan comprender los requerimientos y puedan encontrar un requerimiento específico más fácilmente, puesto que, salvo en sistemas triviales, la lista tiende a tener una longitud considerable.

Cada entrada en la lista debe describir un requerimiento individual, particular o específico, puntual. Cada uno de los requerimientos debe cumplir lo mejor posible con todos los atributos de calidad listados en la Tabla 1, pero muy especialmente con los atributos de cohesión y observable externamente. En efecto, cada requerimiento debe hablar de una única cuestión manteniéndose lo más independiente posible de los otros requerimientos, y, por otro lado, debe describir algo que se pueda observar o medir sin tener que analizar la implementación en detalle. El siguiente es un ejemplo de un requerimiento que no es cohesivo [Som10]:

> *El sistema deberá soportar la generación y control de objetos de configuración; es decir, objetos que son agrupaciones de otros objetos del sistema. El sistema de control de configuraciones deberá permitir el acceso a objetos en un grupo utilizando incluso un nombre incompleto*[6].

Notar que: (a) se define el concepto de *objetos de configuración* que no es un requerimiento sino, precisamente, una definición por lo que debería incluirse en el "Glosario de términos" (ver más abajo); (b) hay un requerimiento general (*soportar la generación y control de objetos de configuración*), que debería incluirse en la sección "Descripción global de los requerimientos", y un requerimiento muy detallado y específico (*permitir el acceso a objetos en un grupo utilizando incluso un nombre incompleto*), que debería ser lo único a incluir en esta sección.

Cada uno de estos requerimientos debería enunciarse en términos de la frase "el sistema deberá...".

Esta lista puede tomar la forma de una tabla con la siguiente información:

Identificador. Cada requerimiento debe identificarse unívocamente.

Descripción. Es el requerimiento en sí descripto en lenguaje natural o con la ayuda de cualquier notación que sea perfectamente comprensible para el cliente.

Referencias. Aquí se liga el requerimiento con las fuentes desde las cuales se obtuvo: prototipos, escenarios de atributos de calidad, minutas de reunión, etc. Concretamente debe haber un *link* apuntando a cada prototipo, escenario de atributos de calidad, minuta de reunión, etc. que sirva de soporte al requerimiento.

Prioridad. Se consigna la prioridad del requerimiento según la escala mencionada más arriba.

Calidad. Se consigna la calidad del requerimiento según la escala mencionada más arriba. Opcionalmente se pueden asignar valores a cada uno de los atributos de calidad de la Tabla 1.

Participantes. Aquí se liga el requerimiento con los participantes que lo proveyeron (cliente), obtuvieron (ingenieros) y validaron (cliente). Los *links* deben apuntar a la Lista de contactos mencionada más abajo.

Otros datos. Se pueden consignar otros datos tales como fechas de obtención y validación del requerimiento, requerimientos relacionados, circuito de aprobación,

[6] En este caso el vocabulario utilizado no es tan erróneo como podría suponerse pues el requerimiento es parte de un *Ada programming support environment (APSE)*

etc.

- Glosario de términos. Es fundamental que en el documento se usen de forma consistente los términos utilizados por el usuario o propios del dominio de aplicación. En el glosario se da una definición clara de cada uno de estos términos y llegado el caso se pueden adjuntar sinónimos. Por ejemplo, si el usuario habla de "factura" y "tique" como cosas diferentes, entonces en el documento se deben utilizar siempre esos términos para hablar de cosas diferentes y no como sinónimos.
- Lista de contactos. Se deben consignar los datos de contacto de todos los participantes (interesados del cliente e ingenieros de la empresa desarrolladora).

Especificación. La especificación de requerimientos describe los requerimientos del lado del desarrollador. Se obtiene a partir de la **Lista consolidada de requerimientos** y es el documento que deberían utilizar los programadores para implementar el sistema. Este documento puede coincidir con la Guía de Módulos, que es parte de la documentación de diseño, o con el Modelo Funcional desarrollado durante la fase homónima del ciclo de vida de cascada ilustrado en la Figura 1.

Documento de requerimientos. El documento de requerimientos es la presentación integral de todos los productos del proceso de la Ingeniería de Software. Este documento puede utilizarse como contrato entre el cliente y el proveedor de software. Luego de lo dicho en la sección 1.2 resulta evidente que este documento *jamás* se termina por lo que debe ser constantemente revisado y actualizado. Por el lado de la empresa desarrolladora este documento es la fuente primordial a partir de la cual se define la arquitectura y el diseño del sistema (y a partir de estos las restantes descripciones del sistema).

1.4.1. Los requerimientos son ítem de configuración

En la Figura 4 se muestra que los productos del proceso de la Ingeniería de Requerimientos deben almacenarse en un CVS[7] tal como se lo debe hacer con los programas y las otras descripciones técnicas desarrolladas durante el proyecto.

Es deseable que cada requerimiento se almacene independientemente en el CVS de manera tal que se pueda seguir su evolución.

1.5. Participantes en el proceso de la Ingeniería de Requerimientos

Hasta el momento nos hemos referido superficial e indistintamente a cliente, usuario e interesado. En esta sección intentaremos precisar quiénes son estas personas y quiénes deberían ser los ingenieros de requerimientos, sus características, intereses, etc.

Podemos dividir a los participantes en el proceso de la Ingeniería de Requerimientos en dos conjuntos disjuntos: los ingenieros de requerimientos y los interesados[8].

1.5.1. Los ingenieros de requerimientos

Los ingenieros de requerimientos (comúnmente llamados analistas funcionales o simplemente analistas) trabajan para la compañía que desarrollará el software. Idealmente deben ser

[7] Sistema de Control de Versiones.

[8] Recordemos que este término es nuestra traducción para *stakeholders*

empleados	comunidades	accionistas
entidades crediticias	inversores	gobierno
gerentes de negocio	sindicatos	agencias de control
asociaciones empresarias	asociaciones profesionales	ONGs
posibles empleados	posibles clientes	comunidades locales
comunidades nacionales	público en general	escuelas
organizaciones asociadas	integradores	usuarios
desarrolladores de otros sistemas	gerente de proyecto	proveedores
gerentes de líneas de productos	equipo de aseguramiento de la calidad	contaduría
recursos humanos	administración	ingenieros de planta
relaciones públicas	legales	depósitos
logística	guardia	telefonistas
cajeros	cadetería	directorio
supervisores del contrato		

Tabla 2: Categorías de posibles interesados.

personas con gran capacidad de comunicación e interacción con otras personas, con un muy buen conocimiento del dominio de aplicación y con un excelente dominio del lenguaje escrito (redacción). Los programadores y los ingenieros de software NO son los mejores candidatos para esta tarea pues usualmente no cumplen ninguna de las dos características mencionadas. Los ingenieros de requerimientos NO necesitan ser expertos en informática y NO conviene que lo sean.

Una buena forma de contar con buenos ingenieros de requerimientos es contratar personal del cliente o de organizaciones que pertenecen al mismo negocio o dominio de aplicación.

1.5.2. Los interesados

Definición 5 (Interesado)**.** *El término interesado se utiliza para referirse a cualquier persona física o jurídica que pueda tener influencia directa o indirecta sobre el sistema.*

Como ya mencionamos es crucial para el éxito del proyecto determinar la lista de interesados en el sistema. Todos los interesados participarán de la **Obtención de Requerimientos** y/o de la **Validación de requerimientos**. Es muy difícil determinar a priori la lista exhaustiva de interesados por lo que no debe sorprendernos que la vayamos ampliando a medida que el proyecto avanza.

Un interesado que no haya sido incluido en la lista y que por lo tanto no participe de la obtención o validación de algún requerimiento puede, más tarde, señalar un error, omisión o contradicción en los requerimientos lo que implicará un costo y/o un conflicto entre el cliente y la empresa.

La Tabla 2 lista categorías de posibles candidatos a ser incluidos en la lista de interesados; en cada proyecto esta lista debería ser usada como *checklist*. Cada categoría debería particionarse en categorías más detalladas y de cada una de ellas se deberían listar tantos interesados como sea posible.

Obviamente tal cantidad de interesados no puede tener la misma visión, necesidades y

deseos sobre el sistema por lo que necesariamente se generarán conflictos entre ellos mismos y con la empresa desarrolladora. Uno de los principales roles de los ingenieros de requerimientos, y de aquí la necesidad de que sean personas con gran capacidad de comunicación, es la de resolver esos conflictos. La única forma de resolver un conflicto es expresarlo con claridad y sin preconceptos y discutirlo entre TODOS los afectados hasta alcanzar un consenso.

1.6. El negocio detrás del sistema

¿Por qué quiere desarrollar el sistema? Esta debe ser la primera pregunta que haga el ingeniero de requerimientos a quién paga por el sistema (que es uno de los interesados clave). Si la respuesta no está directa y claramente relacionada con maximizar las ganancias o minimizar los costos, entonces el proceso de la Ingeniería de Requerimientos debe detenerse inmediatamente. Si en la respuesta el cliente no incluye una *cuantificación* de sus expectativas respecto de los costos y beneficios que lo impulsan a construir el sistema, entonces será muy difícil determinar los requerimientos. Estos datos se llaman habitualmente *el negocio detrás del sistema*[9] [Rob04]. Resumiendo: si quien paga por el desarrollo no tiene claro el negocio detrás del sistema, entonces mejor ni empezar.

El negocio detrás del sistema es el criterio último y definitivo para determinar:

- Cada uno de los requerimientos.
- Si algo que un interesado pide es un requerimiento o no. O dicho de otra forma si un requerimiento es correcto o no. Es decir, el negocio detrás del sistema brinda un criterio para determinar qué es un requerimiento y qué no lo es.
- La prioridad de cada requerimiento.
- La completitud de los requerimientos; es decir, si faltan requerimientos o no.

El negocio detrás del sistema se genera, obviamente, a partir del negocio del cliente. La persona que decide invertir en el proyecto es la persona responsable de cuantificar los costos y beneficios esperados. Para que el negocio detrás del sistema constituya un criterio efectivo para determinar los puntos listados más arriba, los ingenieros de requerimientos deben contar con la siguiente información:

- Los objetivos de la inversión.
- La cantidad a invertir (tiempo y dinero).
- Los beneficios esperados o deseados para el negocio.

El negocio detrás del sistema permite encontrar el camino hacia los interesados y el dominio de aplicación (sección 1.7) y, a partir de estos, los requerimientos y el criterio para priorizarlos. Claramente, el objetivo de la inversión indicará de forma directa o indirecta los sectores de la organización que se verán afectados por el sistema, lo que permite determinar los interesados y el dominio de aplicación. Por ejemplo, si uno de los objetivos de la inversión dice algo como "pensamos en un nuevo sistema de ventas que nos permita incrementar las ventas en un 15% anual" entonces los interesados más directos serán la gerencia de ventas y los vendedores, y el dominio de aplicación los productos y/o servicios que se venden, las formas en que se venden, el plan de negocios de la empresa, los clientes que compran esos productos, etc.

[9] Traducción de *the business case for the system*.

Entonces, cualquier requerimiento que no aporte sustancialmente a alcanzar los objetivos del negocio dentro del presupuesto o no es un requerimiento o "es posible, pero podría eliminarse". Por ejemplo, un vendedor podría plantear la necesidad de "acceder a las últimas compras del cliente cuando lo visita" lo que claramente aportará a alcanzar el objetivo; pero otro podría requerir que el sistema le permita "obtener un Veraz del cliente para ver su capacidad de pago", lo que ciertamente es interesante pero dudosamente aporte algo significativo dado que la empresa vende al contado.

No es de extrañar que en ocasiones la organización no esté en condiciones de desarrollar el negocio detrás del sistema con la precisión necesitan los ingenieros de requerimientos. Por otro lado, difícilmente los ingenieros de requerimientos podrán enunciar por sí solos el negocio detrás del sistema de una organización que apenas conocen. Las cualidades del negocio que se introducen en la sección 3.2.7 pueden ayudar a acercar el negocio a los ingenieros de requerimientos de manera tal que estos puedan ayudar al cliente a elaborar el negocio detrás del sistema.

1.6.1. El negocio errado

Es muy común que el ingeniero escuche que el cliente quiere cambiar el sistema porque "es viejo u obsoleto", "es una tecnología en desuso", "queremos algo más moderno", "nos gustaría una aplicación web", etc. Todos estos son imperativos tecnológicos, no pertenecen al negocio y por lo tanto no pueden ser el negocio detrás del sistema. En consecuencia, ante este tipo de respuestas la primera pregunta que debe hacer el ingeniero de requerimientos es "¿pero cuánto esperan ganar o ahorrar al desarrollar o comprar el nuevo sistema, qué nuevos negocios esperan poder hacer, cuántos clientes más esperan tener?". En otras palabras debe lograr que el cliente presente el verdadero negocio detrás del sistema.

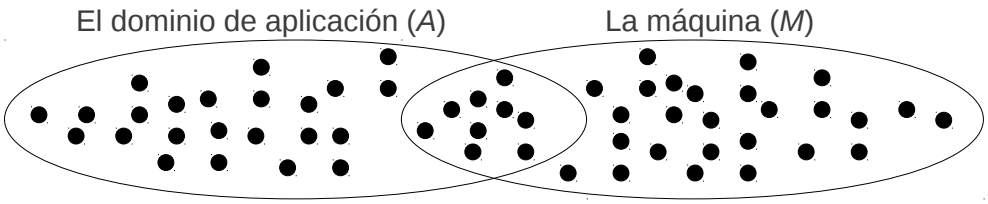
Un sistema de cómputo es a una organización lo que un camión es a un camionero: una herramienta de trabajo. No se cambia solo porque está vieja. Hay Jumbos que vuelan desde 1970.

1.7. La visión de Jackson sobre los requerimientos: una teoría para la Ingeniería de Requerimientos

Michael Jackson ha trabajado durante años en el área de Ingeniería de Requerimientos [Jac95, GGJZ00, ZJ97]. En nuestra opinión sus trabajos en este área arrojan luz sobre algunos problemas que han sido evitados o ignorados por los tratamientos clásicos de esta disciplina. En esta sección resumiremos brevemente su pensamiento y principales resultados.

Según Jackson los ingenieros de software construyen *máquinas*. No construyen el anfitrión físico sino que construyen el comportamiento y las propiedades de la máquina que la harán útil para algún propósito particular. El propósito de una de estas máquinas es ser instalada en el mundo real e interactuar con aquel. La parte del mundo en el cual los efectos de la máquina serán percibidos, evaluados y, con suerte, aprobados por el cliente, se llama *dominio de aplicación*. Tal vez debido a que el desarrollo de software trata sobre la construcción de máquinas, la práctica tradicional ha sido ignorar el dominio de aplicación y focalizar la atención en la máquina. Sin embargo, el dominio de aplicación es donde yacen los requerimientos del cliente, por lo que si el ingeniero no es capaz de identificar correctamente el dominio de aplicación, no será capaz de focalizarse sobre los requerimientos del cliente.

Figura 5 *A* es el conjunto de fenómenos del dominio de aplicación y *M* es el conjunto de fenómenos de la máquina.



Entonces, no debería sorprendernos que si el ingeniero pone énfasis en la máquina, y no en el dominio de aplicación, parezca como que el cliente no sabe lo que quiere o desconoce sus propios requerimientos. Es una creencia común entre los desarrolladores de software pensar que sus clientes no saben lo que quieren, pero Jackson opina que esta superioridad se ha alcanzado solo en virtud de nuestras propias limitaciones y fallas.

Los requerimientos hablan sobre el dominio de aplicación y no sobre la máquina que eventualmente los implementará. En consecuencia es crucial alcanzar una compresión del dominio de aplicación en tanto el problema a resolver se define más en términos de la estructura y propiedades del dominio de aplicación, que en términos de la naturaleza de la máquina. En un sentido el dominio de aplicación es el material que le es dado al ingeniero y los requerimientos es lo que el ingeniero debe hacer con ese material.

Determinar el dominio de aplicación no siempre es sencillo. Parecería natural comenzar investigando los requerimientos para el sistema mirando al viejo sistema (siempre hay un sistema anterior, sea manual o automático, a menos que la organización sea completamente nueva). Sin embargo, esto puede llevar a confusión. Por ejemplo, si se está desarrollando un nuevo sistema de sueldos, el ingeniero puede caer fácilmente en la idea de que el dominio de aplicación es la gerencia de recursos humanos o sueldos. Eso significaría que los requerimientos son sobre los documentos que pasan de empleado a empleado en esa sección. Si bien eso puede ser correcto, en ese caso el sistema no sería un sistema de sueldos sino más bien un sistema de procesamiento de documentos relacionados con el pago de sueldos. Es mucho más probable que si se trata de un sistema de sueldos los requerimientos sean sobre los empleados, sus escalas salariales, sus horarios de trabajo y las horas extras, sus ausencias, feriados, jubilaciones, seguros médicos, etc. Estas son las cosas que se deberían escribir en la **Lista consolidada de requerimientos**.

Aunque los requerimientos se deben buscar en el dominio de aplicación eventualmente se debe construir una máquina (programa o software) que los implemente. La máquina, entonces, debe poder acceder o percibir los *fenómenos* del dominio de aplicación y este los de aquella, pues de lo contrario sería imposible cualquier interacción. En este sentido el dominio de aplicación y la máquina comparten fenómenos (eventos y elementos) como se grafica en la Figura 5. En términos de *A* y *M* se deberían escribir cinco descripciones (aunque solo las tres primeras son relevantes a la Ingeniería de Requerimientos):

El conocimiento sobre el dominio de aplicación. Este documento, llamado *D*, describe las propiedades que posee el dominio de aplicación independientemente de lo que pueda hacer bien o mal la máquina. El vocabulario es únicamente de *A*.

Los requerimientos. Este documento, llamado *R*, describe las propiedades del dominio de

aplicación que el cliente desea que se satisfagan al instalar la máquina. El vocabulario es únicamente de A.

La especificación. Este documento, llamado S, describe las propiedades de la interfaz entre el dominio de aplicación y la máquina. El vocabulario es únicamente de $A \cap M$.

El programa. Este documento, llamado P, describe la propiedades que el desarrollador ha incorporado a la máquina. El vocabulario es únicamente de M.

El conocimiento sobre la máquina. Este documento, llamado C, describe las propiedades del hardware, el software de base y del lenguaje de programación. El vocabulario es únicamente de M.

En consecuencia Jackson propone que para garantizar que el desarrollo cumple con las expectativas del cliente debería ser posible probar los siguientes teoremas:

$$C \wedge P \Rightarrow S$$

y

$$D \wedge S \Rightarrow R$$

1.8. Principios de la Ingeniería de Requerimientos

Por múltiples razones, algunas de las cuales hemos explicado en las secciones anteriores, la Ingeniería de Requerimientos es una tarea crucial para el éxito de un proyecto de desarrollo de software y a la vez es una actividad muy compleja, llena de obstáculos y que requiere de los ingenieros poner en juego sus habilidades menos "duras". Por estos motivos creemos que es muy importante resaltar los principios que nunca deben olvidarse a la hora de desarrollar esta actividad, y en particular a la hora de concebir y aplicar técnicas o herramientas para ayudar en la tarea.

- El negocio detrás del sistema es el último y fundamental criterio para incluir, rechazar y priorizar cada requerimiento particular [Rob04].
- Los requerimientos están en el dominio de aplicación y se los debe describir con ese vocabulario [Jac95].
- Nunca ir a una reunión con un cliente sin un prototipo [Sch04].

Cualquier metodología, técnica o herramienta que se proponga para asistir en la Ingeniería de Requerimientos debe basarse en todos estos principios pues de lo contrario alguna cuestión fundamental de esta disciplina no será tenida en cuenta y pondrá en riesgo el éxito de la actividad y del proyecto en general.

1.9. Temas y problemas en la obtención de requerimientos

Hasta el momento hemos abordado el tema de la Ingeniería de Requerimientos desde un punto de vista más bien teórico. En esta sección daremos algunas indicaciones prácticas generales según [BPKR09, páginas 41–48].

Obtener los requerimientos puede convertirse en una tarea dolorosa y no bien recompensada. Muchas veces la Ingeniería de Requerimientos es vista como algo de último momento y

asignado a personal junior. Existen muchos proyectos donde no han sido documentados los requerimientos y se lo intenta hacer cerca de la entrega del sistema. Si la obtención de los requerimientos no funcionales se retrasa hasta el fin del proyecto, es muy probable que el sistema no los verifique (es decir, no tenga la seguridad o usabilidad adecuadas). Algunas cuestiones que deben tenerse en cuenta durante la obtención de requerimientos se detallan a continuación.

- La obtención de requerimientos debe ser liderada por personal experimentado en Ingeniería de Requerimientos. En el equipo pueden participar jóvenes sin experiencia como método de entrenamiento. Se sugiere que participe alguien sin conocimiento del dominio de aplicación de manera tal que se sienta habilitado a hacer preguntas obvias que el no considera necesario efectuar. Es común que los ingenieros a cargo manejen los mismos conceptos e ideas que los interesados pero con ciertas diferencias que luego tienen impacto en el sistema.
- Es importante que el equipo siempre pregunte "¿por qué?" ante cada pedido de un interesado. La respuesta siempre tiene que estar en términos del negocio detrás del sistema.
- Hay que tener en cuenta el nivel de responsabilidad de cada interesado. En particular:
  - ¿El interesado habla por toda la organización? ¿Está autorizado a hacerlo?
  - Si dos o más interesados tienen diferencias en cuanto al sistema, ¿son diferencias sobre la funcionalidad del sistema o hay temas que no han sido resueltos dentro de la organización?
  - Los interesados, ¿tienen el conocimiento adecuado sobre el dominio de aplicación?
- Los desarrolladores, programadores, arquitectos, expertos en bases de datos, etc. no suelen ser buenos ingenieros de requerimientos aunque tengan una basta experiencia y un curriculum impresionante.
- Cada pedido de un interesado debe estar acompañado por el nombre de esa persona y la fecha y lugar en que lo efectuó. Esta información debe preservarse cuando el pedido se transforme en requerimiento. Esta información puede usarse inteligentemente para priorizar los requerimientos: no es lo mismo un pedido de una unidad de negocios pequeña que de una que es fundamental para la organización. Sin la identificación del interesado que formuló el pedido se pueden priorizar requerimientos solicitados por interesados con menor influencia en el proyecto, y se pueden dejar de lado a los importantes.
- Los interesados clave del sistema suelen ser personas importantes para su organización por lo cual su tiempo disponible para participar en reuniones de obtención de requerimientos puede ser muy limitado. Sin la participación de estos interesados el proyecto peligra. Esto debe ser informado claramente y por escrito a quien paga por el sistema. Debería constar en el contrato y en la planificación inicial que la disponibilidad de estos interesados debe cumplir tales y cuales parámetros. Si esto no se cumple la responsabilidad recae sobre el contratante.
- Muchas veces se les pide a los ingenieros de requerimientos que participen en la definición de requerimientos sobre los que los interesados tienen dudas. Incluso no es raro que los interesados no tengan claro el negocio detrás del sistema. Esto debe ser resuelto antes de formalizar los primeros requerimientos. Se pueden aplicar varias técnicas:
  - Prototipación rápida (ver secciones siguientes).

  - Muchas veces el solo intento de obtener los requerimientos clarifica las ideas y necesidades de los interesados. Cuando eso ocurre documentar el negocio detrás del sistema antes que los primeros requerimientos.
  - Puede ser conveniente que los interesados desarrollen un manual de usuario o documentación de marketing con el fin de determinar el negocio detrás del sistema. Es decir, el hecho de tratar de construir lo que ellos le presentarán a sus clientes les puede ayudar a determinar qué es lo que quieren. Al intentar realizar estas actividades el mismo cliente puede darse cuenta que aun no conoce lo suficiente sobre el mercado.
- Es importante que los interesados entiendan qué son capaces de hacer las computadoras actuales, de manera que no realicen pedidos irrealizables.
- Puede ser peligroso que el ingeniero de requerimientos tenga un conocimiento muy profundo del dominio de aplicación. La comunicación con el cliente es un deber irrenunciable del ingeniero.
- Si los interesados tienen visiones contradictorias sobre un aspecto importante del sistema, su resolución debe programarse para una reunión específica donde solo se discuta este tema.

Ejercicios

Ejercicio 1. Considere una farmacia que se plantea los siguientes objetivos para su negocio:

- Abrir dos sucursales en la ciudad.
- Diferenciar la atención entre clientes de obras sociales y clientes particulares.
- Agilizar la autorización de recetas de obras sociales.

Suponga que la farmacia desea comprar o desarrollar un sistema informático que le ayude a cumplir los objetivos. Entonces:

1. Escriba el negocio detrás del sistema alineado con los objetivos descritos.
2. Determine una lista preliminar de interesados.
3. Escriba la lista estructurada de requerimientos funcionales (no menos de 10 requerimientos).

Ejercicio 2. Considere un restaurante que se plantea los siguientes objetivos para su negocio:

- Establecer una cadena de locales a nivel provincial.
- Evitar diferencias entre comandas y facturas.
- Mejorar control de stock y pedidos a proveedores.

Suponga que el restaurante desea comprar o desarrollar un sistema informático que le ayude a cumplir los objetivos. Entonces:

1. Escriba el negocio detrás del sistema alineado con los objetivos descritos.
2. Determine una lista preliminar de interesados.
3. Escriba la lista estructurada de requerimientos funcionales (no menos de 10 requerimientos).

---

2. Obtención y Validación de Requerimientos Funcionales

Las tres cosas más importantes para obtener y validar requerimientos funcionales son:

1. Nunca ir a una reunión con un cliente sin un prototipo.
2. Nunca ir a una reunión con un cliente sin un prototipo.
3. Nunca ir a una reunión con un cliente sin un prototipo.

Es tan evidente el acierto de esta frase como lo es la necesidad de definir una metodología y herramientas para llevarla a la práctica. ¿Qué es un prototipo? ¿Cuánto tiempo lleva desarrollarlo? Si hablamos de prototipos desechables, ¿cuánto vamos a invertir en desarrollarlos si luego hay que tirarlos? ¿Cuándo deja de ser beneficioso desarrollar un prototipo para obtener el requerimiento correcto y comienza a ser más rentable implementar el programa incorrecto para luego corregirlo? Si hay que ir a cada reunión con el cliente con un prototipo, ¿cuánto podemos demorar en desarrollar un prototipo sin alargar de forma absurda la ingeniería de requerimientos? Si desarrollar un prototipo implica programar, ¿quién debe desarrollarlos? ¿Los ingenieros de requerimientos a quienes les pedimos ser menos técnicos, conocer más el dominio de aplicación y describir los requerimientos sin hablar en términos de implementación? ¿Los programadores, entonces, quienes no estuvieron con el cliente y a los cuales hay que transmitirle lo que el cliente dijo? ¿No podríamos introducir más errores al comunicarles los *supuestos* requerimientos?

Claramente, para cumplir con el principio postulado por Schrage el desarrollo de prototipos debe verificar las siguientes condiciones:

1. Desarrollar un prototipo para dos o tres requerimientos no debe consumir más de dos o tres horas.
2. Modificar un prototipo para adaptarlo a las correcciones indicadas por el cliente durante la validación del mismo, debe demorar menos de quince minutos.
3. Desarrollar un prototipo no debería implicar programar, pues así los ingenieros de requerimientos sin demasiados conocimientos técnicos podrán hacerlo.

En las secciones siguientes se presenta una metodología para obtener y validar requerimientos funcionales utilizando casi exclusivamente prototipos desechables (2.1) y una herramienta que da soporte o asistencia a esa metodología (2.2).

Ahora, ¿por qué prototipos desechables y no prototipos *evolutivos*? La razón es muy sencilla: el objetivo de los prototipos evolutivos es entregar un sistema en funcionamiento al cliente, en tanto que el objetivo de los prototipos desechables es obtener y/o validar los requerimientos [Som10]. Pero, ¿por qué no ir desarrollando un sistema a partir de uno que implemente, mal o bien, los primeros requerimientos como propone la comunidad de la prototipación evolutiva? Porque en general no será posible diseñar correctamente un sistema desarrollado de esa forma, y un buen diseño (es decir un diseño que pueda incorporar eficientemente los cambios más probables) es la clave para un sistema flexible y duradero. Trataremos extensivamente el diseño de software en capítulos posteriores.

2.1. Metodología para la obtención y validación de requerimientos funcionales basada en prototipación rápida desechable

La metodología que proponemos, graficada en la Figura 6, comienza con una primera reunión con los referentes clave del sistema (gerentes, directores, jefes de área, socios, etc.) en la cual, a requisitoria de los ingenieros, deben presentar el negocio detrás del sistema y una exposición general sobre los requerimientos. El contenido y estructura de la reunión deben ser comunicados a los referentes con suficiente antelación, pues de lo contrario esta primera reunión seguramente será un fracaso. Si el cliente logra presentar satisfactoriamente el negocio detrás del sistema, se pasa a la etapa siguiente; caso contrario, se debe asistir al cliente en elaborar el negocio detrás del sistema.

El segundo paso de nuestra metodología consiste en listar todos los interesados relevantes, teniendo en cuenta el contenido de la etapa previa. Esta tarea puede, y de hecho ocurre muy frecuentemente, no completarse en un solo paso por lo que la lista se va completando a medida que avanza la ingeniería de requerimientos. Para determinar, al menos, una lista preliminar se debe considerar el dominio de aplicación según surge de la exposición del negocio detrás del sistema y de los requerimientos mencionados por el cliente en la etapa anterior, y proyectarlos sobre el organigrama de la empresa, sus proveedores, clientes, asociados, organismos de control, etc. Por ejemplo, si el negocio detrás del sistema dice algo como "con este nuevo sistema de control de *stock* se espera que la empresa reduzca en un 50% los retrasos en las entregas lo que redundará en mayor satisfacción de los clientes y un consecuente aumento en las ventas", entonces los primeros interesados serán: el área de almacenes, logística, compras, proveedores, atención al cliente, etc. Probablemente con el correr del proyecto algunos de estos primeros interesados dejarán de serlo y aparecerán otros. Por otro lado, se puede recurrir a alguna lista de referencia de potenciales interesados y utilizarla como *checklist*. La lista inicial deberá incluir a todos los participantes de la primera reunión.

2.1.1. Desarrollo de prototipos

Luego de la primera reunión con el cliente los ingenieros deberían construir un primer prototipo para llevarlo a la siguiente reunión con los interesados, como se explica en la sección 2.2. La segunda reunión y las subsiguientes deben estructurarse alrededor del uso y/o modificación de los prototipos que el equipo de ingenieros vayan desarrollando. A las primeras reuniones deben asistir los interesados que parezcan ser los más prometedores para darnos los requerimientos fundamentales del sistema. En cada una de estas reuniones, que seguramente durarán no más de un par de horas, los ingenieros alentarán a los interesados a utilizar el prototipo y a plantear sus diferencias y expectativas sobre el sistema final. Preguntarán a los interesados, a partir del negocio detrás del sistema y de los requerimientos generales con que ya cuentan, sobre requerimientos más específicos. Uno, dos o tres requerimientos específicos son suficientes como para dar por finalizada la reunión. Por ejemplo, requerimientos más específicos pueden ser "la planta nos pasa un pedido de materiales, nosotros los buscamos en el almacén, les entregamos los que podemos y los damos de baja en el sistema, hacemos un pedido a compras de los que no tenemos disponibles, y nos sentamos a esperar a que lleguen". Al finalizar la reunión se acuerda una fecha, hora, lugar y participantes para la siguiente.

A menos que las reuniones sean especialmente productivas, desarrollar los primeros prototipos no debería consumir más de entre 2 y 10 horas-hombre (los primeros llevarán cerca de

Figura 6 Una descripción general de la metodología.

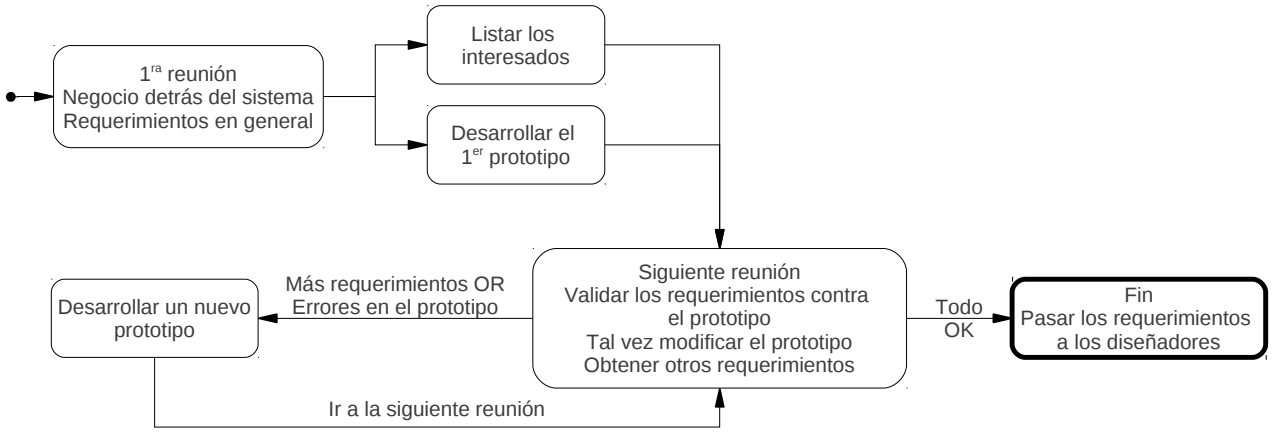
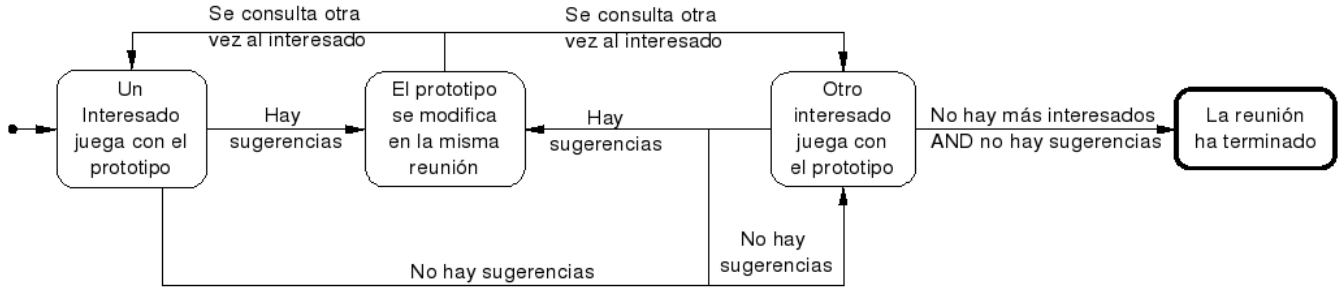


Figura 7 Validar los requerimientos con cada interesado.



10 horas-hombre en tanto que los siguientes requerirán menos tiempo). De esta forma podría pactarse una reunión cada 48 horas.

El equipo de ingenieros asiste a cada reunión con un prototipo. Los interesados "juegan" con el prototipo guiados por los ingenieros, quienes les presentan una serie de ejemplos de cómo usar el sistema para observar los resultados que este arroja. En el mejor de los casos cada interesado que participa de la reunión debe ejecutar todos y cada uno de los ejemplos y dar su opinión sobre cada uno de ellos. Estas opiniones tomarán la forma de aprobaciones, correcciones importantes o secundarias, indicaciones de incompletitud, etc. Los ingenieros deben tomar notas de todas ellas para corregir y ampliar el prototipo. Algunos de los cambios pueden hacerse durante la misma reunión, como se verá en la sección 2.2.

Es sumamente importante para nosotros notar que durante cada reunión se *obtienen* **y** *validan* requerimientos, y se abre el camino para el testing de aceptación, como veremos en los últimos capítulos.

Otro punto a tener en cuenta es que en muchas ocasiones los interesados se dan cuenta de un error, omisión o modificación en los requerimientos varios días después de la reunión y traen el tema luego de que se ha llegado a un acuerdo sobre este. Es parte de la habilidad

de los ingenieros resolver estas cuestiones para lograr cumplir con los plazos, presupuesto y parámetros de calidad.

2.1.2. Validación cruzada

En la Figura 7 se muestra con más detalle cómo debería estructurarse cada reunión, aunque los detalles precisos dependen de cosas como si todos los interesados están disponibles en el mismo momento y en el mismo lugar, cuánto tiempo hay para que los interesados estén a disposición de los ingenieros, etc. El punto clave que intentamos resaltar aquí es hacer una *validación cruzada* de cada requerimiento o prototipo. Por validación cruzada entendemos que el prototipo es validado por todos los interesados, si es posible en la misma reunión, de forma tal que ellos mismos resuelvan sus diferencias, intereses, prioridades y expectativas específicas sobre el sistema.

La figura sugiere una validación cruzada lineal pero podrían implementarse esquemas más seguros. Por ejemplo, si k interesados han validado un prototipo sin sugerir cambios pero el interesado $k+1$ hace sugerencias, lo que implica modificar el prototipo, sería conveniente hacer que los k primeros interesados vuelvan a validarlo.

2.1.3. Ejemplos: la forma de definir prototipos

En el contexto de nuestra metodología validar el prototipo o el requerimiento (que en definitiva es la misma cosa), significa que los usuarios "jueguen" o usen el prototipo casi como si fuera el sistema final. Usar o jugar con el prototipo significa, a su vez, ejecutar lo que nosotros denominamos *ejemplos*. Un ejemplo es una forma específica de usar el sistema que define valores concretos para cada una de las variables de entrada, de estado y de salida del sistema o de una parte de este. Al ejecutar un ejemplo el usuario puede observar la salida producida a consecuencia de las entradas que él mismo ingresó. La diferencia sustancial con el sistema terminado es que el prototipo solo puede ejecutar algunos pocos ejemplos. Estos ejemplos son definidos por los ingenieros al construir el prototipo, a raíz de la interacción con los interesados en la reunión actual o en las anteriores. A partir de los ejemplos y de lo que dicen los interesados, los ingenieros enuncian los requerimientos en lenguaje natural de manera tal que cada ejemplo tiene que ser, precisamente, un ejemplo de ese enunciado más general. De esta forma, el prototipo, el enunciado escrito del requerimiento y los ejemplos se sustentan mutuamente para dar mayor solidez, claridad y precisión al pedido de los interesados.

2.1.4. Los prototipos son desechables... pero no se desechan

Los prototipos solo se utilizan para que los ingenieros, los interesados y el resto del equipo de desarrollo comprendan y acuerden los requerimientos. El arquitecto y/o los diseñadores del sistema reciben toda esta información y a partir de ella generan la arquitectura y el diseño del sistema. El cliente y la empresa desarrolladora pueden usarla para anexarla al contrato comercial. En particular los ejemplos pueden convertirse en el criterio de corrección y aceptación del sistema: si todos los ejemplos definidos durante la ingeniería de requerimientos son ejecutados sobre el sistema final y este da las respuestas por ellos mismos descriptas, entonces el sistema es correcto y el cliente debe aceptarlo. Claramente, los prototipos no se desechan sino que sirven hasta la entrega del sistema.

Sin embargo, son desechables porque no se programa sobre ellos. Es decir, los diseñadores no usan el diseño de los prototipos para generar el diseño del sistema, ni los programadores usan el código de los prototipos para seguir programando.

2.2. Repose: una herramienta para prototipación rápida desechable

Repose es un desarrollo experimental de Flowgate Consulting y personal de CIFASIS[10] que implementa la metodología descripta en la sección anterior. Una característica interesante de nuestra herramienta es que solo se necesita saber usar software de oficina más o menos estándar para poder aplicarla; el usuario de nuestra herramienta no necesita saber programar para desarrollar prototipos en muy poco tiempo.

En tanto es aun un desarrollo experimental se debe tener en cuenta que no se han programado las comprobaciones de entrada habituales por lo que los ingenieros deben ser sumamente cuidadosos al usar la herramienta, ya que de lo contrario el comportamiento será impredecible.

Lo que sigue es una suerte de manual de usuario de la herramienta.

2.2.1. Requisitos para instalar la herramienta

Repose es un programa Java que interactúa con OpenOffice.org. Entonces los requisitos para instalar Repose son los siguientes:

- Linux o Windows (XP, Vista, 2000, 2003, etc.)
- Java SE Runtime Environment 1.6 o superior
- OpenOffice.org 2.3.0 o superior (tener en cuenta que Repose usa la definición de OpenDocument Format de OpenOffice.org la cual suele variar de versión en versión por lo que podría dejar de funcionar en futuras versiones de OpenOffice.org)
- 7-Zip para Windows o gunzip para Linux

Para instalar Repose en Linux solo hay que descomprimir el archivo `repose.tar.gz` en cualquier directorio. Para hacerlo sobre Windows las instrucciones son las siguientes:

1. Descomprimir el archivo `repose.tar.gz` en cualquier carpeta.
2. Instalar 7-Zip, cuyo instalador puede descargarse gratuitamente desde `http://www.7-zip.org`.
3. Una vez instalado 7-Zip, buscar en la carpeta donde se lo instaló (la carpeta por defecto es `C:\Archivos de programa\7-Zip`) el archivo `7z.exe` y copiarlo en la carpeta donde se haya instalado Repose.

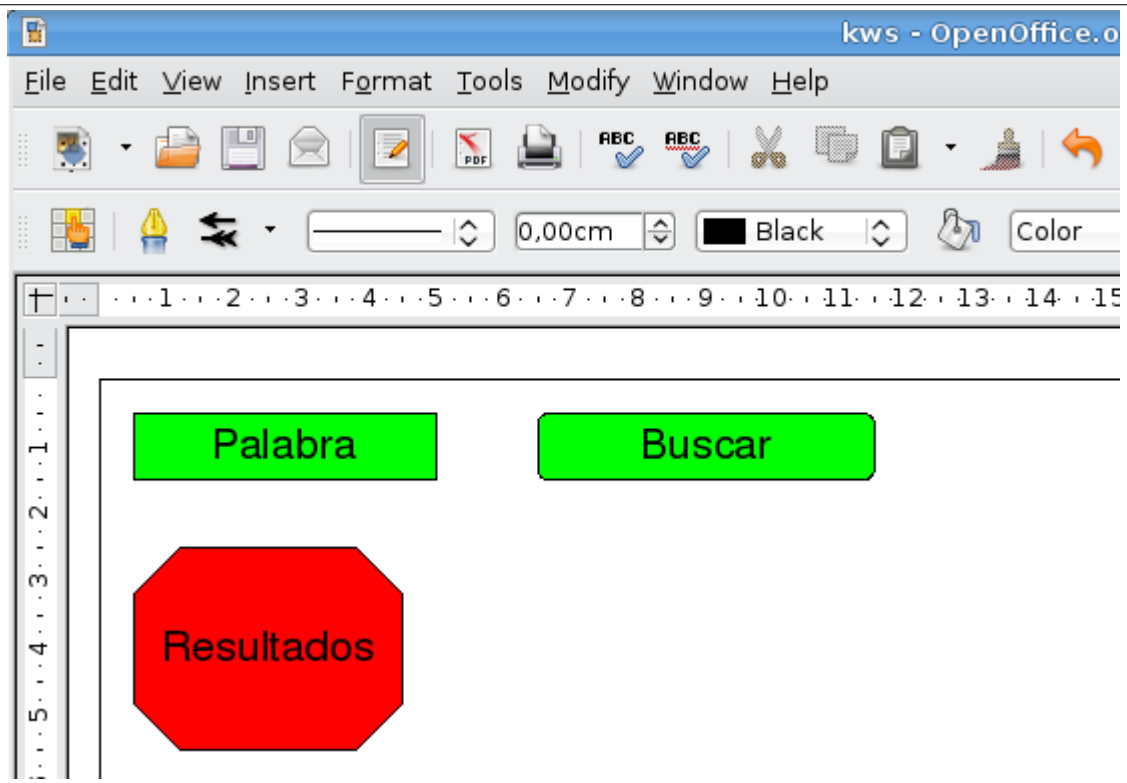
2.2.2. Panorama general de Repose

En primer lugar debemos aclarar que el ámbito de Repose es prototipar aplicaciones con rica interacción con el usuario a través de menús, formularios y otros objetos de interfaces gráficas de usuario (GUI).

El primer paso antes de usar Repose es dividir los requerimientos en función de formularios o pantallas a través de las cuales interactuará el usuario con la aplicación. Puede pensarse que

[10] CIFASIS es el Centro Franco-Argentino de Ciencias de la Información y de Sistemas, instituto de investigación que depende de CONICET, Universidad Nacional de Rosario y Universidad de Marsella.

Figura 8 Dibujando la GUI.

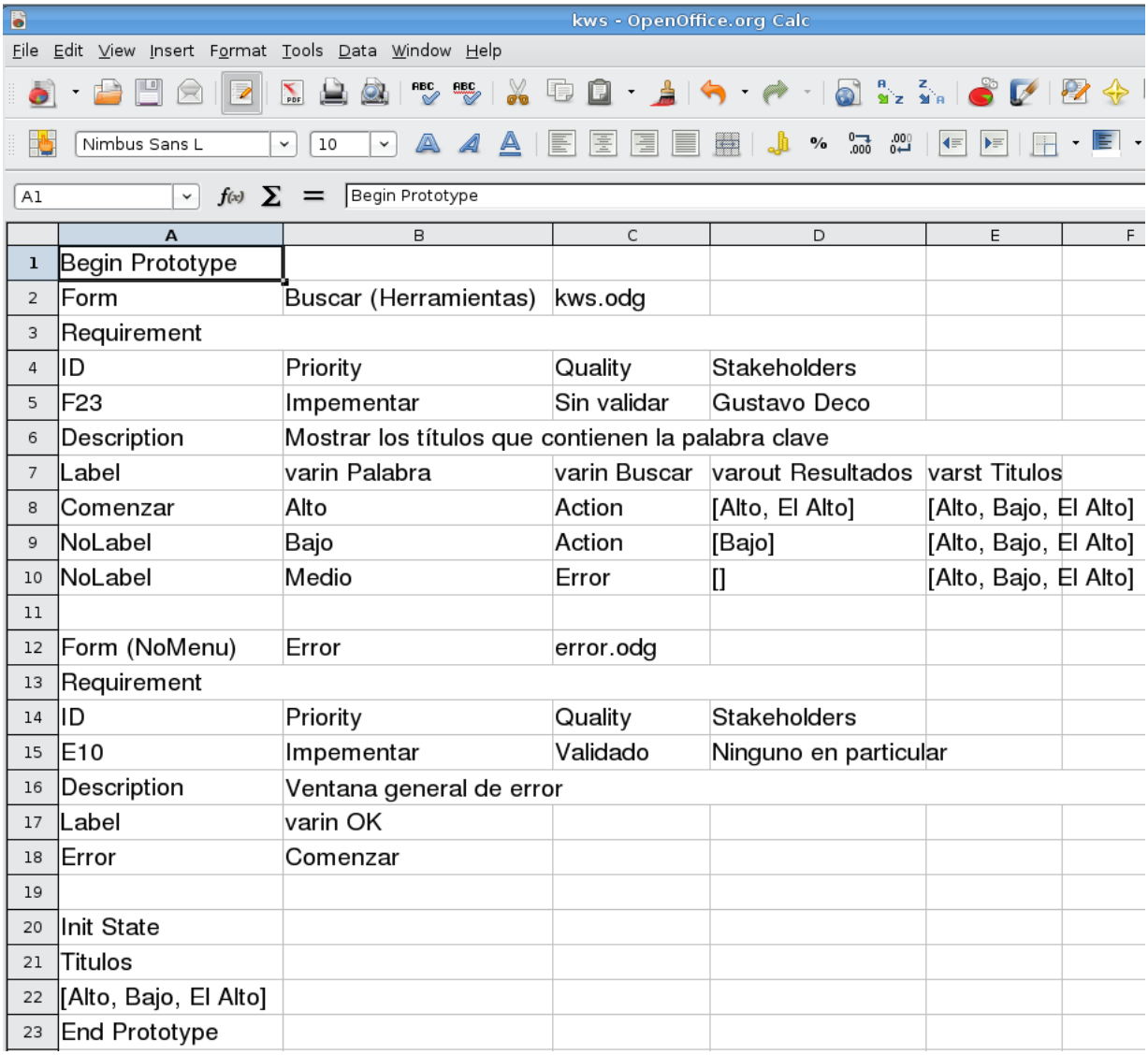


detrás de cada formulario se implementan las reglas de negocio que corresponden al requerimiento. Por ejemplo, si el requerimiento es "procesar facturas", entonces podemos dividirlo en "hacer una factura", "borrar una factura" y "modificar una factura" y asociar a cada uno de ellos formularios que les permitirán a los usuarios realizar esas tareas.

El segundo paso es listar todos los datos que cada formulario necesita clasificándolos en datos de entrada, datos de estado y datos de salida. Los datos de entrada son los que ingresa el usuario (incluyen campos de texto, botones, listas desplegables, etc.), los datos de estado son aquellos que se toman de una base de datos o de un archivo (aunque no es necesario indicar ni la base de datos ni el archivo), y los datos de salida son los que produce el programa luego de ejecutar (por ejemplo un mensaje de error, un listado, etc.). Con toda esta información se define o especifica el prototipo como se describe en la sección 2.2.3; aquí solo daremos una descripción general.

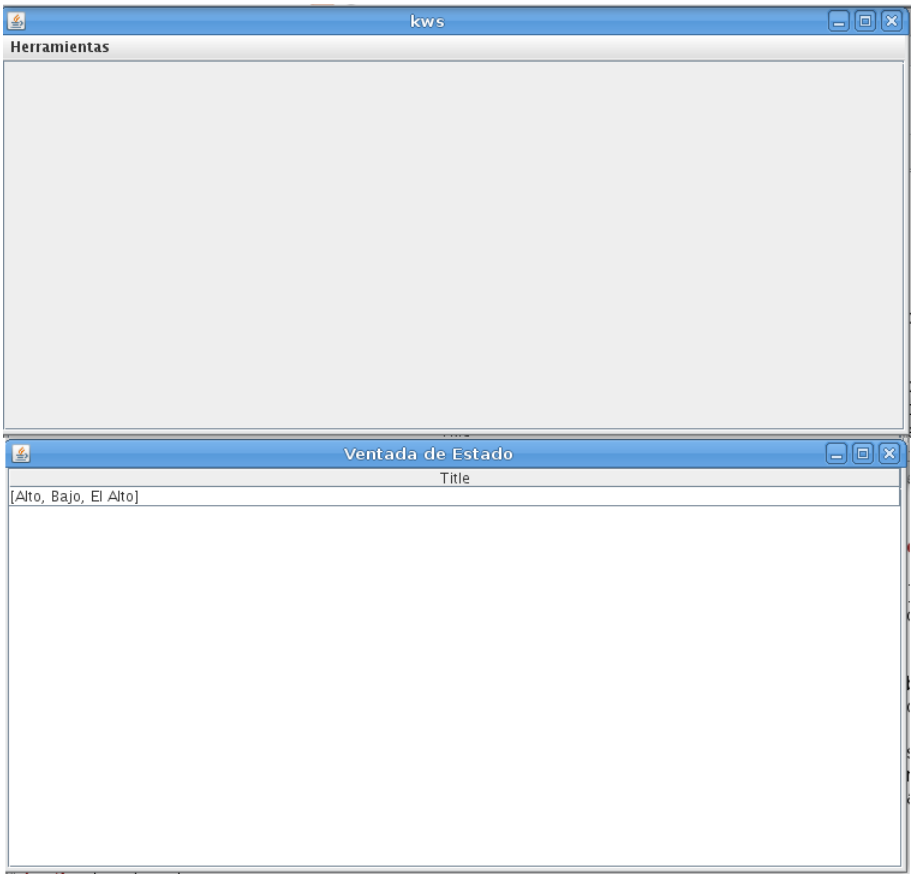
Primero se usa el programa OpenOffice.org Draw para dibujar cada formulario como se muestra en la Figura 8. Cada figura geométrica representa un control de interfaz de usuario diferente; por ejemplo, los rectángulos con bordes redondeados representan botones. Luego, a partir del formulario, Repose genera el esqueleto de una planilla de cálculo donde el ingeniero debe definir el comportamiento del prototipo. Usando OpenOffice.org Calc el ingeniero debe listar los ejemplos que definirán el comportamiento del prototipo como se muestra en la Figura 9. Finalmente, se compila y ejecuta el prototipo. Cuando se ejecuta el prototipo se abren dos ventanas (Figura 10): una es la ventana propia del programa que se está prototipando, la otra es la ventana que muestra la base de datos, llamada *ventana de estado*. Observar que la ventana

Figura 9 Los ejemplos que definen el comportamiento del prototitpo.



	A	B	C	D	E	F
1	Begin Prototype					
2	Form	Buscar (Herramientas)	kws.odg			
3	Requirement					
4	ID	Priority	Quality	Stakeholders		
5	F23	Impementar	Sin validar	Gustavo Deco		
6	Description	Mostrar los títulos que contienen la palabra clave				
7	Label	varin Palabra	varin Buscar	varout Resultados	varst Titulos	
8	Comenzar	Alto	Action	[Alto, El Alto]	[Alto, Bajo, El Alto]	
9	NoLabel	Bajo	Action	[Bajo]	[Alto, Bajo, El Alto]	
10	NoLabel	Medio	Error	[]	[Alto, Bajo, El Alto]	
11						
12	Form (NoMenu)	Error	error.odg			
13	Requirement					
14	ID	Priority	Quality	Stakeholders		
15	E10	Impementar	Validado	Ninguno en particular		
16	Description	Ventana general de error				
17	Label	varin OK				
18	Error	Comenzar				
19						
20	Init State					
21	Titulos					
22	[Alto, Bajo, El Alto]					
23	End Prototype					

Figura 10 Las ventanas típicas de un prototipo. La ventana superior es la de la aplicación propiamente dicha; la ventana inferior es la que muestra los datos almacenados en la base de datos en cada momento.



de aplicación tiene una barra de menú cuyo contenido puede ser definido por el ingeniero. Los formularios que se hayan dibujado pueden activarse seleccionando el menú y la opción de menú apropiadas. Una vez que se elige la opción de menú, el formulario correspondiente se dibuja sobre el fondo de la ventana de aplicación (Figura 11) o como un cuadro de diálogo. Luego, el usuario puede ingresar datos, picar en los botones, seleccionar ítem de una lista desplegable, etc. Dependiendo de los ejemplos que se hayan definido en la planilla de cálculo y de la entrada del usuario, en algún momento el prototipo producirá alguna salida y/o algún cambio de estado (que se podrá apreciar en la ventana de estado).

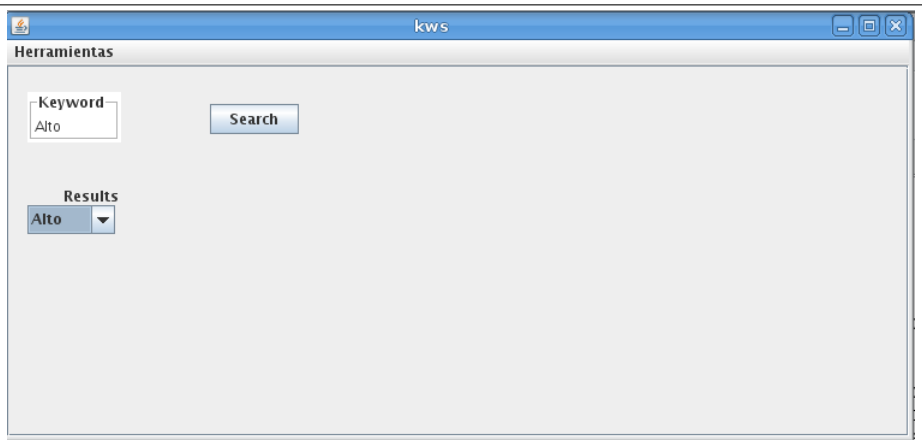
2.2.3. Cómo usar Repose

En esta sección explicamos con detalle cómo desarrollar un prototipo[11]. Cada prototipo se desarrolla en cinco pasos:

1. Modelar el prototipo

[11] En todo lo que sigue se asume que su versión de OpenOffice.org está en castellano.

Figura 11 El usuario puede activar un formulario seleccionándolo en la barra de menús, y luego puede probar los ejemplos especificados en la planilla de cálculo.



2. Dibujar la interfaz de usuario (GUI)
3. Definir el comportamiento del prototipo
4. Compilar el prototipo
5. Ejecutar el prototipo; es decir, permitir que los interesados jueguen con el prototipo

En los párrafos que siguen se explica cada uno de estos pasos.

Modelar el prototipo. Modelar un prototipo involucra:

- Pensar el prototipo en términos de formularios y pensar cada formulario en términos de datos de entrada, de estado y de salida.
- Definir los formularios del prototipo, lo cual a su vez involucra:
  - Definir los datos de entrada para ese formulario (nombre y tipo de control de interfaz).
  - Definir los datos de estado para ese prototipo (nombre). Los datos de estado son globales para todos los prototipos (es decir, todos los prototipos comparten los datos de estado).
  - Definir los datos de salida para ese formulario (nombre y tipo de control de interfaz).

Los ingenieros solo deben escribir esta información en algún lugar o simplemente tenerla en mente.

Dibujar la interfaz de usuario (GUI). Las GUIs de los prototipos se dibujan usando OpenOffice.org Draw. Asumimos que el lector es capaz de usar este tipo de programas. Al dibujar GUIs para los prototipos de Repose, deben tenerse en cuenta las siguientes notas.

- Crear un archivo odg[12] por cada formulario del prototipo.

[12]odg es la extensión estándar de los archivos generados por OpenOffice.org Draw.

La página del archivo representa la pantalla de la computadora. Por lo tanto, el ingeniero debe distribuir en la página los controles de interfaz tal y como desea que se vean en la pantalla, teniendo en cuenta los puntos que siguen.

Cada control de interfaz corresponde a un dato de entrada, estado o salida.

- Llevar la página de cada archivo odg a orientación horizontal y poner los márgenes en cero.
- Los controles de interfaz se dibujan usando algunas de las *Formas Básicas* y las *Formas de Símbolos* de OpenOffice.org Draw. Estas figuras geométricas se obtienen pulsando los botones y de la barra de dibujo, como se muestra en la Figura 12.

  Además, el fondo de cada forma geométrica debe llenarse en un color diferente dependiendo de si es un dato de entrada (Verde claro), estado (Amarillo) o salida (Rojo claro). Es importante seleccionar específicamente los colores con esos nombres pues de lo contrario Repose no funcionará correctamente.

  La Tabla 3 muestra la relación entre figuras geométricas, controles de interfaz de usuario y colores actualmente soportados por Repose. Observar que no todos los controles pueden usarse para representar datos de entrada, estado o salida. Por ejemplo, los *checkbox* solo pueden usarse para datos de entrada.

  Un dato es de entrada cuando lo provee el usuario; estos incluyen a los botones que pulsa el usuario.

  Los datos de estado son aquellos que se almacenan en la base de datos. Por lo tanto una de las funciones principales de los formularios es convertir datos de entrada en datos de estado.

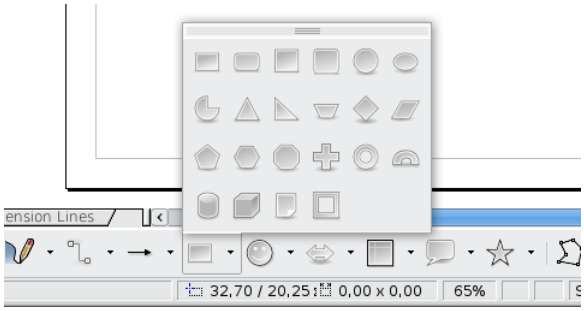
  Un dato es de salida cuando es producido por el sistema.

  Cuando en un formulario se incluye en Amarillo un dato de estado significa que el usuario podrá seleccionarlo como un dato de entrada. Esto facilita al ingeniero tomar datos que ya están en el sistema para que el usuario seleccione los que desea.

  Las listas editables (es decir los octógonos) permiten que el usuario ingrese o elimine de a un valor por vez. Para ello se provee de forma automática de dos botones: Ins, para insertar valores en la lista (en su lugar se puede pulsar Enter); y Sup, para eliminar de la lista el valor seleccionado. Por el contrario, las listas desplegables (es decir los hexágonos) no son editables: solo se pueden ver sus valores (cuando son datos de salida) o se puede seleccionar uno de ellos (cuando son datos de entrada o de estado).
- Dentro de cada figura geométrica escribir el nombre con que se quiere etiquetar cada control una vez que el prototipo esté en funcionamiento, como se muestra en la Figura 13.
- Cuando el prototipo está en ejecución, los formularios pueden mostrarse sobre el fondo de la ventana de aplicación o sobre un cuadro de diálogo. Para lograr el primer efecto simplemente se dibujan los controles de interfaz sobre la página del archivo odg. En cambio para lograr el segundo efecto se debe dibujar primero un Square Bevel sobre la página del archivo, y luego los controles de interfaz dentro de aquel, como se muestra en la figura 14.

  Los Square Bevel pueden tener cualquier color. No se debe incluir una etiqueta para los Square Bevel.

Figura 12 Para dibujar los controles de interfaz deben usarse las Formas Básicas y las Formas de Símbolos de la barra de dibujo. Aquí solo se muestran las primeras.



Control	Figura	Nombre y Grupo	Entrada	Estado	Salida
Caja de texto		Rectángulo Formas Básicas			
Botón		Rectángulo redondeado Formas Básicas			
Lista desplegable		Hexágono Formas Básicas			
Lista editable		Octógono Formas Básicas			
Check box		Papel Formas Básicas			
Cuadro de diálogo		Square Bevel Símbolos			

Tabla 3: Controles gráficos, colores y figuras geométricas. Los cuadros de diálogo se usan para organizar los otros controles de interfaz por lo que no son de entrada, de estado ni de salida. Los colores permitidos son: Verde claro, Amarillo y Rojo claro.

Figura 13 Los nombres dados a las figuras geométricas se usan más tarde para etiquetar el control de interfaz correspondiente.

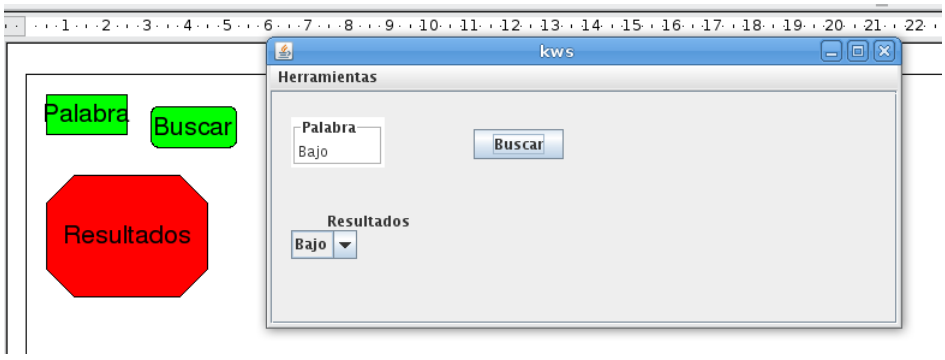


Figura 14 Para indicar que un formulario debe ser presentado como un cuadro de diálogo, y no en el fondo de la ventana de aplicación, se deben dibujar sus controles de interfaz dentro de un Square Bevel.

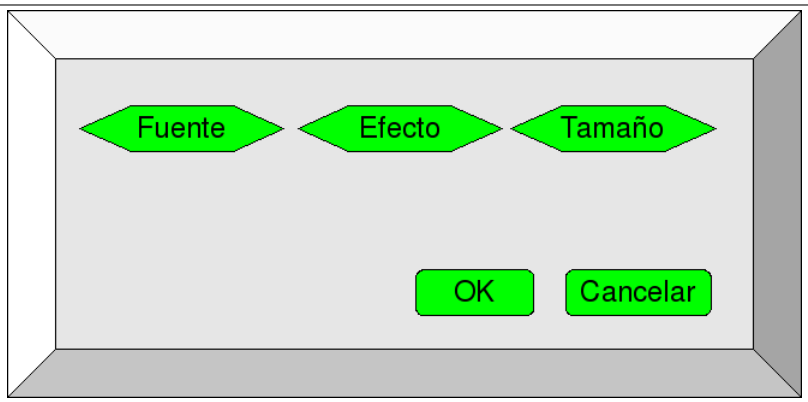
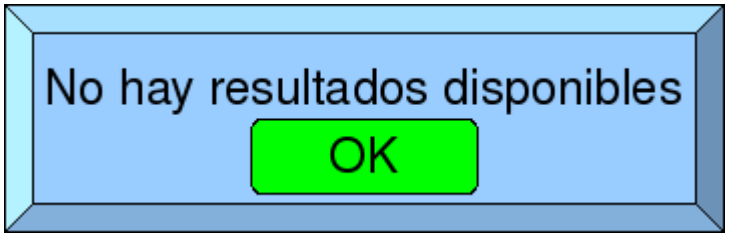


Figura 15 En este ejemplo se muestra el uso de cajas de texto para mostrar un cartel de error dentro de un Square Bevel, es decir de un cuadro de diálogo.



- Se puede escribir texto libre dentro de cada formulario, como se puede ver en la Figura 15. Esto se logra utilizando la herramienta de OpenOffice.org para escribir cajas de texto (T). Este texto no es dato y no es posible modificarlo durante el funcionamiento del prototipo. Es útil para escribir carteles de error, ayuda, comentarios aclaratorios, títulos, etc. El texto se dibuja siguiendo las mismas premisas que para los otros controles de interfaz.
- Al ubicar cada forma en la hoja, tener en cuenta lo que sigue:
  - Repose no tiene en cuenta el tamaño de las formas básicas.
  - Repose tiene en cuenta el tamaño de los Square Bevels (que se usan para mostrar cuadros de diálogo).
  - Ubicar los dibujos cerca del ángulo superior izquierdo de la hoja, pues de lo contrario Repose generará un formulario con barras de desplazamiento.
  - Repose tiene en cuenta únicamente las coordenadas del ángulo superior izquierdo de cada forma básica.
  - El ancho de cada forma será el suficiente como para poder acomodar el valor más ancho que se use en los ejemplos para esa variable. Solo tendrá en cuenta las dimensiones de los cuadros de diálogo, es decir de los Square Bevels.
- En resumen, el usuario debe dibujar y nombrar una figura geométrica por cada dato de acuerdo al control de interfaz con que se quiera representar esa dato en el formulario.

- Guardar los archivos en formato odg en el mismo directorio donde se haya instalado Repose.

Definir el comportamiento del prototipo. El comportamiento del prototipo se define usando OpenOffice.org Calc. Se asume que el lector sabe cómo usar este tipo de aplicaciones. Al definir el comportamiento de un prototipo se deben tener en cuenta las siguientes notas.

- El comportamiento de todo el prototipo se especifica en una planilla de cálculo.
- El comportamiento del prototipo se define especificando el comportamiento asociado a cada formulario y cómo los formularios se relacionan entre sí (más precisamente cómo un formulario invoca a otros formularios).
- El comportamiento de cada formulario se especifica por medio de uno o más ejemplos de cómo se debe usar ese formulario.
- El esqueleto de esta planilla de cálculo puede generarse a partir de todos los archivos odg, aunque también puede crearse manualmente. Para generarla a partir de los archivos odg se debe usar el siguiente comando –solo podrá ejecutarlo desde el directorio de Repose.

  ```
  java GenOds archivo.ods archivo_1.odg ... archivo_n.odg
  ```

  Es decir que se debe indicar el nombre del archivo ods a crear a partir de los archivos odg que se pasan como parámetros. Cada archivo odg debe contener la definición gráfica de un formulario del prototipo.

  Una vez que el archivo ods ha sido creado se debe usar OpenOffice.org Calc para modificarlo y completarlo como se indica más abajo.

  Si no se usa el comando anterior se debe usar OpenOffice.org Calc para crear el archivo como se indica más abajo.

  ¡¡Importante!! Si se usa el comando anterior luego de que el archivo ods ha sido modificado por el ingeniero, se perderán todos los cambios. Es decir el comando crea un archivo desde cero sin considerar los datos que el usuario haya incorporado a la planilla anterior.

En lo que sigue se asume que el usuario va a crear el archivo ods manualmente. Como el comando `GenOds` ejecuta automáticamente varios de los pasos indicados más abajo, el lector puede obviarlos. De todas formas recomendamos leerlos a todos porque el archivo generado automáticamente no contiene todos los detalles. Más aun, recomendamos que la primera vez utilice el comando `GenOds` y complete los detalles que faltan manualmente.

¡¡Importante!! Recordar que Repose es aun un programa experimental por lo que no controla errores del usuario. Por lo tanto poner mucha atención en seguir los pasos al pié de la letra puesto que de lo contrario Repose no funcionará correctamente.

La planilla para describir el comportamiento de un prototipo se puede confeccionar en tres pasos:

1. Describir el comportamiento del primer formulario
2. Describir el comportamiento de más formularios
3. Describir los datos iniciales para los formularios

A continuación se detallan cada uno de estos pasos. Complemente lo que sigue leyendo la Figura 9.

	A	B	C	D
3	Requirement			
4	ID	Priority	Quality	Stakeholders
5				
6	Description			

Tabla 4: Esta estructura debe repetirse para cada formulario de manera tal de poder describir cada requerimiento del sistema.

Comportamiento del primer formulario.

1. Abrir una planilla de cálculo.
2. En la celda A1 escribir `Begin Prototype`.
3. En la celda A2 escribir `Form` o `Form (NoMenu)` (notar que entre `Form` y `(NoMenu)` hay un espacio en blanco). El primer caso se utiliza para los formularios que deben aparecer en algún menú de la barra de menús; el segundo se utiliza para los formularios que no deben aparecer en ningún menú, sino que son ejecutados desde otros formularios (por ejemplo una ventana que comunica un error).
4. En la celda B2 escribir el nombre de una opción de menú para uno de los formularios del prototipo y, dejando un espacio en blanco, el menú entre paréntesis. Por ejemplo `Guardar (Archivo)`. De esta forma cuando se ejecute el prototipo, el formulario se mostrará al seleccionar el nombre de menú y la opción indicadas. Si no se indican estos datos, al menos se debe indicar un nombre para el formulario. En ese caso el formulario se podrá ejecutar seleccionando el menú `Function` y la opción con el nombre del formulario. `GenOds` utiliza el nombre del archivo odg como nombre para el formulario.
5. En la celda C2 escribir el nombre del archivo odg que contiene la definición gráfica del formulario.
6. Las filas 3, 4, 5 y 6 se destinan para describir coloquialmente el requerimiento asociado al formulario. La Tabla 4 muestra la distribución de las celdas en esas filas.

   Debajo de la columna ID se debe ingresar el identificador para el requerimiento; debajo de Priority la prioridad del requerimiento; debajo de Quality la calidad del requerimiento; y debajo de Stakeholders los interesados que participaron en la definición y validación del requerimiento.

   En las celdas a la derecha de Description se debe ingresar la descripción coloquial del requerimiento. Este es tal vez el dato más importante de cada formulario.
7. En la celda A7 escribir `Label`.
8. En la fila 7, comenzando por la columna B, y a razón de un dato por columna, escribir la definición de cada dato (de entrada, de estado y de salida) que usa el formulario. La definición de cada dato consta de:

   clase debe ser uno de `varin`, para las variables de entrada; `varst`, para las variables de estado; y `varout`, para las variables de salida.

   nombre debe ser uno de los nombres usados en el archivo odg o un nombre nuevo.

Se pueden agregar datos de estado pero no se pueden agregar datos de entrada o salida, respecto de los que están en el archivo odg. Todos los datos consignados en el archivo odg deben estar en la definición del formulario que se hace en la planilla.

Hay formularios que se utilizan para cambiar datos en la base de datos, es decir cambiar el estado del sistema. La forma que provee Repose para que el ingeniero indique un cambio en la base de datos es mediante *datos de estado primados*. Es muy simple: para cada dato de estado que sea modificado por el formulario se agrega una columna con el mismo nombre pero decorado al final con un apóstrofe. Por ejemplo, `usuarios` y `usuarios'`. La columna sin apóstrofe representa la base de datos antes de ejecutar el formulario y la columna con apóstrofe representa la base de datos luego de ejecutar el formulario. En resumen: se indica cómo está la base antes de ejecutar y cómo queda después de ejecutar, en dos columnas diferentes con el mismo nombre pero uno de ellos decorado con apóstrofe.

En aquellos formularios donde no se modifica la base de datos no es necesario definir las columnas decoradas con apóstrofe.

9. Repetir este paso cualquier número de veces, solo se debe incrementar el número de fila y tener en cuenta las indicaciones que se dan antes del paso siguiente. En la fila 8, comenzando por la columna B, escribir un valor en cada columna de manera tal que todos los valores de la fila representen una posible ejecución del formulario. Escribir *todos* los valores incluyendo los de salida ya que Repose no calculará nada por sí mismo.

   A cada una de estas filas la llamamos *ejemplo*.

   Tenga en cuenta que el tipo de los valores de un dato depende hasta cierto punto del control de interfaz que se eligió para representarlo. Las restricciones son las siguientes:

   Botones. Los valores para los botones pueden ser `Action`, `NoAction` o el nombre de una `Label` de otro formulario –ver el punto 10.

   Check boxes. Los valores pueden ser 0, que significa que el *check box* no se ha seleccionado; y 1, que significa que se ha seleccionado.

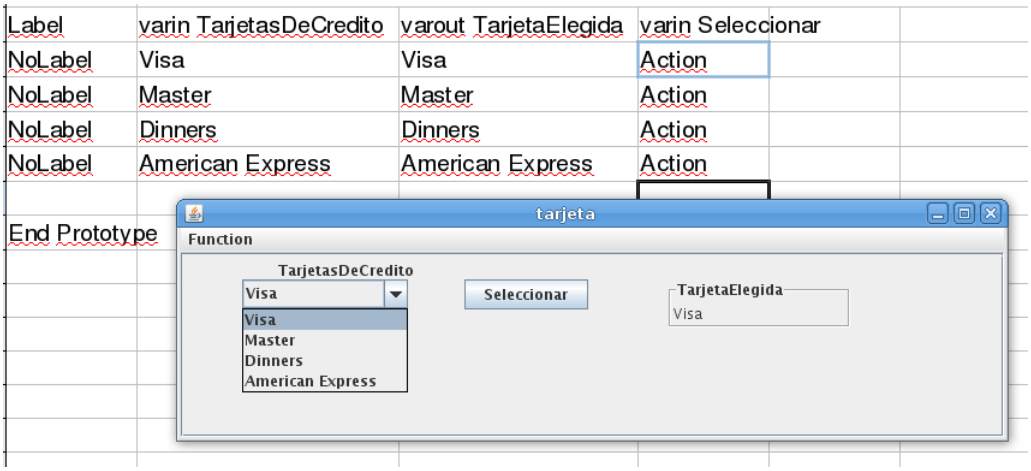
   Listas desplegables. Los valores para las listas desplegables pueden ser cualesquiera pero siempre del mismo tipo (números o cadenas de caracteres). Si la lista desplegable es un dato de entrada, la lista de valores que se le muestre al usuario se armará con todos los valores distintos dados en la columna correspondiente del mismo formulario (Figura 16). De esta forma es simple armar la lista y a la vez especificar qué hay que hacer cada vez que se elige uno de los valores mostrados.

   Listas. Si un dato se representa con un octógono (lista editable) los valores que se pueden ingresar en la columna correspondiente deben ser listas de valores. Es decir deben tener la forma `[val_1, ..., val_n]`, donde cada `val` es una cadena de caracteres que no contiene una coma. También es posible indicar que la lista está vacía ingresando `[]`.

   Otros valores. Para las otras columnas los valores pueden ser números, cadenas de caracteres o listas de valores, pero siempre deben ser del mismo tipo. Las listas de valores se encierran entre corchetes y cada valor se separa con una coma. Las listas de valores son buenas para mostrarlas en listas desplegables.

10. La columna `Label` se utiliza para invocar un formulario desde otros formularios. Los formularios pueden invocarse al pulsar algún botón. Esto se consigue en dos pasos:

Figura 16 La lista TarjetasDeCredito es un dato de entrada por lo que la lista de valores se toma de los ejemplos.



(a) poniendo un nombre cualquiera en la columna `Label` del formulario al cual se quiere invocar, y (b) poniendo ese mismo nombre como `Action` de un botón del formulario desde el cual se quiere invocar al primero. Ver el ejemplo de la Figura 9 entre los formularios Buscar y Error.

Las celdas de la columna `Label` no pueden quedar en blanco. Por lo tanto, se debe escribir `NoLabel` si no se van a utilizar.

Uno de los usos más habituales para las etiquetas es poder mostrar ventanas de error, de ayuda, cuadros de diálogo más específicos, etc. que se invocan al pulsar diferentes botones en otros formularios.

Comportamiento de los otros formularios. Para especificar el comportamiento de más formularios se deben repetir los pasos anteriores excepto los dos primeros. Cada formulario se puede separar del anterior dejando una fila en blanco.

Datos iniciales para los formularios. Hemos visto que algunos formularios usan datos de estado que representan a la base de datos del sistema. Repose exige que se dé un valor inicial para cada columna que representa a la base de datos. Estos valores iniciales se establecen en una sección que se introduce dejando una fila en blanco luego del último ejemplo del último formulario. Digamos que la fila en cuestión es la N, entonces:

1. En la celda AN escribir `Init State`.
2. En la fila $N+1$, comenzando en la columna A, y a razón de un dato de *estado* por columna, escribir el nombre del dato.
3. En la fila $N+2$, escribir el valor inicial para cada dato.
4. En la celda A$N+3$ escribir `End Prototype`.

Esta sección debe escribirse siempre a menos que en ningún formulario se haya utilizado variables de estado.

Además de las variables de estado, en esta sección de la planilla pueden darse más valores para las variables de entrada que se representan como listas desplegables. En alguna columna de la fila $N+1$ se escribe el nombre de la variable y en la celda inferior se escriben los valores adicionales para mostrar al usuario, encerrados entre corchetes y separados por comas. Estos valores se adicionan a los valores que se hayan encontrado en el formulario donde se usa la variable.

¡¡Importante!! No olvide guardar el archivo en el mismo directorio donde está instalado Repose.

2.2.4. Compilar, ejecutar y modificar el prototipo

Una vez que se ha terminado de especificar el comportamiento del prototipo se debe abrir una ventana de línea de comandos, ir al directorio donde se haya instalado Repose y ejecutar los siguientes comandos:

1. `java ProtoGen archivo.ods archivo.java`
   Genera el prototipo a partir de la planilla `archivo.ods` que es donde se ha especificado el comportamiento del prototipo, en tanto que `archivo.java` es el nombre que el usuario le debe dar al prototipo.
   El comando escribe diversos mensajes en la pantalla. Si el último mensaje es `The prototype has been generated. Compile it with: javac archivo.java`, significa que el prototipo se ha generado con éxito; en caso contrario hay algún error en la planilla o en los archivos odg –por el momento Repose es muy poco informativo al respecto por lo que sugerimos mirar con atención la planilla tratando de descubrir algún error; los mensajes en la pantalla pueden ayudar.
2. `javac archivo.java`
   Si el prototipo se ha generado con éxito el comando anterior crea un programa Java ejecutable. Si el prototipo ha sido generado con éxito el comando anterior no debería producir ningún error –si lo hace significa que Repose tiene algún error.
3. `java archivo`
   Esta es la forma en que el prototipo debe ser ejecutado para que el cliente pueda jugar con él.

Una vez que el prototipo esta ejecutando se lo puede usar como a cualquier otro programa pero siempre se debe recordar que los únicos valores de entrada que aceptará son lo que están definidos en en los ejemplos. Además, recuerde que junto a la ventana de aplicación se abre también una ventana que muestra el estado actual de la base de datos a medida que se usa el prototipo.

Si hay que modificar el prototipo:

- Cerrar la ventana de aplicación –la ventana de la base de datos se cerrará automáticamente.
- Modificar cualquiera de los archivos odg agregando o borrando variables, o la representación gráfica de cualquiera de ellas.
- También se pueden crear otros archivos odg para agregar más formularios al prototipo.
- Modificar el archivo ods agregando, borrando o modificando ejemplos y/o agregando o borrando formularios (es decir, secciones `Form`).

- Volver a ejecutar los tres comandos anteriores en el mismo orden.

Si solo se quiere ejecutar el prototipo sin ninguna modificación (por ejemplo para mostrárselo a otro interesado) se debe ejecutar el último de los comandos anteriores.

Recuerde que el comando `java GenOds archivo.ods archivo_1.odg ... archivo_n.odg` le permite generar el esqueleto de la planilla donde especificar el comportamiento del prototipo cuyos formularios se especifican en los archivos odg `archivo_1.odg ... archivo_n.odg`. También recuerde que si la planilla ya fue generada la nueva planilla sobreescribirá a la anterior y usted perderá cualquier modificación que haya hecho.

Sea lo que fuere que se haga con Repose no hay que olvidar que el objetivo principal es

Obtener una lista consistente, completa y
acordada con el cliente de requerimientos funcionales

entonces siempre se debe poner mucho esfuerzo y cuidado en

Escribir el requerimiento asociado a cada formulario
y mantenerlo consistente con los ejemplos

porque es la información clave de todo el proceso.

Ejercicios

Ejercicio 3. Utilice Repose para prototipar al menos cinco de los requerimientos relevados para la farmacia mencionada en el problema 1 de la página 22.

Ejercicio 4. Utilice Repose para prototipar al menos cinco de los requerimientos relevados para el restaurante mencionado en el problema 2 de la página 22.

---

3. Obtención y Validación de Requerimientos No Funcionales

Los requerimientos no funcionales, atributos de calidad o cualidades del software suelen ser la principal causa de grandes, complejos y costosos cambios a los sistemas de software. Usualmente no se los tiene en cuenta y cuando se lo hace las descripciones de estos requerimientos suelen ser confusas y ambiguas.

Los atributos de calidad y la funcionalidad son ortogonales pues si esto no fuera así la elección de una determinada funcionalidad dictaría el nivel de seguridad, desempeño, disponibilidad o usabilidad. Por el contrario, hasta cierto punto es posible elegir un nivel de cada una de ellas independientemente de la funcionalidad. Los sistemas de software se descomponen en elementos no porque la descomposición sea necesaria para alcanzar cierta funcionalidad sino

Figura 17 Los elementos que constituyen un escenario de un atributo de calidad.

porque es necesaria para alcanzar ciertos requerimientos no funcionales. De hecho, si la función fuera el único requerimiento el software podría ser un único y enorme módulo sin ninguna estructura interna.

Existen varios problemas al intentar obtener los requerimientos no funcionales de un sistema [**?**]:

1. Hay demasiados atributos de calidad y los usuarios desconocen la mayoría.
2. Las definiciones que se dan de los atributos por lo general no son operacionales. Carece de significado decir "el sistema debe ser modificable" porque cualquier sistema es modificable.
3. Es discutible a qué atributo de calidad pertenece un concepto. Una falla del sistema, ¿pertenece a disponibilidad, seguridad o usabilidad?
4. Cada comunidad ha desarrollado su propio vocabulario para describir los atributos de calidad.

El primer problema lo atacamos en la sección 3.1 y los restantes en la sección 3.2.

3.1. *Checklist* de atributos de calidad

La Tabla 5 puede ser considerada como un *checklist* muy completo para determinar los atributos de calidad para un sistema. En una o varias de las entrevistas con el cliente los ingenieros de requerimientos deben consultarle al cliente cuál o cuáles de esos atributos de calidad desea para su sistema. Seguramente será necesario explicarle brevemente qué significan. Cuando el cliente manifiesta interés en uno de los atributos de calidad se pasa a la etapa de generación de escenarios para ese atributo de calidad, como se describe en la sección 3.2.

3.2. Escenarios de atributos de calidad

Definición 6 (Escenario de atributo de calidad)**.** *Un escenario de un atributo de calidad es un requerimiento no funcional específico sobre ese atributo de calidad.*

Los escenarios se componen de seis elementos como se puede apreciar en la Figura 17 [**?**]:

accessibility	accountability	accuracy
accesibilidad	responzabilización	precisión
adaptability	administrability	affordability
adaptabilidad	administrabilidad	financiabilidad
agility	auditability	availability
agilidad	auditabilidad	disponibilidad
credibility	standards compliance	process capabilities
credibilidad	cumplimiento de estándares	capacidad del proceso
compatibility	composability	configurability
compatibilidad	composicionabilidad	configurabilidad
customizability	degradability	demonstrability
personalización	degradabilidad	demostrabilidad
dependability	deployability	distributability
dependabilidad	desplegabilidad	distritutibilidad
durability	evolvability	extensibility
durabilidad	evolucionabilidad	extensibilidad
fidelity	flexibility	installability
fidelidad	flexibilidad	instalabilidad
integrity	interchangeability	interoperability
integridad	intercambiabilidad	interoperabilidad
learnability	maintainability	manageability
aprendabilidad	mantenibiliad	gerenciabilidad
mobility	modifiability	modularity
movilidad	modificabilidad	modularidad
operability	portability	precision
operabilidad	portabilidad	precisión
predictability	performance	relevance
predicibilidad	desempeño	relevancia
reliability	repeatability	reproducibility
confiabilidad	repetibilidad	reproducibilidad
responsiveness	reusability	robustness
respuesta	reusabilidad	robustes
scalability	seamlessness	serviceability (a.k.a. supportability)
escalabiliad	sin consturas	serviciabilidad
securability	simplicity	stability
segurabilidad	simplicidad	estabilidad
survivability	sustainability	tailorability
supervivenciabilidad	sutentabilidad	acompañabilidad
testability	timeliness	understandability
testeabilidad	puntualidad	entendibilidad
usability	nomadicity	recoverability
usabilidad	nomadicidad	recuperabilidad
internationalization		
internacionalización		

Tabla 5: Atributos de calidad. Fuente: `http://en.wikipedia.org/wiki/List_of_System_Quality_Attributes`.

Origen del estímulo. Es la entidad (un humano, una computadora, otro sistema, etc.) que produce el estímulo. Es importante tener en cuenta esta parte porque los sistemas suelen reaccionar de forma diferente dependiendo de dónde proviene el estímulo. Por ejemplo, si el atributo de calidad que se está considerando es seguridad, entonces dos escenarios posibles son: "si un dato proviene de un componente confiable el sistema no lo valida" y "si un dato proviene de un componente no confiable lo valida".

Estímulo. Un estímulo es una condición que debe ser considerada cuando llega al sistema.

Entorno. El estímulo se da dentro de ciertas condiciones.

Artefacto. Uno o varios elementos del sistema, o incluso todo el sistema. Usualmente es todo el sistema aunque se suelen distinguir componentes tales como servidor web, servidor de aplicaciones, DBMS, etc.

Respuesta. La respuesta es la actividad que se lleva a cabo luego del arribo del estímulo.

Medida de la respuesta. Cuando la respuesta se da debe ser medible de alguna manera de forma tal que el requerimiento pueda ser testeado.

Existen dos clases de escenarios: los *escenarios generales*, aquellos que son independientes de un sistema particular y pueden, en consecuencia, aplicarse a cualquier sistema; y los *escenarios concretos*, aquellos que son específicos del sistema que se está construyendo, es decir los requerimientos no funcionales para ese sistema. Por ejemplo, un escenario general es "Llega un pedido de cambio de funcionalidad y el cambio de debe ser efectuado en un momento determinado dentro del proceso de desarrollo y dentro de un período de tiempo determinado" y un escenario concreto de ese escenario general es "Llega un pedido para que la aplicación Web EasySell soporte un nuevo navegador dentro de dos semanas". Notar que para un sistema dado un escenario general puede dar lugar a cero, uno o más requerimientos no funcionales (escenarios concretos) lo cuales pueden no necesitar todas las partes de un escenario. Los escenarios concretos, y en particular las respuestas y sus medidas, deben tener el detalle suficiente como para que sean significativos para el equipo de arquitectura y diseño como para que sea posible testear si el sistema los verifica o no.

Una vez que el cliente ha manifestado interés en un atributo de calidad del *checklist* de la sección 3.1, los ingenieros de requerimientos deben:

1. Definir operacionalmente el atributo de calidad por medio de uno o más escenarios generales (si no cuentan con tal definición).
2. Presentarle los escenarios generales de ese atributo al cliente.
3. Derivar cero o más de esos escenarios generales en escenarios concretos según los requerimientos específicos del cliente. Estos son los requerimientos no funcionales del cliente.

En las secciones que siguen se definen operacionalmente seis atributos de calidad por medio de sendas colecciones de posibles escenarios generales. Más aun, para cada atributo de calidad se presenta una tabla con posibles valores genéricos para cada una de las partes de un escenario: combinando esos valores genéricos se obtienen escenarios generales y especificando valores concretos para los valores genéricos, se obtienen escenarios concretos. En todas las campañas de ingeniería de requerimientos no es necesario generar todos los escenarios generales a partir de las tablas. Las tablas sirven como *checklists* para cada atributo de calidad.

Elemento	Posibles valores genéricos
Origen	Interno al sistema; externo al sistema.
Estímulo	Error: omisión, caída, *timing*, respuesta.
Artefacto	Procesadores, canales de comunicación, unidades de almacenamiento, procesos.
Entorno	Operación normal; modo degradado (i.e. menos servicios).
Respuesta	El sistema debe detectar el evento y hace una o más de las siguientes: • Registrar el evento. • Notificar a los interesados, incluyendo usuarios y otros sistemas. • Deshabilitar las fuentes que causaron la falla siguiendo las reglas definidas para tal caso. • Estar no disponible por un cierto intervalo de tiempo que depende de la criticidad del sistema. • Continuar operando en modo normal o degradado.
Medida	Intervalo de tiempo que el sistema debe estar disponible Tiempo de disponibilidad Intervalo de tiempo que el sistema puede operar en modo degradado Tiempo de reparación

Tabla 6: Generación de escenarios generales para disponibilidad.

3.2.1. Disponibilidad

La disponibilidad refiere a las fallas de un sistema y sus consecuencias. Una falla ocurre cuando el sistema no brinda más un servicio consistente con su especificación. Entre las áreas involucradas tenemos: cómo se detectan las fallas, qué tan frecuente puede ocurrir una falla, qué pasa cuando la falla se da, cuánto tiempo puede estar el sistema sin funcionar, cómo se pueden prevenir las fallas, etc. El tiempo necesario para reparar el sistema es un tema que está directamente asociado a la disponibilidad.

La Tabla 6 presenta los posibles valores genéricos para cada una de las partes de un escenario de disponibilidad.

Ejemplos de escenarios generales. Los siguientes son ejemplos de escenarios generales para disponibilidad:

1. Un dato incorrecto proveniente desde el exterior produce la caída de un proceso que estaba operando normalmente ante lo cual se debe notificar a los interesados y el sistema debe ser reparado en un cierto tiempo.

2. Uno de los discos de un servidor se llena inesperadamente, entonces se registra el evento y se pasa a modo degradado hasta que se libera espacio.

3. Una fuente externa al sistema envía un mensaje no anticipado durante el funcionamiento normal de un proceso. Se debe informar a un operador y continuar operando normalmente. No debe haber tiempo sin procesamiento.

Ejemplos de escenarios concretos. A continuación se presentan escenarios concretos a partir de los generales enunciados más arriba:

1. Si un navegador envía un mensaje fuera de protocolo lo que produce que el servidor Web quede fuera de servicio, se deberá enviar un email a los administradores los que deberán levantar el servidor dentro de la hora de producido el suceso.
2. Si debido a una falla en la rotación de los archivos de bitácora se llena uno de los discos del *firewall*, entonces se registra el evento en syslog y se cancela el registro de auditoría hasta el día hábil siguiente.
3. Si a consecuencia de una cantidad inesperada de transacciones se llena el disco de datos de la base de datos, se deberá registrar el evento en la consola del administrador y se comenzará a utilizar el disco reservado a aplicaciones, hasta que se libere espacio.

Ejercicios

Ejercicio 5. Derivar un escenario concreto del escenario general 3 que hable de un sistema cliente-servidor.

Ejercicio 6. Escribir un escenario general para disponibilidad que se relacione con el *timing* de un evento.

Ejercicio 7. Derivar un escenario concreto a partir del escenario general anterior que esté relacionado con el arribo de un mensaje de correo electrónico durante horarios no laborables.

3.2.2. Modificabilidad

La modficabilidad trata sobre el costo del cambio. Esto trae aparejado dos cuestiones:

1. Qué es lo que puede cambiar (el artefacto)? Los cambios pueden darse sobre cualquier aspecto del sistema aunque los más comunes son: las funciones que computa el sistema, las plataformas sobre las cuales ejecuta, el entorno con el cual el sistema interactúa (otros sistemas, protocolos, etc.), las cualidades que el sistema exhibe, y su capacidad (número de usuarios soportados, número de operaciones simultáneas, etc.). Algunos de estos aspectos son tratados específicamente por otros atributos de calidad.
2. Cuándo se hace el cambio y quién lo hace (el entorno)? Lo más común es que los cambios se hagan directamente sobre el código, aunque no es la mejor práctica. También se debe considerar el caso en que los usuarios finales cambian configuraciones, instalan componentes desarrollados por terceros, etc. Además, para mejorar la forma en que se desarrolla se debería considerar cambiar la arquitectura o el diseño y que por lo tanto sean los diseñadores los que inician el cambio.

La Tabla 7 presenta los posibles valores genéricos para cada una de las partes de un escenario de modificabilidad.

Elemento	Posibles valores genéricos
Origen	Usuario final, desarrollador, administrador, gerencias.
Estímulo	Agregar, eliminar o modificar funcionalidad o atributo de calidad.
Artefacto	Interfaz de usuario, funcionalidad del sistema, plataforma, entorno; otro sistema que interopera con el sistema.
Entorno	En tiempo de ejecución, durante la compilación, durante el diseño.
Respuesta	Localizar los elementos de la arquitectura que deben ser modificados; efectuar la modificación sin afectar otra funcionalidad; testear la modificación; desplegar la modificación.
Medida	Costo en relación al número de elementos afectados, esfuerzo, dinero; grado en el que el cambio afecta a otras funciones o atributos de calidad.

Tabla 7: Generación de escenarios generales para modificabilidad.

Ejemplos de escenarios generales. Los escenarios que planteamos son los siguientes:

1. Un usuario quiere modificar la apariencia de la interfaz de usuario mientras usa el programa.
2. El usuario solicita una nueva versión del sistema con nueva funcionalidad, de forma inmediata o planificada.
3. Se solicita que se cambie la plataforma del sistema respetando la metodología de desarrollo vigente y a un costo mínimo.

Ejemplos de escenarios concretos. Los siguientes son escenarios concretos de los escenarios generales planteados más arriba.

1. El usuario debe poder seleccionar la ubicación de las barras de herramientas mientras usa el programa.
2. Un usuario final solicita un nuevo listado que debe ser implementado entregando una nueva versión del sistema dentro de las 48 horas hábiles de efectuado el pedido; el listado debe haber sido testeado y la documentación debe actualizarse.
3. Un usuario final solicita un nuevo listado que debe ser implementado en un par de horas sin detener el sistema.
4. Las resoluciones del BCRA de carácter urgente deben incorporarse al sistema dentro de las 48 horas sin necesidad de actualizar la documentación, ni de testeo por lo que debe proveerse una guardia de desarrollo hasta tanto la modificación se estabilice. Las resoluciones planificadas deben incorporarse siguiendo la metodología de desarrollo pero minimizando las modificaciones sobre módulos existentes. En cualquier caso estas modificaciones entrañan la entrega de una nueva versión del sistema y no pueden afectar su seguridad.
5. El gerente de desarrollo solicita que se adapte en seis meses la aplicación al nuevo servidor corporativo de bases de datos estudiando cuidadosamente el diseño actual, manteniendo

Elemento	Posibles valores genéricos
Origen	Una de una cantidad importante de fuentes independientes, posiblemente desde dentro del sistema.
Estímulo	Arriba un evento periódico; arriba un evento esporádico; arriba un evento estocástico.
Artefacto	Sistema
Entorno	Modo normal; modo sobrecargado.
Respuesta	Se procesan los estímulos; cambios en el nivel de servicio.
Medida	Latencia, *deadline*, capacidad de procesamiento[13], ritmo de pérdida, pérdida de datos.

Tabla 8: Generación de escenarios generales para desempeño.

la consistencia entre la documentación técnica y la implementación y testeando el sistema con los mismos casos de prueba que se usaron para la primera versión.

Ejercicios

Ejercicio 8. Derivar un escenario concreto a partir del escenario general 1 que implique una modificación más compleja que la mencionada en el escenario concreto 1.

Ejercicio 9. Escriba un escenario general cuya respuesta implique volver a desplegar una parte del sistema pero en tiempo de ejecución.

Ejercicio 10. Derive un escenario concreto a partir del escenario general anterior que mencione la posibilidad de actualizar en tiempo de ejecución el cliente de un sistema cliente-servidor.

3.2.3. Desempeño

El desempeño tiene que ver con el uso del tiempo. Ocurren eventos (interrupciones, mensajes, pedidos de los usuarios, o el paso del tiempo) y el sistema debe responder a ellos. El desempeño trata básicamente de qué tanto tiempo demora el sistema en responder a esos eventos. Una de las cosas que hace el estudio del desempeño complejo es el número de los posibles orígenes de los eventos y los patrones de llegada.

La Tabla 8 presenta los posibles valores genéricos para cada una de las partes de un escenario de desempeño. A continuación se comentan brevemente esos valores.

Ejemplos de escenarios generales. Para desempeño presentamos un único escenario general.

1. Fuentes externas, transacciones, sistema, modo normal, procesar, latencia.

Ejemplos de escenarios concretos. Pero listamos tres escenarios concretos derivados de aquel escenario general.

1. Si los usuarios generan más de 1.000 transacciones por minuto durante funcionamiento normal, el sistema las debe procesar con una latencia promedio de dos segundos.
2. Si los proveedores generan más de 100 transacciones por minuto durante funcionamiento normal, el sistema las debe procesar con una latencia promedio de 1 segundo.
3. Si una caja que ha estado fuera de línea envía más de 100 transacciones por minuto durante el funcionamiento normal del sistema, las debe procesar con una latencia promedio menor a la empleada cuando la caja está en línea.

Ejercicios

Ejercicio 11. Escribir un escenario general que involucre un cambio en el nivel de servicio con pérdida de datos.

Ejercicio 12. Derive un escenario concreto a parir del escenario general anterior que corresponda a un sistema Web que recibe una cantidad inusual de pedidos.

3.2.4. Seguridad

La seguridad es una medida de la capacidad del sistema de resistir el uso no autorizado al mismo tiempo que sigue brindando servicios a los usuarios legítimos. Un intento por romper la seguridad se denomina *ataque*. La seguridad puede caracterizarse por medio de los siguientes atributos: no repudiación, confidencialidad, integridad, autenticidad, disponibilidad y auditoría.

La Tabla 9 presenta los posibles valores genéricos para cada una de las partes de un escenario de seguridad.

Ejemplos de escenarios generales. Presentamos los que siguen.

1. Los agentes externos que intenten acceder datos del sistema deben ser autenticados de forma tal que no puedan evitarlo.
2. Todos los intentos de conexión al sistema que presenten alguna identificación se deberán registrar de manera que exista alguna probabilidad de detectar un posible ataque.

Ejemplos de escenarios concretos. A los escenarios generales anteriores les corresponden al menos estos escenarios concretos.

1. Los vendedores que se conectan desde redes externas y que quieran consultar la base de ventas deberán ser autenticados por contraseña y certificado digital.
2. Los miembros del directorio que necesiten consultar desde el exterior el estado financiero de la empresa, se deberán autenticar mediante contraseña, certificado digital y huella digital.

Elemento	Posibles valores genéricos
Origen	Individuo o sistema que ha sido identificado, no ha sido identificado, se desconoce su identidad el cual es interno/externo, autorizado/no autorizado con acceso a recursos limitados, recursos ilimitados.
Estímulo	El origen intenta ver, modificar o eliminar datos, usar servicios del sistema, reducir la disponibilidad de los servicios del sistema.
Artefacto	Servicios del sistema; datos procesados por el sistema.
Entorno	En línea, fuera de línea, conectado, desconectado, conexión filtrada, o abierta.
Respuesta	Autentica al usuario; oculta la identidad del usuario; bloquea el acceso a los datos y/o servicios; otorga o revoca permisos para acceder a los datos y/o servicios; registra los accesos/modificaciones o los intentos de acceso/modificación de datos/servicios por identidad; almacena los datos en un formato no legible; reconoce un inexplicable pico en la demanda de servicios, e informa a un usuario o sistema, y restringe la disponibilidad de los servicios.
Medida	Tiempo/esfuerzo/recursos necesarios para evitar las medidas de seguridad con probabilidad de éxito; probabilidad de detectar un ataque; probabilidad de identificar los individuos responsables de los ataques o por el acceso/modificación de datos/servicios; porcentaje de servicios que quedan operando durante un ataque de negación de servicio; restauración de datos/servicios; nivel de daño a los datos/servicios y/o nivel de negación de servicio a usuarios legítimos.

Tabla 9: Generación de escenarios generales para seguridad.

3. Los clientes que accedan a través de la interfaz web para consultar su cuenta corriente, deberán autenticarse por contraseña.
4. Se deberá registrar en la base del IDS la dirección IP y la fecha de todos los intentos de conexión al servidor web seguro.
5. Se deberá enviar un mail diario al administrador de seguridad con la identidad de todos los usuarios que ingresen al sistema a través de la VPN, dejando constancia además de la fecha, hora y dirección IP de origen.

Ejercicios

Ejercicio 13. Escribir un escenario general que involucre atacantes que trabajan fuera de línea intentando leer información confidencial de la empresa utilizando los permisos de usuarios legítimos.

Ejercicio 14. Derivar un escenario concreto a partir del escenario general anterior que mencione los datos de un sistema de gestión.

3.2.5. Testeabilidad

La testeabilidad refiere a la capacidad de qué tan simple es demostrar las fallas del sistema por lo general a través de testing. Al menos el 40 % del costo total de desarrollar sistemas bien hechos corresponde al testing. Para que un sistema se pueda testear debe ser posible controlar el estado interno y las entradas ingresadas a cada componente, y poder observar sus salidas.

La Tabla 10 presenta los posibles valores genéricos para cada una de las partes de un escenario de testeabilidad.

Ejemplos de escenarios generales. Los escenarios generales para testeabilidad son los que se listan más abajo.

1. Desarrollador, finaliza unidad, componente del sistema, implementación, ambiente de testing, cobertura.
2. Se debe testear el sistema de forma tal que se hayan recorrido todos los requerimientos antes de poner el sistema en producción.
3. Integrador de incrementos, entrega del sistema, etapa del desarrollo, tiempo para preparar el ambiente de testing.

Ejemplos de escenarios concretos.

1. Cuando un desarrollador finaliza un módulo debe utilizar el ambiente de testing para testear cada una de sus subrutinas de manera tal que se ejecute el 80 % de las sentencias, no demorando más de medio día.

Elemento	Posibles valores genéricos
Origen	Desarrollador de unidad Integrador de incrementos Verificador de sistema Tester del cliente Usuario
Estímulo	Se ha completado un requerimiento, arquitectura, diseño, módulo o subsistema; se ha entregado el sistema.
Artefacto	Parte del diseño; parte del código; aplicación.
Entorno	En tiempo de diseño; en tiempo de implementación; en tiempo de compilación; en tiempo de despliegue.
Respuesta	Se tiene acceso al estado; se tiene acceso a las salidas; se dispone del ambiente de testing.
Medida	Porcentaje de sentencias ejecutables ejecutadas. Probabilidad de falla si existe un error. Tiempo para efectuar los tests. Longitud de la cadena de dependencia más larga en un test. Tiempo para preparar el ambiente de testing.

Tabla 10: Generación de escenarios generales para testeabilidad.

2. Los usuarios deberán testear el sistema comprobando cada uno de los requerimientos en los que ellos estuvieron involucrados. El testing lo efectuarán luego de que se entregue el sistema pero antes de pasarlo a producción. Se deberán proveer las aplicaciones necesarias para que los usuarios puedan monitorear el estado de la base de datos. Se dispondrán de 10 días calendario para esta tarea.
3. Los integradores del sistema deberán testear cada incremento antes de la entrega del sistema siempre y cuando el tiempo para preparar el ambiente de testing no exceda de un día.

Ejercicios

Ejercicio 15. Escribir un escenario general que se relacione con el testing de aplicaciones de las cuales no se dispone del código fuente.

Ejercicio 16. Derivar un escenario concreto a partir del escenario general anterior para testear funcionalmente aplicaciones Web.

3.2.6. Usabilidad

Usabilidad refiere a qué tan fácil es para el usuario llevar a cabo una tarea y la clase de ayuda que este recibe de parte del sistema. Se la puede descomponer en las siguientes áreas:

Elemento	Posibles valores genéricos
Origen	Usuario final
Estímulo	Aprender la funcionalidad del sistema; usar el sistema eficientemente; minimizar el impacto de los errores; adaptar el sistema; sentirse confortable.
Artefacto	Sistema
Entorno	En tiempo de ejecución o durante la configuración.
Respuesta	El sistema provee una o más de las siguientes respuestas: • Para soportar "aprender la funcionalidad del sistema": ayuda sensible al contexto; interfaz conocida o "estándar"; la interfaz es usable en un contexto desconocido. • Para soportar "usar el sistema eficientemente": composición de datos y/o comandos; reuso de comandos y/o datos ya ingresados; soporte para navegación eficiente dentro de la pantalla; diferentes vistas con operaciones consistentes; búsqueda; múltiples actividades simultáneas. • Para soportar "minimizar el impacto de los errores": deshacer, cancelar, recuperación de una caída del sistema, reconocer y corregir errores del usuario, recuperar contraseña olvidada, verificar los recursos del sistema. • Para soportar "adaptar el sistema": personalización, internacionalización. • Para soportar "sentirse confortable": mostrar el estado del sistema; trabajar al ritmo del usuario.
Medida	Duración de cada tarea, número de errores, número de problemas solucionados, satisfacción del usuario, incremento de los conocimientos del usuario, proporción entre operaciones exitosas y total de operaciones, tiempo/datos perdidos.

Tabla 11: Generación de escenarios generales para usabilidad.

- Aprender la funcionalidad del sistema.
- Usar el sistema eficientemente.
- Minimizar el impacto de los errores.
- Adaptar el sistema a las necesidades del usuario.
- Incrementar la confianza y la satisfacción del usuario.

La Tabla 11 presenta los posibles valores genéricos para cada una de las partes de un escenario de usabilidad.

Ejemplos de escenarios generales. Todos los escenarios generales de usabilidad tienen al usuario final como origen y al sistema por artefacto, por lo que omitimos esos parámetros.

1. Aprender funcionalidad, en tiempo de ejecución, ayuda sensible al contexto, incremento de los conocimientos del usuario.
2. Adaptar el sistema, durante la configuración, internacionalización, duración de la tarea.

Ejemplos de escenarios concretos. Mostramos dos escenarios concretos para cada escenario general.

1. El IDS debe proveer ayuda contextual para los botones relacionados con la administración de reglas de detección. La ayuda debe ser tan detallada como para que el usuario aprenda a aplicar esas reglas sin tener que volver a consultarla.
2. Cada vez que el usuario seleccione una opción de menú por primera vez se le presentará una ventana de ayuda que el usuario podrá evitar que se vuelva a abrir cuando considere que ha aprendido la nueva funcionalidad.
3. Cuando el usuario configura la agenda debe poder adaptarlo a su país seleccionado idioma, uso horario, símbolo monetario y feriados nacionales de forma tal que no le consuma más de 15 minutos.
4. Cuando el jefe del departamento administrativo configure el módulo de administración deberá poder seleccionar el país y asociado a este los parámetros impositivos y contables propios de ese país. En particular deberá tener la opción de seleccionar los diferentes comprobantes. Toda la tarea deberá llevarle aproximadamente una mañana.

Ejercicios

Ejercicio 17. Escribir un escenario general que tenga en cuenta el uso eficiente del sistema en tiempo de ejecución.

Ejercicio 18. Derivar un escenario concreto a partir del escenario general anterior que se relacione con el uso de un sistema de facturación que debe ser usado por cajeros y por clientes a través de una interfaz Web.

3.2.7. Cualidades del negocio

Además de los atributos de calidad, ciertos objetivos de calidad del negocio o *cualidades del negocio* suelen dar forma a los requerimientos no funcionales del sistema. Estos objetivos se centran en los costos, plan de trabajo, mercado, y consideraciones de marketing. Claramente estos conceptos se relacionan estrechamente con el negocio detrás del sistema (sección 1.6). Las cualidades del negocio ayudan a precisar el negocio detrás del sistema y, a través de este, influencian la ingeniería de requerimientos. Estas cualidades del negocio sufren de la misma ambigüedad que los atributos de calidad, por lo que es importante hacerlas específicas y precisas por medio de escenarios. Aquí se presentan cinco cualidades pero no se introducen escenarios.

Ventana de oportunidad. Refiere a la presión de la competencia o una ventana de oportunidad muy estrecha dentro de la cual el sistema debe comenzar a funcionar.

Costo y beneficio. El presupuesto destinado al sistema estará en función de los beneficios que este pueda arrojar y de las posibilidades financieras de la organización.

Tiempo de vida proyectado para el sistema. Esta cualidad afecta notablemente a los requerimientos e incluso a la arquitectura del sistema. Un sistema que se espera que se utilice por corto período de tiempo puede ser desarrollado de manera más laxa que otro que deberá operar por años.

Mercado objetivo. Son los destinatarios finales del sistema. Un sistema destinado al mercado masivo requiere características diferentes a uno destinado a una única organización.

Primera presentación al público. Puede darse el caso que el núcleo del sistema deba entrar en producción mucho antes que la mayoría de sus características avanzadas.

3.3. Validación de requerimientos no funcionales

Para la validación de requerimientos no funcionales proponemos utilizar la técnica de validación cruzada mencionada en la sección 2. En este caso, cada escenario de cada atributo de calidad se discute con cada interesado haciendo énfasis en las ventajas, desventajas y consecuencias que tendrá sobre el sistema su implementación. Si uno o más interesados presentan objeciones sobre el escenario, el equipo de ingenieros lo modificará y volverá a validarlo.

En algunos casos los escenarios podrán prototiparse más o menos fácilmente y a bajo costo por lo que el cliente podrá evaluar los prototipos y determinar si muestran lo que él esperaba o no.

4. Casos de uso

Si bien en las secciones anteriores dimos algunas indicaciones sobre la manera correcta de documentar requerimientos, en esta sección veremos una herramienta que ha sido muy difundida y publicitada para la documentación o descripción de requerimientos funcionales: los casos de uso.

4.1. Casos de uso y requerimientos funcionales

Comenzamos por la definición de caso de uso.

Definición 7 (Caso de Uso)**.** *Un caso de uso es un comportamiento del sistema que produce un resultado medible para al menos un actor [SW01]. Los casos de uso se describen como una sucesión de los pasos o acciones entre el actor y el sistema de software.*

Un ejemplo de caso de uso, perteneciente a un sistema de compras, puede verse en la Figura 18 (luego veremos con más detalle su estructura). Un caso de uso siempre es iniciado por un actor, aunque muy ocasionalmente puede ser iniciado desde dentro del sistema. De la definición se puede deducir que los casos de uso solo sirven para describir requerimientos funcionales. Entonces, ¿qué relación hay entre un requerimiento funcional y un caso de uso? La primera respuesta es que no hay una relación clara y precisa y diferentes autores entienden esa relación de distintas formas. Dependiendo del nivel de detalle del caso de uso, este puede ser lo mismo que un requerimiento funcional o puede ser algo más detallado que involucra más de un requerimiento funcional. El caso de uso mostrado en la Figura 18 es muy detallado (podríamos decir adecuadamente detallado). Ese caso de uso incluye, al menos, los siguientes requerimientos funcionales:

- Al realizar un pedido de mercadería, el cliente debe ingresar su nombre y dirección.
- Al realizar un pedido de mercadería, el cliente debe ingresar los códigos de productos que desea solicitar.

Figura 18 Ejemplo de un caso de uso

Caso de Uso: 01-AB — Solicitud de mercadería

Precondiciones: se ha ejecutado exitosamente el caso de uso "Ingreso al sistema".

Flujo de Eventos

Camino Básico

1. El caso de uso comienza cuando el cliente selecciona "Solicitud de mercadería".
2. El cliente ingresa su nombre y código postal.
3. El sistema muestra la ciudad y la provincia.
4. El cliente ingresa códigos de productos que desea comprar.
5. Para cada código de producto ingresado
   a) El sistema muestra la descripción y el precio del producto.
   b) El sistema suma el precio al total.
6. El cliente ingresa los datos de la tarjeta de crédito.
7. El cliente selecciona "Enviar".
8. El sistema verifica la información, almacena la solicitud como "pendiente" y reenvía la información del pago a la red de tarjetas de crédito.
9. Cuando el pago es confirmado, la solicitud se marca como "confirmada", un número de solicitud se le muestra al cliente, y el caso de uso finaliza.

Caminos Alternativos

Alternativa 1: Datos incorrectos

1. Esta alternativa comienza en el paso 8 del camino básico cuando el sistema detecta información incorrecta.
2. El sistema le solicita al cliente que corrija la información.
3. El camino básico continúa en el paso 8.

Alternativa 2: Cancelación

1. En cualquiera de los pasos del caso de uso el cliente puede seleccionar "Cancelar".
2. El sistema le solicita al cliente que confirme la cancelación.
3. El cliente selecciona "Aceptar" y el caso de uso finaliza.

Postcondiciones: si la solicitud no fue cancelada, se la almacena en el sistema y se la marca como "Confirmada".

Requerimientos especiales: si la confirmación del pago demora más de 1 minuto, la solicitud debe ser abortada.

- Al realizar un pedido de mercadería, los clientes pueden pagar con tarjeta de crédito o mediante transferencia.

En general, un requerimiento funcional no necesariamente describe una secuencia de pasos; es suficiente con que describa una interacción. Por ejemplo, las oraciones anteriores describen interacciones pero nada dice que el cliente deba primero ingresar su nombre y dirección, y luego los productos que desea solicitar. Esos requerimientos dicen todo lo que el sistema deberá hacer pero no en qué orden. Los casos de uso dan esa idea de orden. Por lo tanto deben ser usados en las situaciones donde esa idea de orden es importante. Si los requerimientos funcionales han sido deducidos a partir de prototipos como los que mencionamos en secciones anteriores, entonces los casos de uso tienden a ser innecesarios porque esa información está en los prototipos. Tampoco son muy necesarios en sistemas donde no hay un orden único (o unos pocos) para que los usuarios usen las funciones de aquellos. Por ejemplo, en un editor de textos los usuarios pueden primero escribir todo el texto y luego formatearlo, o pueden intercalar una cosa con la otra. Las combinaciones son millones. No se puede escribir un caso de uso para cada una. Además, no aportarían sustancialmente nada a una lista de requerimientos como la que sigue:

- El editor debe permitir que el usuario escriba un texto.
- El editor debe permitir que el usuario modifique el texto que ya ha escrito.
- El editor debe permitir que el usuario ponga una secuencia de texto en negrita.
- El editor debe permitir que quite la negrita a un texto.
- *El editor debe permitir mil operaciones más de formateado...*

Sin embargo, en el ejemplo del editor de texto, un caso de uso puede ser útil para detallar los pasos que el usuario deberá seguir para poner una secuencia de texto en negrita, a menos que los requerimientos funcionales hayan sido descriptos como en la página 4.

En consecuencia, y dependiendo del nivel de detalle y de la necesidad de establecer un orden de ejecución entre ciertos requerimientos, un caso de uso puede ser el contenido del campo **Descipción** de la lista consolidada de requerimientos mencionada en la página 13 de la Sección 1.4.

Definamos ahora el concepto de actor mencionado en la definición de caso de uso.

Definición 8 (Actor)**.** *En el contexto de los casos de uso, un actor es un interesado u otro sistema o dispositivo físico (por ejemplo, un radar, un sensor, una placa de red, etc.; en general son dispositivos no controlados por una persona) con los que tiene que interactuar el sistema que se está desarrollando.*

Al utilizar casos de uso, el primer paso es listar los actores que se relacionan con el sistema que se va a desarrollar. Los actores están fuera de la frontera del sistema; interactúan con él pero no son parte del él. En la descripción de un caso de uso se debe indicar con precisión cuál es el actor que lo inicia, y cuáles son los que participan; es decir se deben usar los actores listados. Por ejemplo, si se ha identificado el actor “Gerente” no es lo mismo usar en un caso de uso la palabra “Jefe” como sinónimo de “Gerente”. Esos cambios dan lugar a errores, malos entendidos, ambigüedades, contradicciones, etc. En otras palabras los nombres de los actores son similares a los nombres de variables en un programa: no se pueden cambiar y no se puede escribir uno que sea parecido.

Para identificar los primeros casos de uso suele ser útil responder las siguientes preguntas:

- ¿Cuáles son las funciones que el usuario quiere del sistema?
- ¿El sistema almacena información? ¿Cuáles son los actores que crearán, consultarán, actualizarán o borrarán esa información?
- ¿Es necesario que el sistema notifique a un actor sobre algún cambio en su estado?
- ¿Hay eventos externos que el sistema deba conocer? ¿Cuáles son los actores que le informarán al sistema sobre esos eventos?

También se puede comenzar por considerar un caso de uso para cada una de las siguientes situaciones: inicio, terminación, diagnóstico, instalación y cambio de un proceso de negocio.

Una vez identificados los primeros casos de uso se procederá a detallarlos como se indica más abajo, lo que seguramente dará lugar a nuevos casos de uso. Este proceso podría repetirse varias veces pero seguramente la aparición de nuevos casos de uso decrecerá con el tiempo como indica la Figura 3. Más aun, la aparición de nuevos casos de uso puede implicar tener que revisar y reformular algunos de los casos de uso ya detallados. Sin embargo, la reformulación de casos de uso debería, también, decrecer con el tiempo. Tanto la aparición de nuevos casos de uso como la necesidad de reformular los ya descriptos, son indicadores de la proximidad del fin de la etapa de ingeniería de requerimientos.

4.2. Requerimientos funcionales temporales

En algunos sistemas ciertas actividades deben realizarse en ciertos momentos. Por ejemplo, la liquidación de sueldos o la impresión de un reporte de ventas diarias. Estas actividades corresponden a uno o varios *requerimientos funcionales temporales*, los cuales normalmente pueden expresarse como casos de uso. Para describir casos de uso que involucran requisitos temporales hay dos estrategias:

- Considerar al tiempo como un actor; por ejemplo un temporizador que emite un evento cuando llega a cero.
- Considerar que el sistema cuenta el tiempo y en tal caso inicia un caso de uso cuando se cumple el tiempo esperado.

4.3. La forma de un caso de uso

Lo primero y más importante que queremos resaltar es que, en nuestra opinión, solo los casos de uso de pequeños proyectos pueden describirse gráficamente (opinión que se opone a las sugerencias de la mayoría de los autores). En proyectos medianos a grandes, los casos de uso deben describirse textualmente. No hay otra forma de lograr dominar la complejidad de tales proyectos ni de visualizar el número de casos de uso que habrá. En otras palabras, describir casos de uso es esencialmente una tarea de escritura, no de dibujo [Lar04].

Un caso de uso es, entonces, un (pequeño) texto en lenguaje natural que se se compone de las siguientes secciones (ver la Figura 18):

Flujo de los eventos. Es una lista de pasos que deben seguirse para arribar al resultado esperado, según la perspectiva del usuario. En otras palabras estos pasos deben seguir todos los lineamientos que hemos visto: solo debe haber vocabulario del entorno, solo debe hablarse del dominio de aplicación, etc. Los pasos deben enumerarse como en la descripción de

un algoritmo (los números se utilizarán para referirse desde otro paso). Se pueden usar oraciones condicionales (Si-Entonces) y los pasos se pueden organizar como para ejecutar un bucle.

Se recomienda subdividir el flujo de eventos en las siguientes secciones:

Camino básico. El camino básico describe el flujo de eventos suponiendo que no hay errores de ningún tipo. Se asume que los actores se comportan de manera correcta. Este camino describe la forma más común de hacer lo que el caso de uso intenta capturar. Debe haber un único camino básico por caso de uso. El camino básico no tiene oraciones condicionales.

Caminos alternativos. Un camino alternativo es aquel que describe una secuencia de eventos diferentes a la que describe el camino básico. En esta sección, por consiguiente, se pueden documentar comportamientos anómalos, errores cometidos por el actor, opciones seleccionadas por el actor que no fueron descriptas en el camino básico, etc. Los caminos alternativos se documentan de la misma forma que el camino básico, aunque pueden contener oraciones condicionales referidas al camino básico u otros caminos alternativos. Cada camino alternativo debe ser identificado con un nombre apropiado.

Un método para encontrar los caminos alternativos es recorrer el camino básico y preguntarse lo siguiente:

- ¿Hay alguna otra acción que puede ocurrir en este punto?
- ¿Hay algo que pueda salir mal en este punto?
- ¿Hay algún comportamiento que se pueda dar en cualquier momento?

También es posible explorar lo siguiente:

- Un actor sale del sistema.
- Un actor cancela una operación particular.
- Un actor solicita ayuda.
- Un actor provee datos erróneos.
- Un actor provee datos incompletos.
- Un actor elige una forma alternativa de llevar adelante el caso de uso.
- El sistema se cuelga.
- El sistema no está disponible.

Postcondiciones. Propiedades que deben ser verdaderas una vez que el caso de uso ha finalizado. Es decir, el sistema debe dejar las cosas en un estado tal que las postcondiciones del caso de uso se verifiquen (caso contrario el sistema no cumplirá con los requerimientos y por lo tanto será incorrecto).

Algunos ejemplos de postcondiciones son:

- El saldo de la caja de ahorros es igual al saldo anterior menos el monto de la extracción; o el saldo queda igual si la extracción fue cancelada.
- Si se provee un usuario y contraseña correctos, se registra al usuario como “logueado”; caso contrario el sistema queda igual que antes.

Observar que puede haber postcondiciones que refieran al camino básico y otras que refieran a algún camino alternativo. Es decir, si el caso de uso discurre por el camino

básico el sistema satisface una cierta postcondición, pero si discurre por uno alternativo, se satisface otra. En consecuencia puede ser conveniente relacionar las postcondiciones con el camino básico o los alternativos. Por ejemplo:

- Si "Camino básico" entonces: *postcondición 1*.
- Si "Alternativa 1" entonces: *postcondición 2*.
- .....
- Si "Alternativa n" entonces: *postcondición* $n+1$.

Precondiciones. Propiedades que deben ser verdaderas antes de que el caso de uso comience a ejecutarse. Si alguna de las precondiciones no se cumple entonces el caso de uso no debe ser ejecutado. Si no hay ninguna precondición entonces esta sección puede omitirse.

Una forma que muchas veces resulta útil para establecer las precondiciones de un caso de uso es indicando los casos de uso que deben ejecutarse antes del que se está describiendo. Como un caso de uso incluye también la descripción de las postcondiciones (ver siguiente ítem), entonces indicar la ejecución previa de otro caso de uso implica que las postcondiciones establecidas por este satisfacen la precondiciones requeridas por el que se está describiendo. Debe tenerse en cuenta que un caso de uso puede finalizar satisfaciendo diferentes postcondiciones, entonces puede surgir la necesidad de indicar cuál de ellas es la que se espera que valga justo antes de comenzar la ejecución del caso de uso que se está describiendo (¿esperamos que el caso de uso que invocamos termine en el camino básico o en uno de los alternativos?).

Algunos ejemplos de precondiciones son:

- El cliente debe haber efectuado una "Solicitud de mercadería".
- Debe haber alguna solicitud de mercadería marcada como "confirmada" a nombre del cliente.

Requerimientos especiales. En esta sección se deben describir requerimientos no funcionales que sean específicos al caso de uso. Si no hay tales requerimientos entonces esta sección puede omitirse.

Los casos de uso pueden incluir otras secciones y pueden derivarse unos de otros o incluirse unos en otros, pero no veremos esas técnicas en este curso, aunque pueden ser muy útiles en proyectos medianos o grandes [SW01, Lar04]. También existen lenguajes más formales para describir casos de uso como SilabReq [SAV+11]. A quienes se interesen aun más en estos temas les sugerimos profundizar la bibliografía que se indica en cada caso.

Ejercicios

Ejercicio 19. Escriba cinco casos de uso para el problema de la farmacia del Ejercicio 1.

Ejercicio 20. Escriba cinco casos de uso para el problema del restaurante del Ejercicio 2.

4.4. Casos de uso y diseño orientado a objetos

Si bien la metodología para obtener los requerimientos de un sistema debería ser independiente de de la metodología para diseñar el sistema, conviene que haya cierta vinculación entre ambas. Los casos de uso como técnica para documentar requerimientos funcionales han sido ampliamente utilizados junto a metodologías de diseño orientado a objetos (DOO). Según esta visión los casos de uso son una forma muy apropiada para, a partir de ellos, obtener las clases del diseño del sistema que los implementará.

Una vez más disentimos con esta opinión predominante. Nosotros creemos que los casos de uso son una buena técnica para describir requerimientos funcionales pero no son la fuente más apropiada para determinar las clases de un DOO. La razón fundamental por la cual sostenemos esta idea es que los casos de uso representan una visión algorítmica del sistema en tanto que el DOO es ortogonal a esta visión. Más aun, el diseño de un sistema es la herramienta fundamental para reducir sus costos de mantenimiento, permitiendo la incorporación de nuevos requerimientos a bajo costo y sin reducir la integridad conceptual del sistema. Los casos de uso son descripciones de los requerimientos presentes, no dan ninguna pista evidente sobre los requerimientos futuros ni sobre los cambios más probables a los que será sometido el sistema. En otras palabras, no hay una relación simple, clara y directa entre casos de uso y clases de un DOO. Entre los caso de uso y las clases de un DOO hay una distancia muy grande la cual es muy difícil de recorrer si no se dispone de la herramienta técnica apropiada.

En nuestra opinión las clases de un DOO se deben obtener a partir de los ítem de alta probabilidad de cambio presentes en los requerimientos (es decir, si se usan casos de uso para documentar los requerimientos, en los casos de uso) según indica la metodología de diseño basado en ocultación de información propuesta por David L. Parnas a principios de los años setenta del siglo pasado [Cri10].

Para finalizar con las discrepancias con el modelo de desarrollo dominante, diremos que el análisis orientado a objetos adolece, en nuestra opinión, de un error de fundamento: los dominios de aplicación no están poblados de objetos en el sentido informático. Como los requerimientos yacen en el dominio de aplicación, intentar describirlos mediante objetos llevará inevitablemente al fracaso [Jac95, páginas 135–138]. Esta es la razón por la cual no hemos tratado este tema en este curso.

5. Estructura de un documento de requerimientos según el estándar IEEE 830-1998

En esta sección se presenta brevemente la estructura sugerida por el estándar IEEE 830-1998 [IEE98] para el documento de requerimientos. Para más detalles sugerimos leer detenidamente el mencionado estándar.

Introducción. Esta sección a su vez se subdivide en las siguientes.

Propósito. Describe el propósito del documento de requerimientos y el público al cual se dirige.

Alcance. Debe identificar los productos que se producirán por su nombre, explicar brevemente qué harán esos productos y describir el negocio detrás del sistema.

Definiciones, siglas y abreviaturas.

Referencias.

Contenido general del documento.

Descripción general. Esta sección a su vez se subdivide en las siguientes.

El producto en perspectiva. Se describe el producto en relación a los otros sistemas con los cuales interactúa. Si es un componente de un sistema mayor entonces debe aclararse esta situación y deben describirse las interfaces que los unen y cuáles de los requerimientos del sistema mayor viene a cumplir este componente.

Además se debe describir cómo el software operar en relación a las siguientes cuestiones:

1. Interfaces con otros sistemas (usualmente desarrollados dentro de la misma organización)
2. Interfaces de usuario
3. Interfaces de hardware
4. Interfaces de software (sistemas de bases de datos, sistemas operativos, paquetes de cálculo matemático, etc.)
5. Interfaces de comunicación
6. Memoria
7. Operaciones (política de respaldo, modos de operación, etc.)

Funciones del producto. Resumen de las principales funciones que el producto cumplirá.

Características de los usuarios del producto.

Restricciones. En esta sección se deben listar y explicar cualquier otra cuestión que limite las opciones del desarrollador para diseñar e implementar el sistema. Estas incluyen:

1. Políticas de regulación
2. Limitaciones de hardware
3. Interfaces con otras aplicaciones
4. Si el sistema debe operar en paralelo o no
5. Funciones de auditoría
6. Funciones de control
7. Requerimientos sobre el lenguaje de programación
8. Protocolos de comunicación
9. Criticidad de la aplicación
10. Consideraciones de seguridad y salvaguarda

Supuestos y dependencias. Se deben listar los supuestos y las dependencias que de no cumplirse implicarán cambios en uno o más de los requerimientos que se describen a continuación. Por ejemplo: se asume que el sistema interactuará con tal o cual DBMS. Si se confirma que tal DBMS no está disponible entonces uno o más requerimientos deben ser modificados.

Requerimientos específicos. Aquí se listan los requerimientos específicos. En términos generales corresponde a lo que nosotros denominamos "lista estructurada de requerimientos" en la Sección 1.4. En aquella sección dijimos que esta lista se dividía en secciones, subsecciones, etc. con el fin de organizarla de manera tal que permita encontrar un requerimiento específico más fácilmente. El IEEE 830-1998 sugiere que esta organización esté basada en alguno de los siguientes criterios:

Modos de operación del sistema. Algunos sistemas se comportan de manera muy diferente de acuerdo a los modos de operación en que pueden trabajar.

Usuarios. Algunos sistemas proveen diferentes conjuntos de funciones para diferentes tipos de usuarios.

Servicios. En algunos sistemas es natural identificar servicios claramente separados.

Estímulos. Algunos sistemas pueden organizarse muy bien al describir todas las funciones afectadas por un estímulo dado que recibe el sistema.

Respuestas. Algunos sistemas pueden organizarse muy bien al describir todas las funciones que contribuyen a la generación de un cierta respuesta.

Jerarquía funcional. Cuando ninguno de los esquemas anteriores resulta conveniente, la funcionalidad puede organizarse en una jerarquía de funciones basada en entradas, salidas o estructuras de datos internas comunes.

Ejercicios

Ejercicio 21. Describa en forma general la estructura que tendría la lista de requerimientos si el criterio de división fuesen los tipos de usuarios.

Ejercicio 22. Describa en forma general la estructura que tendría la lista de requerimientos si el criterio de división fuesen los diferentes estímulos.

Referencias

[BP88] B. W. Boehm and P. N. Papaccio. Understanding and controlling software costs. *IEEE Trans. Softw. Eng.*, 14(10):1462–1477, 1988.

[BPKR09] Brian Berenbach, Daniel Paulish, Juergen Kazmeier, and Arnold Rudorfer. *Software & Systems Requirements Engineering: In Practice*. McGraw-Hill, Inc., New York, NY, USA, 2009.

[Cri10] Maximiliano Cristiá. Diseño de software. http://www.fceia.unr.edu.ar/ingsoft/diseno-a.pdf, 2010.

[GGJZ00] Carl A. Gunter, Elsa L. Gunter, Michael Jackson, and Pamela Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37–43, May 2000.

[IEE98] IEEE. IEEE recommended practice for software requirements specifications. Technical report, 1998.

[Jac95] Michael Jackson. *Software requirements & specifications: a lexicon of practice, principles and prejudices*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.

[Lar04] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[Pfl01] Shari Lawrence Pfleeger. *Software Engineering: Theory and Practice*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[Rob04] Suzanne Robertson. Requirements and the business case. *IEEE Softw.*, 21:93–95, September 2004.

[SAV+11] Dusan Savić, Ilija Antović, Siniša Vlajić, Vojislav Stanojević, and Miloš Milić. Language for use case specification. In *Proceedings of the 34th IEEE Annual Software Engineering Workshop*, Limerik, Irland, 2011. IEEE Computer Society. — to be published.

[Sch04] Michael Schrage. Never go to a client meeting without a prototype. *IEEE Software*, 21(2):42–45, 2004.

[Som10] Ian Sommerville. *Software Engineering*. Addison-Wesley, Harlow, England, 9th edition, 2010.

[SW01] Geri Schneider and Jason P. Winters. *Applying use cases (2nd ed.): a practical guide*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[ZJ97] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1), January 1997.