UNIVERSIDAD DE CHILE
FACULTAD DE DERECHO
ESCUELA DE POSTGRADO

Análisis de causales que generan controversia en los contratos de desarrollo y/o implementación de software

Proyecto de Actividad Formativa Equivalente a Tesis para optar al grado de Magíster en Derecho y Nuevas Tecnologías

DANIEL ANDRÉS VALENZUELA SUÁREZ

Profesor guía: Claudio Magliona Markovicth

Santiago, 28 de Agosto de 2017

RESUMEN

El presente trabajo de investigación tiene por objetivo abordar la problemática que existe en los contratos de prestaciones y/o desarrollo de software, en relación directa con las controversias que se generan por falta de definiciones técnicas, de negocio y/o de implementación.

Por consiguiente, deseamos dar a conocer cómo los contratos de software han cambiado la esquematización de los contratos convencionales solo por el hecho de incorporar en su estructuración, productos intangibles y no especificados completamente.

Teniendo en consideración que la Ley 19.496 detalla claramente como son respaldados los consumidores pese a que sus contratos sean de manera virtual y no a lo que acostumbramos a hacer personalmente. ¿a qué tipo de contratos corresponden?, ¿existen protecciones legales para dichos contratos?, y si ¿la ley ampara a las partes en cuanto a los vacíos legales?.

Dedicado a mi familia, amigos y personas
que me han apoyado en todo este proceso,
en especial a la persona que me motivó
a realizar este programa, esto es para ti abuelita.

AGRADECIMIENTO

Con estas palabras me gustaría agradecer en primer lugar a mis padres Zenón Valenzuela y María Eugenia Suárez, quienes desde pequeño me han apoyado, guiado y enseñado los valores y medios para desarrollarme en este mundo. Nunca podré agradecerles lo suficiente. También agradezco el apoyo y cariño que me brindan mis hermanos Cristian Valenzuela y Yesenia Valenzuela.

También me gustaría agradecer a las familia, quienes durante todo mi proceso educacional, siempre han estado apoyándome, entregándome ánimo y confianza. En particular me gustaría dedicarle este trabajo a mi abuela Zelmira Vásquez, quien estoy seguro estaría orgullosa de haberme visto culminar mi formación educacional. De la misma forma, agradecer a mis amigos de vida quienes siempre me han dado su ayuda, consejos y divertidos momentos para sobrellevar las dificultades ocasionales en este proceso.

Agradezco de corazón a mi profesor guía Claudio Magliona, quien me dio la oportunidad de trabajar con él, y durante todo este proceso siempre estuvo dispuesto a apoyarme y guiarme con una paciencia infinita. Su constante ayuda, consejos y lecciones no sólo han contribuido al desarrollo de este trabajo, sino también a formarme como la persona y profesional que deseo llegar a ser.

Doy gracias también a los profesores Daniel Álvarez, Renato Jijena y a diversos académicos de la Facultad de Derecho de esta Universidad, por enseñarme, aconsejarme y orientarme durante este camino.

Finalmente, pero no menos importante, me gustaría dedicar mis agradecimientos a mis compañeros de postgrado, con quienes he formado lazos de amistad en este proceso y deseo que prosperen durante mucho tiempo más.

TABLA DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

CAPÍTULO 1 INTRODUCCIÓN

1.1 ANTECEDENTES Y MOTIVACIÓN

Hace un par de años atrás era impensable concebir que la tecnología llegaría a tal punto, en que podríamos dejar atrás funciones tan básicas como ir al supermercado, a merced de una computadora o una transacción electrónica.

Es debido a los avances en esta área que, al día de hoy, todo se desea transmitir a través de plataformas de internet, por lo cual, comienza a quedar atrás todo lo relacionado con transacciones de persona a persona. Es por esto, que el sistema jurídico se ha visto expuesto a avanzar en este sentido, ya que la legislación no contemplaba muchos casos que, hoy en día, están siendo vulnerados diariamente.

Y es, en este sentido, que hoy queremos exponer unos de los temas que tiene claras falencias en cuanto a regulación, los llamados “contratos de software”. Estos han tenido una gran controversia, debido a que son contratos, por lo general, de productos intangibles. Si bien el artículo 1438 del Código Civil, define contrato como "un acto por el cual una parte se obliga para con otra a dar, hacer o no hacer alguna cosa", entonces en base a esta definición, ¿son los contratos de software de la misma índole que un contrato convencional?, ¿de qué tipo de contrato se trata? ¿deben ser estos contratos elaborados por quien conoce de leyes, quien conoce de software o ambos especialistas juntos?

Abordaremos estos puntos en los siguientes capítulos de esta investigación, para reflejar cómo se resuelven dichas controversias en el caso de estos nuevos contratos.

1.2 DESCRIPCIÓN DEL PROBLEMA

Según el instituto de estudios en ingeniería de software **Standish Group** (Standish Group, 2016), las problemáticas que se presentan en los contratos de desarrollo de software, radican principalmente, en una falta de conocimiento de las partes sobre elementos técnicos propios del desarrollo de software, como lo son:

- La toma de requerimientos
- Diseños de la solución
- Prototipos de aplicación
- Experiencia de usuario
- Pruebas de aceptación
- Pruebas de rendimiento y seguridad
- Continuidad de la operación

Estos aspectos, que pueden visualizarse de manera simple, conllevan a un entendimiento por ambas partes del objetivo principal del contrato, el cual es **resolver el problema mediante una solución tecnológica (informática)**. Si bien, estos elementos son abordados, las partes lo afrontan una vez empezado el proyecto y elaborado el contrato, por lo cual, no quedan resguardados ninguno de ellos.

Lo que se realiza en la investigación de este proyecto, es contribuir con la generación de un documento que se pueda anexar a un contrato de software, el cual contenga las terminologías técnicas y etapas necesarias, con el fin de garantizar definiciones claras, puntos de equilibrio entre ambas partes y las etapas detalladas del desarrollo de software.

1.3 SOLUCIÓN PROPUESTA

Para afrontar el problema presentado, se deben abordar las temáticas descritas anteriormente, las cuales se visualizan a continuación.

1.3.1 Características de la solución

Para solucionar el problema de investigación, debemos revisar elementos fundamentales. El primer elemento a revisar, se refiere a la sección de Investigación, que suele identificarse como "I", la cual implica que se valide la siguiente hipótesis: "*Si los contratos de desarrollo y/o implementación de software fueran elaborados por especialistas legales y técnicos se disminuiría el riesgo de fracaso en la entrega final del producto y cumplimiento efectivo del contrato celebrado*". Para ello, se deben obtener resultados lo más precisos posibles (con variables del modelo, tratamientos y recogidas de datos con instrumentos estandarizados) (Gallego, Villagrá, Satorre, Compañ, Molina & Llorens Largo, 2014).

La investigación considera las siguientes variables:

1. **Independientes (Causa):** Se definen los tratamientos que se realizarán para disminuir las controversias:
    a. Ingeniería de requerimientos
    b. Pruebas de calidad

2. **Dependientes (Efecto):** Cada variable generará un efecto que necesita ser medido de forma cualitativa y cuantitativa. Se consideran tres factores como parte del efecto:
    a. Ingeniero de software
    b. Cliente
    c. Usuario final

Para lograr solucionar la sección de investigación, en primer lugar es necesario elaborar un **Anexo Técnico**. Debemos analizar un segundo elemento que es la sección de desarrollo "D" de nuestra investigación. En dicha sección, se pretende diseñar un documento técnico, que permita la realización de la toma de requerimientos, con las siguientes características:

1. Dominio del problema
2. Características de la solución

3. Actores de la problemática
4. Tratamiento de toma de requerimientos
5. Análisis de las funcionalidades que se desean implementar

1.3.2 Propósito de la solución

Este proyecto de investigación pretende ser una contribución al área de contratos de servicios y/o productos ligados a la ingeniería de software e ingeniería de requerimientos, dando a conocer que existe un análisis sobre las causales que generan controversias en dichos contratos.

Finalmente, se propone elaborar un documento técnico que permita disminuir las controversias, detallando de manera clara para ambas partes, los detalles técnicos del problema.

1.4 OBJETIVOS Y ALCANCE DEL PROYECTO

El presente trabajo se encuentra desarrollado dentro de las líneas de investigación de la ingeniería de software, ingeniería de requerimientos, contratación de productos y servicios.

1.4.1 Objetivo general

Dar a conocer cómo los contratos de software han cambiado la esquematización de los contratos convencionales solo por el hecho de incorporar en su estructuración productos intangibles y no especificados completamente.

1.4.2 Objetivos específicos

Para poder conseguir el objetivo general de la investigación, se deben realizar los siguientes objetivos específicos:

1. Determinar a qué tipo de contratos corresponden.
2. Determinar las protecciones legales.
3. Determinar si la ley ampara a las partes en cuanto a los vacíos legales.

1.4.3 Alcances

Tomando en consideración que este proyecto es de I+D, los alcances o limitaciones son los siguientes:

1. El contexto de Investigación (I) que es la arista principal de la tesis, consistirá de un estudio que analiza las causales que generan controversias en los contratos ligados a la ingeniería de software. Se debe considerar que el alcance de esta investigación será la ingeniería de requerimientos.

2. En el contexto del Desarrollo (D) se elaborará un Documento Técnico que será anexado a los contratos de servicios y/o productos de software. El tiempo de desarrollo no excederá el 40% del tiempo del trabajo de tesis, dado que se enfocará en desarrollos de software orientados a la Web 2.0 y 3.0.

1.5 ORGANIZACIÓN DEL DOCUMENTO

El presente documento se estructura de la siguiente forma. En el capítulo 2 'Marco teórico', se estipulan los conceptos teóricos que se deben definir para contar con una base consensuada respecto de los distintos conceptos que se tratan en este documento. En el mismo capítulo se aborda el estado del arte que define los trabajos e investigaciones científicas realizadas sobre la ingeniería de software con énfasis en la ingeniería de requerimientos y, por otra parte, los estudios en materia legal sobre los contratos en servicios y productos ligados a la ingeniería de software.

Luego, en el capítulo 3 'Metodología', se presenta la formulación y características de todo el proceso de propuesta de solución planteada. Esto contempla aspectos como el diseño del estudio, lo que se plantea descubrir, las tareas necesarias para realizar la investigación, las características de los documentos técnicos y la forma en que se deben llevar a cabo la recopilación de los datos.

Finalmente, en el capítulo 4 'Conclusiones', se entregan todas las conclusiones obtenidas respecto al desarrollo de la investigación, trabajos futuros, implicaciones teóricas y prácticas, y reflexiones finales del proceso realizado.

CAPÍTULO 2 MARCO TEÓRICO

2.1 MARCO METODOLÓGICO

Toda investigación se fundamenta en un marco metodológico, el cual define el uso de métodos, técnicas, instrumentos, estrategias y procedimientos a utilizar en el estudio que se desarrollara. Al respecto Balestrini (Balestrini, 2006) define el marco metodológico como "la instancia referida a los métodos, las diversas reglas, registros, técnicas y protocolos con los cuales una teoría y su método calculan las magnitudes de lo real". Según Finol y Camacho (Finol & Camacho, 2008) el marco metodológico está referido a "cómo se realizará la investigación, muestra el tipo y diseño de la investigación, población, muestras, técnicas e instrumentos para la recolección de datos, validez y confiabilidad, y las técnicas para el análisis de datos".

2.1.1 Tipos y niveles de investigación

La investigación se realizó a nivel descriptivo, dado que se identificaron los elementos y características del problema planteado, con el fin de estipular las bases para futuras investigaciones y/o mejoras legislativas en la contratación de desarrollo y/o implementación de software. Adicionalmente de lo anterior, se ha realizado una investigación exploratoria sobre la temática abordada en la investigación, realizando una hipótesis del proyecto investigado. La investigación que se realizó es de tipo documental y de variables dependientes e independientes (Elgueta, Palma, 2010), las cuales permiten analizar y comparar los datos a una realidad nacional, con el fin obtener resultados que puedan ser aplicables al sistema jurídico.

2.1.2 Métodos de investigación

Los métodos que se emplearon para realizar la investigación, comienzan con el método científico especificado en el libro de Roberto Hernández (Hernández, Fernández, Baptista & Casas, 1998/9), a su vez, el método analítico (Álvarez, 2003) para determinar las posibles falencias en nuestra legislación respecto de la materia a tratar, las posibles causas y problemas que presentan, y de qué manera es posible abordarlas. Adicionalmente, se incorporaron los métodos histórico, sistemático y estadístico (Elgueta, Palma, 2010) para profundizar en las distintas etapas de la correspondiente investigación, con el fin de lograr un entendimiento más completo de algunos de los datos o términos necesarios para su completo estudio.

2.1.3 Técnicas e instrumentos de recolección de los datos

Para realizar esta investigación, se utilizaron documentos, leyes de la República de Chile, sentencias judiciales en directa relación a la temática investigada, estudios realizados por instituciones especializadas, artículos comparativos y de prensa especializada que pudiera ser de utilidad, es decir, fuentes especializadas de información.

2.1.4 Tratamiento de los datos

Los datos obtenidos mediante las técnicas mencionadas anteriormente y su correspondiente análisis, fueron incorporados de forma sistemática dentro del informe final, catalogadas por áreas definidas en el índice de contenidos. La representación de los datos (información), es presentada de forma escrita y gráfica según corresponda, para un mayor entendimiento del análisis obtenido. Además de la información ya mencionada, se visualizan las falencias y/o vulnerabilidades que presentan los contratos de desarrollo y/o implementación de software respecto de la legislación chilena.

2.2 MARCO CONCEPTUAL

En la actualidad, el software tiene un papel dual. Es un producto y al mismo tiempo es el vehículo para entregar un producto. En su forma de producto, brinda el potencial de cómputo incorporado en el hardware, con más amplitud, en una red de computadoras a las que se accede por medio de un hardware local. Ya sea que resida en un teléfono móvil u opere en el interior de una computadora central, el software es un transformador de información que produce, administra, adquiere, modifica, despliega o transmite información que puede ser tan simple como un solo bit o tan compleja como una presentación con multimedios generada a partir de datos obtenidos de decenas de fuentes independientes. Como vehículo utilizado para distribuir el producto, el software actúa como la base para el control de la computadora (sistemas operativos), para la comunicación de información (redes) y para la creación y control de otros programas (herramientas y ambientes de software).

El software distribuye el producto más importante de nuestro tiempo: información. Transforma los datos personales (por ejemplo, las transacciones financieras de un individuo) de modo que puedan ser más útiles en un contexto local, administra la información de negocios para mejorar la competitividad, provee una vía para las redes mundiales de información (la internet) y brinda los medios para obtener información en todas sus formas.

En el último medio siglo, el papel del software de cómputo ha sufrido un cambio significativo. Las notables mejoras en el funcionamiento del hardware, los profundos cambios en las arquitecturas de computadora, el gran incremento en la memoria y capacidad de almacenamiento, y una amplia variedad de opciones de entradas y salidas exóticas han propiciado la existencia de sistemas basados en computadora más sofisticados y complejos. Cuando un sistema tiene éxito, la sofisticación y complejidad producen resultados deslumbrantes, pero también plantean problemas enormes para aquellos que deben construir sistemas complejos.

En la actualidad, la enorme industria del software se ha convertido en un factor dominante en las economías del mundo industrializado. Equipos de especialistas de software, cada uno centrado en una parte de la tecnología que se requiere para llegar a una aplicación compleja, han reemplazado al programador solitario de los primeros tiempos. A pesar de

ello, las preguntas que se hacía aquel programador son las mismas que surgen cuando se construyen sistemas modernos basados en computadora:

- ¿Por qué se requiere tanto tiempo para terminar el software?
- ¿Por qué son tan altos los costos de desarrollo?
- ¿Por qué no podemos detectar todos los errores antes de entregar el software a nuestros clientes?
- ¿Por qué dedicamos tanto tiempo y esfuerzo a mantener los programas existentes?
- ¿Por qué seguimos con dificultades para medir el avance mientras se desarrolla y mantiene el software?

Éstas y muchas otras preguntas, denotan la preocupación sobre el software y la manera en que se desarrolla, preocupación que ha llevado a la adopción de la práctica de la ingeniería del software.

2.2.1 Definición de software

En la actualidad, la mayoría de profesionales y muchos usuarios tienen la fuerte sensación de que entienden el software. Pero, ¿es así?

La descripción que daría un libro de texto sobre software sería más o menos así: el software es: 1) instrucciones (programas de cómputo) que cuando se ejecutan proporcionan las características, función y desempeño buscados; 2) estructuras de datos que permiten que los programas manipulen en forma adecuada la información, y 3) información descriptiva tanto en papel como en formas virtuales que describen la operación y uso de los programas.

No hay duda de que podrían darse definiciones más completas. Pero es probable que una definición más formal no mejore de manera apreciable nuestra comprensión. Para asimilar lo anterior, es importante examinar las características del software que lo hacen diferente de otros objetos que construyen los seres humanos. El software es elemento de un sistema lógico y no de uno físico. Por tanto, tiene características que difieren considerablemente de las del hardware:

1 - El software se desarrolla o modifica con intelecto; no se manufactura en el sentido clásico.
Aunque hay algunas similitudes entre el desarrollo de software y la fabricación de hardware, las dos actividades son diferentes en lo fundamental. En ambas, la alta calidad se logra a través de un buen diseño, pero la fase de manufactura del hardware introduce problemas de calidad que no existen (o que se corrigen con facilidad) en el software. Ambas actividades dependen de personas, pero la relación entre los individuos dedicados y el trabajo logrado es diferente por completo. Las dos actividades requieren la construcción de un "producto", pero los enfoques son distintos. Los costos del software se concentran en la ingeniería. Esto significa que los proyectos de software no pueden administrarse como si fueran proyectos de manufactura.

2 - El software no se desgasta.
La figura 2-01 ilustra la tasa de falla del hardware como función del tiempo. La relación, que es frecuente llamar "curva de tina", indica que el hardware presenta una tasa de fallas relativamente elevada en una etapa temprana de su vida (fallas que con frecuencia son atribuibles a defectos de diseño o manufactura); los defectos se corrigen y la tasa de fallas se abate a un nivel estable (muy bajo, por fortuna) durante cierto tiempo.

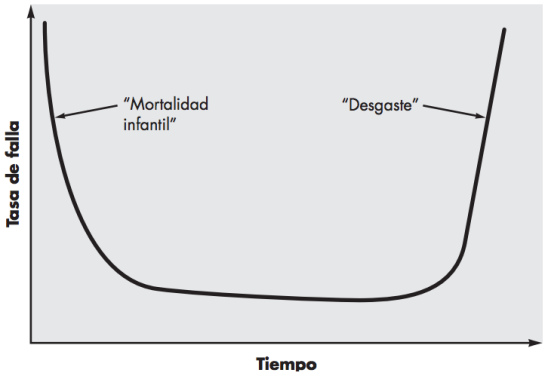


Figura Nº2-01 - Curva de fallas de hardware

No obstante, conforme pasa el tiempo, la tasa de fallas aumenta de nuevo a medida que los componentes del hardware resienten los efectos acumulativos de suciedad, vibración, abuso, temperaturas extremas y muchos otros inconvenientes ambientales. En pocas palabras, el hardware comienza a desgastarse. El software no es susceptible a los problemas ambientales que hacen que el hardware se desgaste.

Por tanto, en teoría, la curva de la tasa de fallas adopta la forma de la "curva idealizada" que se aprecia en la figura 2-02. Los defectos ocultos ocasionarán tasas elevadas de fallas al comienzo de la vida de un programa. Sin embargo, éstas se corrigen y la curva se aplana, como se indica. La curva idealizada es una gran simplificación de los modelos reales de las fallas del software. Aun así, la implicación está clara: el software no se desgasta, ¡pero sí se deteriora! Esta contradicción aparente se entiende mejor si se considera la curva real en la figura 2-02.

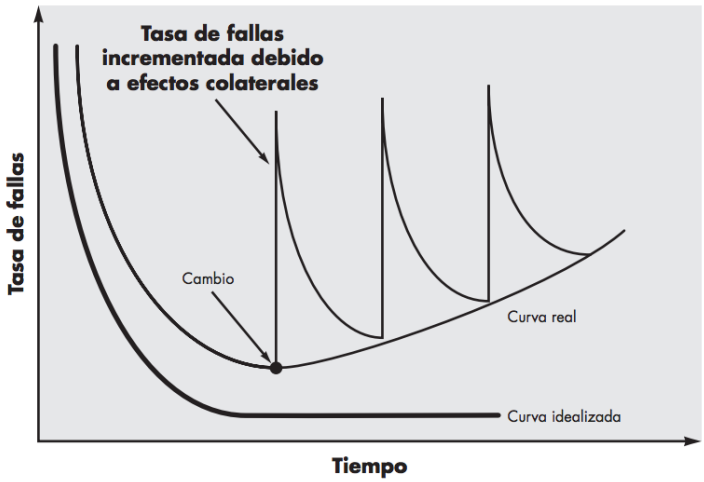


Figura Nº2-02 - Curvas de fallas de software

Durante su vida, el software sufrirá cambios. Es probable que cuando éstos se realicen, se introduzcan errores que ocasionen que la curva de tasa de fallas tenga aumentos súbitos, como se ilustra en la "curva real" (ver la figura 2-02). Antes de que la curva vuelva a su tasa de fallas original de estado estable, surge la solicitud de otro cambio que hace que la curva se dispare otra vez. Poco a poco, el nivel mínimo de la tasa de fallas comienza a aumentar: el software se está deteriorando como consecuencia del cambio. Otro aspecto del desgaste ilustra la diferencia entre el hardware y el software. Cuando un componente del hardware se desgasta es sustituido por una refacción. En cambio, no hay refacciones para el software. Cada falla de éste indica un error en el diseño o en el proceso que tradujo el diseño a código ejecutable por la máquina. Entonces, las tareas de mantenimiento del software, que incluyen la satisfacción de peticiones de cambios, involucran una complejidad considerablemente mayor que el mantenimiento del hardware.

3 - Aunque la industria se mueve hacia la construcción basada en componentes, la mayor parte del software se construye para un uso individualizado.

A medida que evoluciona una disciplina de ingeniería, se crea un conjunto de componentes estandarizados para el diseño. Los tornillos estándar y los circuitos integrados preconstruidos son sólo dos de los miles de componentes estándar que utilizan los ingenieros mecánicos y eléctricos conforme diseñan nuevos sistemas. Los componentes reutilizables han sido creados para que el ingeniero pueda concentrarse en los elementos verdaderamente innovadores de un diseño; es decir, en las partes de éste que representan algo nuevo. En el mundo del hardware, volver a usar componentes es una parte natural del proceso de ingeniería. En el del software, es algo que apenas ha empezado a hacerse a gran escala. Un componente de software debe diseñarse e implementarse de modo que pueda volverse a usar en muchos programas diferentes. Los modernos componentes reutilizables incorporan tanto los datos como el procesamiento que se les aplica, lo que permite que el ingeniero de software cree nuevas aplicaciones a partir de partes susceptibles de volverse a usar. Por ejemplo, las actuales interfaces interactivas de usuario se construyen con componentes reutilizables que permiten la creación de ventanas gráficas, menús desplegables y una amplia variedad de mecanismos de interacción. Las estructuras de datos y el detalle de procesamiento que se requieren para construir la interfaz están contenidos en una librería de componentes reusables para tal fin.

2.2.2 Dominios de aplicación del software

Actualmente, hay siete grandes categorías de software de computadora que plantean retos continuos a los ingenieros de software:

Software de sistemas: conjunto de programas escritos para dar servicio a otros programas. Determinado software de sistemas (por ejemplo, compiladores, editores y herramientas para administrar archivos) procesa estructuras de información complejas pero deterministas. Otras aplicaciones de sistemas (por ejemplo, componentes de sistemas operativos, manejadores, software de redes, procesadores de telecomunicaciones) procesan sobre todo datos indeterminados. En cualquier caso, el área de software de sistemas se caracteriza por: gran interacción con el hardware de la computadora, uso intensivo por parte de usuarios múltiples, operación concurrente que requiere la secuenciación, recursos compartidos y

administración de un proceso sofisticado, estructuras complejas de datos e interfaces externas múltiples.

Software de aplicación: programas aislados que resuelven una necesidad específica de negocios. Las aplicaciones en esta área procesan datos comerciales o técnicos en una forma que facilita las operaciones de negocios o la toma de decisiones administrativas o técnicas. Además de las aplicaciones convencionales de procesamiento de datos, el software de aplicación se usa para controlar funciones de negocios en tiempo real (por ejemplo, procesamiento de transacciones en punto de venta, control de procesos de manufactura en tiempo real).

Software de ingeniería y ciencias: se ha caracterizado por algoritmos devoradores de números. Las aplicaciones van de la astronomía a la vulcanología, del análisis de tensiones en automóviles a la dinámica orbital del transbordador espacial, y de la biología molecular a la manufactura automatizada. Sin embargo, las aplicaciones modernas dentro del área de la ingeniería y las ciencias están abandonando los algoritmos numéricos convencionales. El diseño asistido por computadora, la simulación de sistemas y otras aplicaciones interactivas, han comenzado a hacerse en tiempo real e incluso han tomado características del software de sistemas.

Software incrustado: reside dentro de un producto o sistema y se usa para implementar y controlar características y funciones para el usuario final y para el sistema en sí. El software incrustado ejecuta funciones limitadas y particulares (por ejemplo, control del tablero de un horno de microondas) o provee una capacidad significativa de funcionamiento y control (funciones digitales en un automóvil, como el control del combustible, del tablero de control y de los sistemas de frenado).

Software de línea de productos: es diseñado para proporcionar una capacidad específica para uso de muchos consumidores diferentes. El software de línea de productos se centra en algún mercado limitado y particular (por ejemplo, control del inventario de productos) o se dirige a mercados masivos de consumidores (procesamiento de textos, hojas de cálculo, gráficas por computadora, multimedios, entretenimiento, administración de base de datos y aplicaciones para finanzas personales o de negocios).

Aplicaciones web: llamadas "webapps", esta categoría de software centrado en redes agrupa una amplia gama de aplicaciones. En su forma más sencilla, las webapps son poco más que un conjunto de archivos de hipertexto vinculados que presentan información con uso de texto y gráficas limitadas. Sin embargo, desde que surgió Web 2.0, las webapps están evolucionando hacia ambientes de cómputo sofisticados que no sólo proveen características aisladas, funciones de cómputo y contenido para el usuario final, sino que también están integradas con bases de datos corporativas y aplicaciones de negocios.

Software de inteligencia artificial: hace uso de algoritmos no numéricos para resolver problemas complejos que no son fáciles de tratar computacionalmente o con el análisis directo. Las aplicaciones en esta área incluyen robótica, sistemas expertos, reconocimiento de patrones (imagen y voz), redes neurales artificiales, demostración de teoremas y juegos.

Son millones de ingenieros de software en todo el mundo los que trabajan duro en proyectos de software en una o más de estas categorías. En ciertos casos se elaboran sistemas nuevos, pero en muchos otros se corrigen, adaptan y mejoran aplicaciones ya existentes. No es raro que una joven ingeniera de software trabaje en un programa de mayor edad que la de ella… Las generaciones pasadas de los trabajadores del software dejaron un legado en cada una de las categorías mencionadas. Por fortuna, la herencia que dejará la actual generación aligerará la carga de los futuros ingenieros de software.

Computación en un mundo abierto: el rápido crecimiento de las redes inalámbricas quizá lleve pronto a la computación verdaderamente ubicua y distribuida. El reto para los ingenieros de software será desarrollar software de sistemas y aplicación que permita a dispositivos móviles, computadoras personales y sistemas empresariales comunicarse a través de redes enormes.

Construcción de redes: la red mundial (World Wide Web) se está convirtiendo con rapidez tanto en un motor de computación como en un proveedor de contenido. El desafío para los ingenieros de software es hacer arquitecturas sencillas (por ejemplo, planeación financiera personal y aplicaciones sofisticadas que proporcionen un beneficio a mercados objetivo de usuarios finales en todo el mundo).

Fuente abierta: tendencia creciente que da como resultado la distribución de código fuente para aplicaciones de sistemas (por ejemplo, sistemas operativos, bases de datos y ambientes de desarrollo) de modo que mucha gente pueda contribuir a su desarrollo. El desafío para los ingenieros de software es elaborar código fuente que sea autodescriptivo, y también, lo que es más importante, desarrollar técnicas que permitirán tanto a los consumidores como a los desarrolladores saber cuáles son los cambios hechos y cómo se manifiestan dentro del software.

Es indudable que cada uno de estos nuevos retos obedecerá a la ley de las consecuencias imprevistas y tendrá efectos (para hombres de negocios, ingenieros de software y usuarios finales) que hoy no pueden predecirse. Sin embargo, los ingenieros de software pueden prepararse desarrollando un proceso que sea ágil y suficientemente adaptable para que acepte los cambios profundos en la tecnología y las reglas de los negocios que seguramente surgirán en la década siguiente.

2.2.3 Ingeniería de software

- El software se ha incrustado profundamente en casi todos los aspectos de nuestras vidas y, como consecuencia, el número de personas que tienen interés en las características y funciones que brinda una aplicación específica ha crecido en forma notable. Cuando ha de construirse una aplicación nueva o sistema incrustado, deben escucharse muchas opiniones. Y en ocasiones parece que cada una de ellas tiene una idea un poco distinta de cuáles características y funciones debiera tener el software. Se concluye que debe hacerse un esfuerzo concertado para entender el problema antes de desarrollar una aplicación de software.

- Los requerimientos de la tecnología de la información que demandan los individuos, negocios y gobiernos se hacen más complejos con cada año que pasa. En la actualidad, grandes equipos de personas crean programas de cómputo que antes eran elaborados por un solo individuo. El software sofisticado, que alguna vez se implementó en un ambiente de cómputo predecible y autocontenido, hoy en día se halla incrustado en el interior de todo, desde la electrónica de consumo hasta dispositivos médicos o sistemas de armamento. La complejidad de estos nuevos sistemas y productos basados en computadora demanda atención cuidadosa a las

interacciones de todos los elementos del sistema. Se concluye que el diseño se ha vuelto una actividad crucial.

- Los individuos, negocios y gobiernos dependen cada vez más del software para tomar decisiones estratégicas y tácticas, así como para sus operaciones y control cotidianos. Si el software falla, las personas y empresas grandes pueden experimentar desde un inconveniente menor hasta fallas catastróficas. Se concluye que el software debe tener alta calidad.

- A medida que aumenta el valor percibido de una aplicación específica se incrementa la probabilidad de que su base de usuarios y longevidad también crezcan. Conforme se extienda su base de usuarios y el tiempo de uso, las demandas para adaptarla y mejorarla también crecerán. Se concluye que el software debe tener facilidad para recibir mantenimiento.

Estas realidades simples llevan a una conclusión: debe hacerse ingeniería con el software en todas sus formas y a través de todos sus dominios de aplicación. Y esto conduce al tema de este libro: la ingeniería de software. Aunque cientos de autores han desarrollado definiciones personales de la ingeniería de software, la propuesta por Fritz Bauer (Naur & Randall, 1969) en la conferencia fundamental sobre el tema todavía sirve como base para el análisis:

La ingeniería de software es el establecimiento y uso de principios fundamentales de la ingeniería con objeto de desarrollar en forma económica software que sea confiable y que trabaje con eficiencia en máquinas reales.

El lector se sentirá tentado de ampliar esta definición.9 Dice poco sobre los aspectos técnicos de la calidad del software; no habla directamente de la necesidad de satisfacer a los consumidores ni de entregar el producto a tiempo; omite mencionar la importancia de la medición y la metrología; no establece la importancia de un proceso eficaz.

No obstante, la definición de Bauer proporciona una base:

- ¿Cuáles son los “principios fundamentales de la ingeniería” que pueden aplicarse al desarrollo del software de computadora?

- ¿Cómo se desarrolla software “en forma económica” y que sea “confiable”?
- ¿Qué se requiere para crear programas de cómputo que trabajen con “eficiencia”, no en una sino en muchas “máquinas reales” diferentes?

Éstas son las preguntas que siguen siendo un reto para los ingenieros de software.

2.3 MARCO LEGAL

Si bien es cierto el tema de la contratación de software, es un tema nuevo y poco experimentado para la legislación chilena, tenemos una norma base que puede ir dando el sustento de lo que sería una futura reglamentación y formalización legal de una contratación electrónica, nos referimos a la protección de los consumidores que se menciona en la regulación contenida en la ley 19.496 sobre protección de los derechos de los consumidores. Esta ley detalla claramente como son respaldados los consumidores pese a que sus contratos sean de manera virtual y no a lo que acostumbramos a hacer personalmente.

Es por ello, que consideramos muy apropiado tener como base la ley anteriormente mencionada, por consiguiente, queremos hacer una comparación entre lo que trata esta ley referente a los contratos y cómo se abordan estos, tanto del punto de vista del derecho civil y del derecho comercial, sin perder la perspectiva comparativa que explicaría, por qué es tan importante una pronta legislación de este tema en chile.

Entonces entrando en materia, sabemos que uno de los requisitos que tiene todo contrato es el consentimiento de quienes participan en la celebración de este, pero ¿en las distintas ramas del Derecho la formación del consentimiento se manifiesta de la misma forma? En el Derecho Civil, el Código Civil establece en el artículo 1445, como regla suprema, para que alguien pueda quedar obligado a otro por un acto o declaración de voluntad, es necesario que se consienta en dicho acto o declaración y que tal consentimiento no adolezca de vicios (aparte del objeto, la causa y la capacidad). y aunque este mismo código no regula la formación de este consentimiento, da reglas claras de sobre eventuales vicios que pueden adolecer, a fin de asegurar la libertad de contratación, tanto respecto con quien se contrata, como del contrato mismo que se celebra, y de su objeto. Son atentados graves a la libertad de contratar, el dolo y la fuerza, que la ley los califica como vicios del consentimiento.

A diferencia de la rama del Derecho Comercial, la cual dispuso reglas sobre formación del consentimiento, que se entiende que son también aplicables a los contratos civiles (artículos 97 a 106). Todo este conjunto de normas permite afirmar el principio de la autonomía de la voluntad o autonomía privada.

La clave en los contratos civiles es la voluntad del consentimiento, y ello se advierte no sólo en lo que dispone el artículo 1445 del Código Civil, sino también en la norma contenida en el artículo 1545 del mismo Código, por la cual, todo contrato legalmente celebrado es una ley para los contratantes y no puede ser invalidado sino por consentimiento mutuo o por causas legales, consagrándose el tradicional principio del *pacta sunt servanda*. En materia de interpretación contractual, la doctrina jurisprudencial y de los autores ha encontrado en el artículo 1560 del Código Civil, una regla que también descansa en el principio de autonomía de la voluntad, cuando ordena, como primera regla interpretativa, aplicar la intención de los contratantes, si ella es claramente conocida, respecto de la letra del mismo contrato. Esto significa que la voluntad contractual (la intención única de los contratantes), es un elemento que no solo da vida al contrato, sino que, además, orienta su ejecución hasta el total cumplimiento del mismo.

La libertad contractual se sostiene en el supuesto de que las personas son libres e iguales, y por lo tanto, protegiéndose esa libertad e igualdad consustancial a todo ser humano, el resultado de toda negociación contractual debe ser aceptado. Para asegurar esa libertad de negociación el Código Civil pone el acento en la libertad de consentir y, por lo mismo, asegura a las partes la posibilidad de pedir la nulidad del acto, cuando su consentimiento aparece afectado por error, por fuerza o por dolo.

Entonces, si vamos analizando para celebrar un contrato partimos de bases como el consentimiento, la igualdad, la libertad de quienes participan en el acto de contratación y de que todos tienen la libertad de dicho acto. en base a esto, los participantes de la contratación de software ¿tienen los mismos derechos, copulan para ellos las mismas libertades y consentimiento? ¿tienen ellos la posibilidad de retractarse cuando no están conformes con dicha contratación?.

Cabe señalar que, en cuanto a los deberes de información en las tratativas preliminares, no existe un principio general en Derecho Civil y Comercial que obligue a los que negocian un contrato a ser plenamente transparentes, es decir, a dar plena información sobre lo que se negocia, aunque con el tiempo se han ido perfilando deberes de informar previos a la celebración del contrato, esto quiere decir que ¿no es necesario que los contratos de software tengan todo debidamente estipulado en el contrato? ¿qué pasa si un cliente de una empresa desarrolladora de software no queda conforme con algunas de las funciones o

diseño de su producto? ¿tiene este cliente derecho de exigir que se modifiquen lo que no le gusto, o expresar fuera de contrato lo que no dijo antes de su redacción?.

Si bien es cierto que estos contratos son nuevos por la materia en que versan, no debemos basarnos en ello para no cumplir con las exigencias mínimas de un producto, con la finalidad de satisfacer las necesidades del cliente y, es aquí, donde cabe mencionar lo que ya hemos explicado y expuesto anteriormente, de cómo es la toma de requerimientos de un producto. Y es en esta línea, la contribución que deseo aportar con mi trabajo de AFET, lograr una unificación de criterios a la hora de la toma de requerimientos, para que cuando se elabore el contrato la cantidad de falencias sean menores o prácticamente ninguna.

Si a lo anterior agregamos el reforzamiento que la propia Constitución chilena establece a la libertad de contratación, al asegurar y garantizar como Derecho Constitucional, que todas las personas son libres para desarrollar cualquier actividad económica (CFR artículo 19 n° 21), lo que supone la libertad de asociación empresarial y de negociación contractual, quiere decir que, en nuestro sistema civil y comercial, la autonomía de la voluntad es un principio fundamental. Mencionamos esto con el fin de respaldar la libertad de contratación, donde nuestra máxima ley lo dispone y lo ampara.

Ahora, lo que más nos importa, dentro de esta controversia es la protección de los derechos de los consumidores y la autonomía privada, para esto hay unos principios que lo resguardan:

2.3.1 Principio de transparencia

Si revisamos ahora las reglas de la Ley 19.496, podemos advertir que el paradigma de la autonomía privada no encaja con la misma profundidad en los actos de consumo. El supuesto de fondo que se suele invocar para explicar la estricta regulación de los actos sujetos a la ley del consumidor, dicen relación con un desequilibrio que se advierte entre proveedores de bienes y servicios, y los consumidores, en términos de una fuerza económica superior o muy superior, dependiendo de los casos, de una falta de equilibrio o asimetría en materia de información y de la necesidad de evitar, por lo mismo, abusos en contra de los consumidores. Más al fondo de la cuestión, aparece un tema de orden público económico, por el cual, se postula que el desarrollo de la economía y del consumo exige ofrecer confianza a los consumidores, en donde al proveedor no solo se le exige

transparencia en la oferta sino en general lealtad en el cumplimiento de buenas prácticas comerciales, la buena fe llega entonces a exigir una operación comercial respetuosa de los consumidores.

No se trata ya de proteger la pura libertad contractual, en el sentido de asegurar libertad a la espontánea decisión de contratar, sino de garantizar a los consumidores que los bienes y servicios que se les ofrecen podrán adquirirlos, o servirse de ellos, en los términos que ellos están siendo ofrecidos, que no serán sometidos a condiciones inicuas, que se les respetarán las condiciones y modalidades ofrecidas, que no serán dañados o menoscabados, entre otras cosas y, en general, no serán sometidos a prácticas comerciales desleales. Por ello, parece que el bien superior ya no es asegurar la clásica libertad de contratación, sino proteger el acto de consumo masivo, a partir de la confianza que le ha suscitado la propuesta del proveedor (que en nuestra materia sería el ingeniero de software que desarrollaría el producto a entregar) . El contraste, por lo mismo, no se hace entre lo que se convino y lo que se recibió, sino entre lo que se ofreció por el proveedor (ingeniero o desarrollador) y lo que efectivamente este entregó. Se introduce, así, un principio rector en el derecho del consumo, que es el principio de la transparencia del proveedor. Es una depuración cuya finalidad no busca dar garantía para el logro de un consentimiento libre, sino generar condiciones para un consumo libre y confiado.

El principio de la transparencia queda recogido en la Ley 19.456. Así, el artículo 1 nº2 consagra la categoría "información básica comercial", que debe ser entregada al consumidor; el artículo 3 b) configura en favor de los consumidores el derecho a una información veraz y oportuna sobre los bienes y servicios ofrecidos su precio, condiciones de contratación y otras características relevantes de los mismos, y el deber de informarse responsablemente de ellos"; por su parte, el artículo 12 de la ley dispone que todo proveedor de bienes o servicios "estará obligado a respetar los términos, condiciones y modalidades conforme a las cuales se hubiere ofrecido o convenido con el consumidor la entrega del bien o la prestación del servicio". Si leemos bien esta última disposición, el proveedor no solo está obligado a respetar los términos, condiciones o modalidades convenidas, sino las que "hubiere ofrecido", con lo cual, es claro que la plena información pasa a ser un elemento relevante en la relación de consumo (entrega del producto final “software”).

Entonces podemos concluir que, mientras en el sistema civil y comercial, sólo por excepción, existen deberes de información expresamente reconocidos, y otros que derivan de una manera general del principio de buena fe contractual, para el derecho del consumo estos están en su base. Mientras en el Código Civil y de Comercio la protección al consentimiento libre se configura por vía de reacción, anulando contratos que aparecen viciados y por causas tipificadas, en la ley de protección a los derechos de los consumidores existe un sistema preventivo, pues, se asegura al consumidor la plena información. Más que la voluntad negocial, lo que se busca es dar protección a la expectativa de un consumo seguro y de acuerdo con lo que se ha ofrecido.

2.3.2 Restricciones a la libertad de contratación y a la manifestación de la voluntad de contratar

Como idea primera, debe decirse que todo el sistema de derechos que la Ley 19.496 consagra en favor de los consumidores, se constituye en una barrera infranqueable para la libertad contractual, pues el artículo 4 estatuye que los derechos de los consumidores no pueden ser renunciados anticipadamente; en los actos de consumo, el ámbito de lo negociable queda constreñido fuertemente, a diferencia de lo que dispone el Código Civil en su artículo 12, que asegura la renuncia a los derechos conferidos por las leyes, con tal que la renuncia mire al solo interés individual del renunciante y no esté prohibida la renuncia.

Este es un aspecto que nos parece central, debido a que pone en la raíz del sistema del derecho del consumidor un criterio protector, con lo cual, el paradigma clásico del Derecho Civil y Comercial, de la libertad de contratación queda fuertemente reducido, desde el momento que existe un elenco amplio de derechos del consumidor, que no pueden ser renunciados.

En cuanto a la forma o manera en que la voluntad debe ser manifestada en materia de consumo, tenemos dos disposiciones que conviene recordar, porque también aparecen parcialmente modificadas respecto a lo que es el derecho común. Por una parte, desde el punto de vista del consumidor, el artículo 3 letra a) asegura la libertad de elección del bien o servicio, y por otra, el silencio no constituye aceptación en los actos de consumo. Esto significa que la manifestación de voluntad en los actos de consumo, por parte del

consumidor, está sujeta a una importante restricción, entendiendo por silencio a la falta de expresión de voluntad, fundamentalmente por la vía del lenguaje oral o escrito.

En consecuencia de la falta de manifestación o expresión de voluntad, no puede derivarse la aceptación del consumidor. La norma tiene interés para los efectos de analizar la posible validez de aceptaciones tácitas, es decir, de aquellas que no se derivan de una explícita, clara e inequívoca manifestación de consentimiento por parte del consumidor. Ellas no podrán admitirse con tanta amplitud como en el derecho común, pues, pueden esconder abusos, en el sentido de que el consumidor no ha conocido exactamente la propuesta que está aceptando. Incluso más, la misma disposición permite rechazar la posibilidad de que el consumidor pueda ser compelido por el proveedor, de que su no expresión se tendrá como aceptación o desistimiento. Naturalmente el acto mismo de consumo sí es aceptación, pero únicamente en cuenta expresa la voluntad de recibir el bien o servicio ofrecido. Por lo mismo, las aceptaciones tácitas, como la que acabamos de referir, están severamente restringidas en el derecho de consumo.

En lo que se refiere a la libertad de contratación, en su dimensión de la autonomía para escoger, ella solo está protegida para el consumidor, quien, no solo tiene derecho a la libre elección del bien o servicio a contratar, sino que, además, de acuerdo con el artículo 3 letra c) de la Ley 19.496, tiene derecho para no ser discriminado arbitrariamente, por lo que el proveedor debe someter a todos los consumidores a condiciones similares, sin distinguir entre ellos sin fundamento objetivo y racional. Más aún, el artículo 13 de la Ley dispone que "los proveedores no podrán negar injustificadamente la venta de bienes o la prestación de servicios comprendidos en sus respectivos giros en las condiciones ofrecidas". Si se aprecia, dado que el proveedor no puede incurrir en discriminaciones arbitrarias, su libertad de contratación, en la dimensión de cuándo, con quién y qué contratar, está severamente disminuida.

En resumen, creemos que con estos principios, nos queda más que clara la necesidad de legislar de manera pronta y correcta lo que serán los contratos de ahora en adelante donde la vulneración de los derechos por falta de legislación se vuelve cada día más significativo, donde podríamos evitar exceso de demandas por incumplimiento de cláusulas, contratos, o requerimientos malentendidos que no lograron realizar el fin del producto requerido.

Un término que quisiéramos reflejar que nos ayudaría bastante a entender el sistema de contratación chileno, es el EL PACTA SUNT SERVANDA, una vez celebrado un acto de consumo, los términos en que este debe ejecutarse superan con mucho a la regla del *pacta sunt servanda* o de la intención de los contratantes como regla matriz para ejecutar un contrato, según antes recordábamos. En efecto, el mismo artículo 12 de la Ley 19.496 indica como principal deber legal del proveedor el de respetar los términos, condiciones o modalidades ofrecidas, norma similar al artículo 1545 del Código Civil. Sin embargo, no existe una regulación recíproca para el consumidor, al que se le reconoce, en algunos casos, el derecho de retracto, en los términos que establece el artículo 3 bis de la Ley 19.496.

Si se revisa con atención el contenido mismo de lo que el proveedor debe dar o prestar, ello no lo determina exclusivamente el contrato, es decir, lo estrictamente convenido, pues, la lectura del artículo 20 de la Ley 19.496 permite afirmar que existe un ámbito de protección del consumidor que tiene carácter amplio y objetivo, en el sentido que debe darse o prestarse la cosa o servicio ofrecido, más allá de los términos que pudieran haberse pactado. De hecho, el artículo 19 confiere al consumidor el derecho de reposición del bien, a la devolución del dinero pagado en exceso, o la bonificación de su valor para la compra de otro producto, cuando no hubiera recibido una cantidad o contenido neto de un producto distinto de la que aparece en el "envase o empaque". Es decir, no es la convención la que en estos casos determina lo que se debía, sino lo que objetivamente estaba indicado, como cantidad o peso contenido, en el bien o producto.

2.3.3 Las consecuencias que derivan del incumplimiento de la ley

A continuación nos referimos a las consecuencias o efectos que se derivan del incumplimiento de la Ley 19.496, en comparación con las reglas generales sobre incumplimiento contractual. Estamos conscientes que en esta materia, la ley que estudiamos ofrece un sistema disperso, y con serias falencias de consistencia, lo que se ha prestado para muchas dificultades interpretativas; tal vez la más saliente es la contravención que se le ha dado a sus disposiciones, ello seguramente influido por la ley que la precedió.

Como se sabe, la doctrina tradicional chilena explica que del incumplimiento de una obligación pueden derivar varias consecuencias, entre ellas, la ejecución forzosa de la misma, los daños y perjuicios, y otros derechos auxiliares del acreedor. Esta estructura

aparece ampliada y reforzada en el sistema de remedios procesales que ofrece la Ley 19.496.

La Ley de protección de los derechos de los consumidores tiene un carácter mixto, porque junto con consagrar acciones reparatorias derivadas del incumplimiento, también establece sanciones controvertidas que están penadas con multas (entre otras artículos 9, 12, 13, 14, 31, 32, 33, 35 inciso 3˚, 23 inciso 2˚, 29, 45 inciso 1˚, y sin perjuicio de contemplar una sanción residual en el artículo 24 inciso 1˚). Se ha discutido la conexión que existen entre ambas, dada la competencia que, como regla, la Ley le confirió a los Juzgados de Policía Local. Para una amplia jurisprudencia es presupuesto de las acciones civiles el que pueda perseguirse la responsabilidad contravencional, la que conforme con el artículo 26 de la Ley, prescribe en 6 meses desde que se ha incurrido en la infracción respectiva, lo que supondría que pasado este plazo, al perder el tribunal la competencia para sancionar la infracción, tampoco podría ejercerse la acción civil indemnizatoria, por ejemplo.

La Ley de protección de derechos de los consumidores confiere al afectado, entre otros derechos, lo que se ha denominado la responsabilidad derivada de la garantía legal, que conforme con el artículo 20 la Ley 19.496, ofrece varias opciones, todo ello sin perjuicio del derecho a la indemnización por los daños ocasionados. Existen en este caso cuatro acciones distintas:

1. La reparación gratuita del bien
2. La reposición del bien (previa devolución de lo recibido)
3. La restitución de la cantidad pagada (previa devolución de lo recibido)
4. La indemnización complementaria por los daños causados, todo lo cual demuestra el amplio espectro de acciones que la ley de protección de los derechos de los consumidores ofrece en caso de venta de un producto.

Paralelamente, tenemos la responsabilidad civil derivada del ilícito infraccional, contemplados en un serie de disposiciones, entre otras el artículo 23 y que está formulada de la siguiente manera: "Artículo 23.- Comete infracción a las disposiciones de la presente ley el proveedor que, en la venta de un bien o en la prestación de un servicio, actuando con negligencia, causa menoscabo al consumidor debido a fallas o deficiencias en la calidad,

cantidad, identidad, sustancia, procedencia, seguridad, peso o medida del respectivo bien o servicio".

Los contratos de prestación de servicios

Respecto de las prestaciones de servicios, también cabe destacar el deber u obligación de garantía legal, con amplia cobertura de servicios, según se lee del artículo 41, y que consiste en la necesidad en que queda el proveedor de responder por el servicio prestado y de su reparación, dentro de un plazo que debe indicar en el recibo, escrito o boleta que entregue. Esta garantía apunta a asegurar la reiteración del servicio, la devolución del precio recibido, si fuere del caso, y a la indemnización de todo otro perjuicio al consumidor. En el artículo 40 de la Ley 19.496, se dispone que en los contratos de prestación de servicios cuyo objeto sea la reparación de cualquier tipo de bienes, se entenderá implícita la obligación del prestador del servicio de emplear, en tal reparación, componentes o repuestos adecuados al bien de que se trate, ya sean nuevos o refaccionados, siempre que se informe al consumidor de esta circunstancia. El incumplimiento de esta obligación dará lugar, además de las sanciones e indemnizaciones que procedan, a exigir la sustitución de los componentes o repuestos correspondientes, sin cargo adicional.

2.3.4 Justificación del tema de investigación

La justificación del presente tema de investigación se centra en los análisis obtenidos de distintos casos de contratos entre clientes del sector público con empresas del sector privado, en donde se pueden observar una serie de controversias por malas definiciones a la hora de la realización del contrato de la prestación del servicio.

Entre los casos abordados, tenemos el contrato entre la Dirección de Presupuesto (DIPRES) y Everis Chile S.A., en donde se solicitó a la empresa desarrolladora elaborar el sistema de presupuesto para el estado (SIGFE 2.0). Analizando este contrato, se pueden observar diversos errores en las definiciones de las funcionalidades, cualidades, requerimientos y observaciones técnicas del sistema a desarrollar.

Según estudios del equipo Standish Group, podemos analizar la siguiente comparativa de casos exitosos, completados con dificultades y fracasados, como se ve en la tabla 2-01:

Tabla N°2-01 - Informe Chaos de proyectos de ingeniería de software

	SUCCESSFUL	CHALLENGED	FAILED
Grand	2%	7%	17%
Large	6%	17%	24%
Medium	9%	26%	31%
Moderate	21%	32%	17%
Small	62%	16%	11%
TOTAL	100%	100%	100%

En base a nuestra experiencia profesional, se detallan 60 problemas que acontecieron en el presente proyecto informático de gran envergadura. Se han tenido en cuenta únicamente las dos etapas de desarrollo descritas en el fallo que analiza el desarrollo y posterior entrega del sistema presupuestario (SIGFE 2.0).

Se han agrupado en tres grupos, los problemas relacionados con los contratos (20), los problemas orientados a definiciones técnicas (20) y otros problemas generales (20).

Problemas contractuales

1. No tener definido quién es la contraparte en el cliente
2. No tener definido que se entregará
3. No tener definido cómo se acepta el proyecto
4. No tener definido cuándo se entregará algo
5. No tener definido los niveles de soporte
6. No tener definida la capacitación
7. No cuantificar todo
8. No requerir y disponer de infraestructura de trabajo: espacio físico, sillas, escritorios, computadoras, teléfonos
9. No contar con acceso fuera del horario de oficina

10. No especificar el costo de las horas adicionales
11. No contar con la propiedad del código o de las fuentes
12. No especificar soporte postproducción
13. No especificar mantenimiento
14. Realizar cotizaciones incompletas
15. No especificar cómo tratar los temas relativos a la confidencialidad
16. No considerar aspectos impositivos y tributarios del proyecto
17. No determinar qué hacer ante problemas con personas que participan
18. Composición del equipo de trabajo
19. Rotación del equipo
20. No prevenir los fracasos

Problemas técnicos

1. No tener suficiente variedad de pruebas
2. No tener un plan de calidad
3. No tener un grupo independiente de pruebas
4. No contar con clasificación ni cuantificación de errores
5. No tener coordinación entre grupos
6. No tener control presupuestario
7. No tener un flujo de fondos del proyecto
8. No hacer reportes de avance
9. No prever problemas de infraestructura
10. No prever problemas de licencias necesarias para el desarrollo
11. No tener un plan
12. No actualizar el plan
13. Grandes barras y problema del 90%
14. Quién controla lo que hace el programador
15. No identificar prerrequisitos
16. No identificar dependencias críticas o entregables críticos
17. No considerar ni manejar los riesgos
18. No tener control de cambios
19. No tener documentación
20. Comenzar sin un acuerdo formal

Problemas generales

1. Hacer proyectos muy largos
2. No tener entornos de prueba
3. Tener conocimiento centrado en pocas personas
4. Tener problemas de idioma
5. No prever vacaciones, enfermedades, embarazos, tiempos de traslado, curvas de aprendizaje, entre otros aspectos
6. Rompecabezas tecnológico
7. Usar la última tecnología disponible
8. Sobreventa de recursos
9. Construir el sistema que nadie necesita
10. No tener a nadie que apruebe o que firme
11. No integrar la aplicación (Torre de Babel)
12. Arquitectura de la solución
13. Fanáticos de una tecnología
14. Mala comunicación
15. Limpieza de datos
16. Mantenibilidad de la aplicación
17. Construir o comprar
18. Proyectos multipaís y multihorario
19. Bajo nivel de usuarios
20. Terminología común en contratos y proyecto

Al observar estos tipos de problemas en el contrato, podemos prever un resultado no esperado en la entrega del producto final, tal como ocurrió en este proyecto, donde se tuvo que licitar en una segunda etapa una adaptación del sistema ya desarrollado para cumplir con el 100% de aceptación por parte del cliente (Dirección de presupuesto).

2.3.5 Análisis jurídico de las cláusulas del contrato

Analizando el caso de investigación de la Contraloría de la República sobre el contrato de Everis Chile S.A. y la Dirección de Presupuesto (DIPRES), encontramos muchas cláusulas que fueron incumplidas, las cuales por su parte, podrían haber sido evitadas con buenas definiciones en los requerimientos y proceso de desarrollo del software.

Por consiguiente y para que se entienda mejor, tomaremos ciertos extractos de lo que fue el informe final de la investigación realizada por parte de la Contraloría División de Auditoría Administrativa (área hacienda, economia y fomento), con el fin de poder argumentar y dar una estructura válida de que debido a no contar con una legislación específica para casos como estos, se cometen errores que pueden ser evitados si existiera un patrón guía.

A continuación analizaremos las siguientes cláusulas con el fin de dar validez jurídica a nuestra investigación:

1.- En el mes de octubre del año 2010 DIPRES formuló el primer plan de implantación del SIGFE 2.0, en el cual se establece que 169 instituciones participarán en la ejecución del aplicativo, el cual fue modificado dos veces en el año 2011, producto de problemas relacionados con la velocidad de respuesta del sistema desarrollado por Everis Chile S.A., lo que causó la suspensión temporal del proceso de ejecución de dicho plan.

2.- Luego, en el año 2012, la DIPRES elaboró un nuevo plan de implantación con vigencia al 31 de diciembre de ese año, el que se ajustó 3 veces, rebajandose a 85 instituciones, de las cuales 15 postergaron su implantación para el año 2013, esta vez, por problemas de inestabilidad en la reportabilidad y otros inconvenientes operacionales.

3.- Por otra parte, es importante señalar que el plan de implantación 2012 no contempló el funcionamiento de la operatividad del total de las funcionalidades del sistema. entre otras, banco de bienes, conciliaciones bancarias e interoperabilidad, incluyendo solo la operatoria básica del mismo, sin identificar los plazos para la implementación gradual de todas las aplicaciones del sistema. Lo anterior, da cuenta de la falta de control en la ejecución de este plan, lo que pugna con lo previsto en la letra e), sobre supervisión. del capítulo 111, de las normas específicas, de la resolución exenta N° 1.485, de 1996, de esta Contraloría General, que Aprueba Normas de Control Interno.

4.- Asimismo, cabe señalar que de los antecedentes que mantiene este Organismo de Control, a esa misma fecha, se constató que de las 51 entidades en producción informadas por la DIPRES, sólo 18 habían efectuado con éxito el proceso de cierre contable, presentando problemas en las otras 33.

5.- Al respecto, cabe observar que los planes señalados precedentemente corresponden a minutas confeccionadas por el equipo de implantación de la DIPRES, que fueron suscritas por el jefe de la Unidad Ejecutora Proyecto, UEP, SIGFE 2.0, y que no se encuentran aprobadas formalmente. contraviniendo con ello lo dispuesto en el artículo 3º de la ley Nº19.880, que Establece Bases de los Procedimientos Administrativos que Rigen los Actos de los Órganos de la Administración del Estado, el cual señala que las decisiones que adopte la administración, se expresarán por medio de actos formales emitidos por los órganos del Estado, los que contienen declaraciones de voluntad, realizadas en el ejercicio de la potestad pública.

6.- Manifiesta, que los actos a que se refiere el mencionado artículo 3º, de la citada ley Nº19.880, son aquellos que afectan los intereses de terceros, ya sea creando, extinguiendo o modificando sus derechos, por lo que, dado carácter decisorio o resolutivo de éstos, lo que hace el órgano administrativo al dictarlos, es tomar una decisión acerca de la aplicación del ordenamiento jurídico a un caso concreto, consecuencia que no se produce con el diseño del plan de implementación.

De los puntos anteriormente observados, hacemos un análisis de los problemas que se presentaron en la puesta en marcha del proyecto:

Sobre el proyecto, corresponde hacer presente que las 33 instituciones que presentaron dificultades, las cuales se detallan en el Anexo Nº2 del análisis de la Contraloría, solicitaron a esta, las respectivas prórrogas para presentar el Balance de Comprobación y Saldos, y para cerrar el ejercicio contable 2012, información que debían entregar y procesar, hasta el 8 de enero de 2013, según lo requerido por este Organismo de Control en los dictámenes Nº80.001 y Nº80.008, ambos de 2012, aludiendo para ello problemas de funcionamiento del SIGFE versión 2.0, entre otros, inconsistencias en la información y descuadraturas aritméticas.

Ahora bien, las 33 instituciones informaron diversas dificultades para emitir los reportes denominados "Estado de Ejecución de Gastos" y "Compromiso", de los módulos de reportabilidad y tesorería, entre otras, duplicidad en las operaciones de devengos y compromisos y errores en los ajustes contables y en el pago de depósitos a terceros. Además, reportaron que la funcionalidad de cierre contable no consideraba los saldos iniciales. Asimismo, algunos servicios comunicaron que no podían efectuar los reintegros de dinero mediante la utilización de cuentas corrientes, que el sistema no permitía la impresión de cheques, y que no se había habilitado la funcionalidad para realizar la conciliación bancaria.

Con respecto al mismo tema de plazos según el informe se indica que "También señalaron que tales incidencias, comunicadas al área de ServiceDesk, de Everis Chile S.A., encargada de dar solución a las fallas del SIGFE, tuvieron retraso en su contestación", lo cual es una clara manifestación que los encargados de dar las soluciones pertinentes ya sea a través de una mesa de ayuda o de un soporte técnico presencial, vía online o telefónico, no cumplieron en los tiempos de respuesta acordes para el proyecto desarrollado.

Por otra parte, es importante señalar que el plan de implantación 2012 no contempló el funcionamiento de la operatividad del total de las funcionalidades del sistema. entre otras, banco de bienes, conciliaciones bancarias e interoperabilidad, incluyendo solo la operatoria básica del mismo, sin identificar los plazos para la implementación gradual de todas las aplicaciones del sistema.

Lo anterior, da cuenta de la falta de control en la ejecución de este plan, lo que pugna con lo previsto en la letra e), sobre supervisión. del capítulo 111**,** de las normas específicas, de la resolución exenta N° 1.485, de 1996, de esta Contraloría General, que Aprueba Normas de Control Interno.

Ley 19880, Artículo 3° "Concepto de Acto administrativo. Las decisiones escritas que adopte la Administración se expresarán por medio de actos administrativos. Para efectos de esta ley se entenderá por acto administrativo las decisiones formales que emitan los órganos de la Administración del Estado en las cuales se contienen declaraciones de voluntad, realizadas en el ejercicio de una potestad pública. Los actos administrativos tomarán la forma de decretos supremos y resoluciones.

El decreto supremo es la orden escrita que dicta el Presidente de la República o un Ministro "Por orden del Presidente de la República", sobre asuntos propios de su competencia. Las resoluciones son los actos de análoga naturaleza que dictan las autoridades administrativas dotadas de poder de decisión. Constituyen, también, actos administrativos los dictámenes o declaraciones de juicio, constancia o conocimiento que realicen los órganos de la Administración en el ejercicio de sus competencias.

Las decisiones de los órganos administrativos pluripersonales se denominan acuerdos y se llevan a efecto por medio de resoluciones de la autoridad ejecutiva de la entidad correspondiente. Los actos administrativos gozan de una presunción de legalidad, de imperio y exigibilidad frente a sus destinatarios, desde su entrada en vigencia, autorizando su ejecución de oficio por la autoridad administrativa, salvo que mediare una orden de suspensión dispuesta por la autoridad administrativa dentro del procedimiento impugnatorio o por el juez, conociendo por la vía jurisdiccional.

Según lo visto anteriormente, y de acuerdo al análisis de los datos expuestos, podemos concluir que si nos basamos en el listado que está en las páginas anteriores donde se detallan cada uno de los posibles problema, es por esto que en el mismo informe se hace referencia a futuras auditorías para ver si lo que concluyeron se formalizó “En virtud de lo expuesto, este Organismo de Control mantiene la totalidad de lo observado, hasta que en una próxima auditoría se verifique que este tipo de documentos, se encuentren debidamente formalizados y que se hayan establecido plazos para incluir la operatividad del total de las funcionalidades del SIGFE 2.0”.

Abordando el Artículo 1547 del Código Civil, nos permite acordar convenciones que alteren la responsabilidad del deudor, aumentando su responsabilidad, disminuyéndola o eliminándola. Por lo cual, nos permite establecer un monto máximo en dinero de responsabilidad en caso de incumplimiento o no se responde por lucro cesante o daños indirectos (por ejemplo, se responde solo por daño emergente) o solo se responde por daños que en forma exclusiva pueden ser atribuibles a su actuar. Este punto es importante, dado que se establecen las multas correspondientes a atrasos y/o entregas insatisfechas para el cliente por parte de la empresa desarrolladora.

El Artículo 1564 del Código Civil. nos menciona que las cláusulas de un contrato se interpretarán unas por otras, dándose a cada una el sentido que mejor convenga al contrato en su totalidad. Podrán también interpretarse por las de otro contrato en las mismas partes y sobre la misma materia. O por la aplicación práctica que hayan hecho de ellas ambas partes, o una de las partes con aprobación de la otra. Este punto es de vital importancia, ya que nos menciona sobre la aceptación de los acuerdos por ambas partes, con el fin de poder hacer criterios de aceptación en las diversas entregas del proyecto al equipo desarrollar y, no caer en malas definiciones de negocio por parte del equipo.

De todo lo anterior sin duda alguna, el hecho de que no existan bases legales, no exista un padrón a seguir en la creación de un software que se utilice en grandes proyectos como el estudiado recientemente, son desventajas que tiene nuestro sistema legal también, ya que sólo algunas de estas cláusulas podrían ser subsanadas bajo una ley que las ampare o que pueda servirles de base para una posible discusión o juicio futuro. Mientras no se logre establecer un modelo, seguirán pasando este tipo de incumplimientos de contratos, de plazos, de funcionalidades, de apoyo. Es por eso, que el presente AFET, lo que busca conseguir es sentar bases de cómo se realizaría una buena toma de requerimientos para que no sucedan errores, como los que vemos en este informe donde claramente una cantidad no menor de instituciones no lograron su objetivo final, y no porque no estuviese contemplada sino porque sus requerimientos eran diferentes a los demás, ser específico en estos casos, ayuda a concretar de mejor manera un resultado óptimo y satisfactorio.

No cabe duda que nos falta mucho por incorporar aún a nuestra legislación con respecto a lo que tecnología y avances se trata, pero sin desmerecer lo que tenemos, debemos implementar grandes cambios en pos al progreso y la modernidad, debemos ser capaces de adelantarnos a los hechos futuros, de prever los acontecimientos que podrían ser parte de nuestro acontecer diario, y este AFET y en su anexo, pretende apoyar estas circunstancias que no están fijadas en ninguna ley pero que servirán de orientación para evitar muchos de los errores que en este caso pudimos dilucidar.

2.4 ESTADO DEL ARTE

La práctica de la ingeniería de software está guiada por un conjunto de principios fundamentales que ayudan en la aplicación del proceso de software significativo y en la ejecución de métodos eficaces de ingeniería de software.

En el nivel del proceso, los principios fundamentales establecen un fundamento filosófico que guía al equipo de software cuando realiza actividades estructurales y actividades sombrilla, cuando navega por el flujo del proceso y elabora un conjunto de productos del trabajo de la ingeniería de software. En el nivel de la práctica, los principios fundamentales definen un conjunto de valores y reglas que sirven como guía cuando se analiza un problema, se diseña una solución, se implementa y prueba ésta y cuando, al final se entrega el software a la comunidad de usuarios.

Entender los requerimientos de un problema es una de las tareas más difíciles que enfrenta el ingeniero de software. Cuando se piensa por primera vez, no parece tan difícil desarrollar un entendimiento claro de los requerimientos. Después de todo, ¿acaso no sabe el cliente lo que se necesita?, ¿no deberían tener los usuarios finales una buena comprensión de las características y funciones que le darán un beneficio?. Sorprendentemente, en muchas instancias la respuesta a estas preguntas es “no”. E incluso si los clientes y los usuarios finales explican sus necesidades, éstas cambiarán mientras se desarrolla el proyecto.

Ralph Young (Young R, 2001) menciona en su libro “Un cliente entra en la oficina, toma asiento, lo mira fijamente a los ojos y le dice: Sé que cree que entiende lo que digo, pero lo que usted no entiende es que lo que digo no es lo que quiero decir. Invariablemente, esto pasa cuando ya está avanzado el proyecto, después de que se han hecho compromisos con los plazos de entrega, que hay reputaciones en juego y mucho dinero invertido.

Todos los que hemos trabajado en el negocio de los sistemas y del software durante algunos años hemos vivido la pesadilla descrita, pero pocos hemos aprendido a escapar. Batallamos cuando tratamos de obtener los requerimientos de nuestros clientes. Tenemos problemas para entender la información que obtenemos.

Es frecuente que registremos los requerimientos de manera desorganizada y que dediquemos muy poco tiempo a verificar lo que registramos. dejamos que el cambio nos

controle en lugar de establecer mecanismos para controlarlo a él. En pocas palabras, fallamos en establecer un fundamento sólido para el sistema o software. Cada uno de los problemas es difícil. Cuando se combinan, el panorama es atemorizador aún para los gerentes y profesionales más experimentados. Pero hay solución."

2.4.1 Ingeniería de requerimientos

El diseño y construcción de software de computadora es difícil, creativo y sencillamente divertido. En realidad, elaborar software es tan atractivo que muchos desarrolladores de software quieren ir directo a él antes de haber tenido el entendimiento claro de lo que se necesita. Argumentan que las cosas se aclararán a medida que lo elaboren, que los participantes en el proyecto podrán comprender sus necesidades sólo después de estudiar las primeras iteraciones del software, que las cosas cambian tan rápido que cualquier intento de entender los requerimientos en detalle es una pérdida de tiempo, que las utilidades salen de la producción de un programa que funcione y que todo lo demás es secundario. Lo que hace que estos argumentos sean tan seductores es que tienen algunos elementos de verdad[1]. Pero todos son erróneos y pueden llevar un proyecto de software al fracaso.

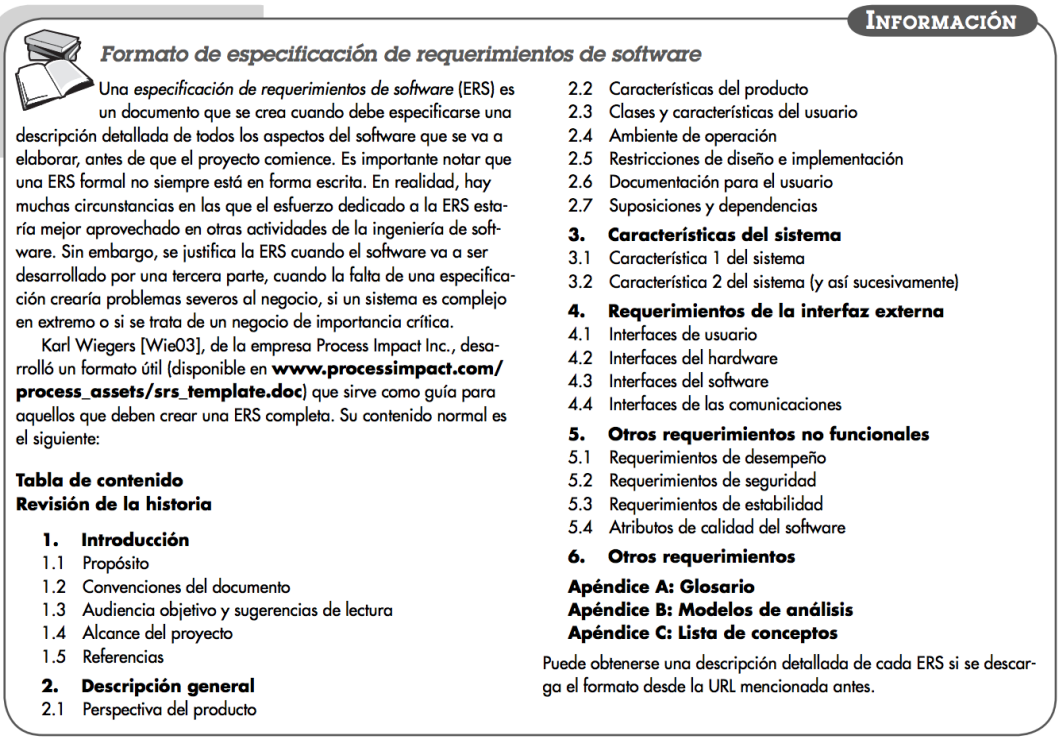
El espectro amplio de tareas y técnicas que llevan a entender los requerimientos se denomina *ingeniería de requerimientos*. Desde la perspectiva del proceso del software, la ingeniería de requerimientos es una de las acciones importantes de la ingeniería de software que comienza durante la actividad de comunicación y continúa en la de modelado. Debe adaptarse a las necesidades del proceso, del proyecto, del producto y de las personas que hacen el trabajo.

La ingeniería de requerimientos tiende un puente para el diseño y la construcción. Pero, ¿dónde se origina el puente?. Podría argumentarse que principia de los participantes del proyecto (por ejemplo, gerentes, clientes y usuarios), donde se define las necesidades del negocio, se describen los escenarios de uso, se delinean las funciones y características y se identifican las restricciones del proyecto. Otros tal vez sugieren que empieza con una definición más amplia del sistema, donde el software no es más que un componente del dominio del sistema mayor. Pero sin importar el punto de arranque, el recorrido por el

[1] Esto es cierto en particular para los proyectos pequeños (menos de un mes) y muy pequeños, que requieren relativamente poco esfuerzo de software sencillo. A medida que el software crece en tamaño y complejidad, estos argumentos comienzan a ser falsos.

puente lo lleva a uno muy alto sobre el proyecto, lo que le permite examinar el contexto del trabajo del software que debe realizarse; las necesidades específicas que deben abordar el diseño y la construcción; las prioridades que guían el orden en el que se efectúa el trabajo, y la información, las funciones y los comportamientos que tendrán un profundo efecto en el diseño resultante.

La ingeniería de requerimientos proporciona el mecanismo apropiado para entender lo que desea el cliente, analizar las necesidades, evaluar la factibilidad, negociar una solución razonable, es específica sin ambigüedades, validar la especificación y administrar los requerimientos a medida de que se transforman en un sistema funcional. Incluye 7 tareas diferentes: concepción, indagación, elaboración, negociación, especificación, validación y administración. Es importante notar que algunas de estas tareas ocurren en paralelo y que todas adaptan a las necesidades del proyecto.



INFORMACIÓN

Formato de especificación de requerimientos de software

Una *especificación de requerimientos de software* (ERS) es un documento que se crea cuando debe especificarse una descripción detallada de todos los aspectos del software que se va a elaborar, antes de que el proyecto comience. Es importante notar que una ERS formal no siempre está en forma escrita. En realidad, hay muchas circunstancias en las que el esfuerzo dedicado a la ERS estaría mejor aprovechado en otras actividades de la ingeniería de software. Sin embargo, se justifica la ERS cuando el software va a ser desarrollado por una tercera parte, cuando la falta de una especificación crearía problemas severos al negocio, si un sistema es complejo en extremo o si se trata de un negocio de importancia crítica.

Karl Wiegers [Wie03], de la empresa Process Impact Inc., desarrolló un formato útil (disponible en **www.processimpact.com/process_assets/srs_template.doc**) que sirve como guía para aquellos que deben crear una ERS completa. Su contenido normal es el siguiente:

Tabla de contenido
Revisión de la historia

1. Introducción
1.1 Propósito
1.2 Convenciones del documento
1.3 Audiencia objetivo y sugerencias de lectura
1.4 Alcance del proyecto
1.5 Referencias

2. Descripción general
2.1 Perspectiva del producto
2.2 Características del producto
2.3 Clases y características del usuario
2.4 Ambiente de operación
2.5 Restricciones de diseño e implementación
2.6 Documentación para el usuario
2.7 Suposiciones y dependencias

3. Características del sistema
3.1 Característica 1 del sistema
3.2 Característica 2 del sistema (y así sucesivamente)

4. Requerimientos de la interfaz externa
4.1 Interfaces de usuario
4.2 Interfaces del hardware
4.3 Interfaces del software
4.4 Interfaces de las comunicaciones

5. Otros requerimientos no funcionales
5.1 Requerimientos de desempeño
5.2 Requerimientos de seguridad
5.3 Requerimientos de estabilidad
5.4 Atributos de calidad del software

6. Otros requerimientos

Apéndice A: Glosario
Apéndice B: Modelos de análisis
Apéndice C: Lista de conceptos

Puede obtenerse una descripción detallada de cada ERS si se descarga el formato desde la URL mencionada antes.

Figura N°2-03 - Formato de especificación de requerimientos de software

Concepción

¿Cómo inició un proyecto de software?, ¿Existe un solo evento que se convierte en el catalizador de un nuevo sistema o producto basado en una computadora o la necesidad evoluciona en el tiempo?. no hay respuestas definitivas a estas preguntas. En ciertos casos, una conversación casual es todo lo que se necesita para desencadenar un trabajo grande de ingeniería de software.

Pero en general, la mayor parte de proyectos comienza cuando se identifica una necesidad de negocio o se descubre un nuevo mercado servicio potencial. Los participantes de la comunidad del negocio (por ejemplo: los directivos, personal de mercadotecnia, gerentes de producto, etc) definen en un caso de negocio para la idea, tratan de identificar el ritmo y profundidad del mercado, hacen un análisis de gran visión de la factibilidad e identifican una descripción funcional del alcance del proyecto. Toda esta información está sujeta cambio, pero es suficiente para desencadenar análisis con la organización de ingeniería de software[2].

En la concepción del proyecto[3], se establece el entendimiento básico del problema, las personas que quieren una solución, la naturaleza de la solución que se desea, así como la eficacia de la comunicación y la colaboración preliminares entre los participantes y el equipo de software.

Indagación

En verdad que parece muy simple: preguntar al cliente, a los usuarios y a otras personas cuáles son los objetivos para el sistema o producto, qué es lo que va a lograrse, cómo se ajusta el sistema o producto a las necesidades del negocio y, finalmente, cómo va a usarse el sistema producto en las operaciones cotidianas. Pero no es simple: es muy difícil.

Christel y Kang (Christel & Kang, 1992) identificaron un cierto número de problemas que se encuentran cuando ocurre la indagación.

- **Problemas de alcance:** La frontera de los sistemas está mal definida o los clientes o usuarios finales especifican detalles técnicos y necesario que confunden, más que clarifican, los objetivos generales del sistema.

[2] Si va a desarrollarse un sistema basado en computadora, los análisis comienzan en el contexto de un proceso de ingeniería de sistemas. Para más detalles de la ingeniería de sistemas, visite el sitio web de esta obra.
[3] Recuerde que el proceso unificado define una "fase de concepción" más amplia que incluye las fases de concepción, indagación y elaboración, que son estudiadas en dicho capítulo.

- **Problemas de entendimiento:** Los clientes o usuarios no están completamente seguro de lo que se necesita, comprenden mal las capacidades y limitaciones de su ambiente de computación, no entienden todo el dominio del problema, tienen problemas para comunicar las necesidades al ingeniero de software, omiten información que creen que es "obvia", especifican requerimientos que están en conflicto con las necesidades de otros clientes o usuarios, o solicitan requerimientos ambiguos o que no pueden someterse a prueba.
- **Problemas de volatilidad:** Los requerimientos cambian con el tiempo.

Para superar estos problemas, debe enfocarse la obtención de requerimientos en forma organizada.

Elaboración

La información obtenida del cliente durante la concepción e indagación se expande y refina durante la elaboración. esta tarea se centran desarrollaron modelo refinado de los requerimientos que identifique distintos aspectos de la función del software, su comportamiento e información.

La elaboración está motivada por la creación y mejora de escenarios de usuario que describan cómo interactuará el usuario final (y otros actores) con el sistema. Cada escenario de usuario se enuncia con sintaxis apropiada para extraer clases de análisis, que son entidades del dominio del negocio visibles para el usuario final.

Se definen los atributos de cada clase de análisis y se identifican los servicios[4] que requiere cada una de ellas. Se identifican las relaciones y colaboración entre clases, y se producen varios diagramas adicionales.

Negociación

No es raro que los clientes y usuarios pidan más de lo que puede lograrse dado lo limitado de los recursos del negocio. También es relativamente común que distintos clientes o usuarios propongan requerimientos conflictivos con el argumento de que su versión es "esencial para nuestras necesidades especiales".

[4] Un servicio manipula los datos agrupados por clase. También se utilizan los términos operación y método. Si no está familiarizado con conceptos de la orientación a objetos, consulte el apéndice 2, en el que se presenta una introducción básica.

Estos conflictos deben reconciliarse por medio de un proceso de negociación. Se pide a clientes, usuarios y otros participantes que ordenen sus requerimientos según su prioridad y que después analicen los conflictos. Con el empleo de un enfoque iterativo que da prioridad a los requerimientos, se evalúa su costo y riesgo, y se enfrentan los conflictos internos; algunos requerimientos se eliminan, se combinan o se modifican de modo que cada parte logre cierto grado de satisfacción.

Especificación

En el contexto de los sistemas basados en computadora (y software), el término especificación tiene diferentes significados para distintas personas. Una especificación puede ser un documento escrito, un conjunto de modelos gráficos, un modelo matemático formal, un conjunto de escenarios de uso, un prototipo o cualquier combinación de éstos.

Algunos sugieren que para una especificación debe desarrollarse y utilizarse una "plantilla estándar", con el argumento de que esto conduce a requerimientos presentados en forma consistente y por ello más comprensible.

Sin embargo, en ocasiones es necesario ser flexible cuando se desarrolla una especificación. Para sistemas grandes, el mejor enfoque puede ser un documento escrito que combine descripciones en un lenguaje natural con modelos gráficos. No obstante, para productos o sistemas pequeños que residan en ambientes bien entendidos, quizá todo lo que se requiera sea escenarios de uso.

Validación

La calidad de los productos del trabajo que se generan como consecuencia de la ingeniería de los requerimientos se evalúa durante el paso de validación. La validación de los requerimientos analiza la especificación[5] a fin de garantizar que todos ellos han sido enunciados sin ambigüedades; que se detectaron y corrigieron las inconsistencias, las omisiones y los errores, y que los productos del trabajo se presentan conforme a los estándares establecidos para el proceso, el proyecto y el producto.

[5] Recuerde que la naturaleza de la especificación variará con cada proyecto. En ciertos casos, la "especificación" no es más que un conjunto de escenarios de usuario. En otros, la especificación tal vez sea un documento que contiene escenarios, modelos y descripciones escritas.

El mecanismo principal de validación de los requerimientos es la revisión técnica. El equipo de revisión que los valida incluye ingenieros de software, clientes, usuarios y otros participantes, que analizan la especificación en busca de errores de contenido o de interpretación, de aspectos en los que tal vez se requiera hacer aclaraciones, falta de información, inconsistencias (problema notable cuando se hace la ingeniería de productos o sistemas grandes) y requerimientos en conflicto o irreales (no asequibles).

Administración de los requerimientos

Los requerimientos para sistemas basados en computadora cambian, y el deseo de modificarlos persiste durante toda la vida del sistema. La administración de los requerimientos es el conjunto de actividades que ayudan al equipo del proyecto a identificar, controlar y dar seguimiento a los requerimientos y a sus cambios en cualquier momento del desarrollo del proyecto[6].

2.4.2 Establecer las bases

En el caso ideal, los participantes e ingenieros de software trabajan juntos en el mismo equipo[7]. En esas condiciones, la ingeniería de requerimientos tan sólo consiste en sostener conversaciones significativas con colegas que sean miembros bien conocidos del equipo. Pero es frecuente que en la realidad esto sea muy diferente. Los clientes o usuarios finales tal vez se encuentren en ciudades o países diferentes, quizá sólo tengan una idea vaga de lo que se requiere, puede ser que tengan opiniones en conflicto sobre el sistema que se va a elaborar, que posean un conocimiento técnico limitado o que dispongan de poco tiempo para interactuar con el ingeniero que recabará los requerimientos. Ninguna de estas posibilidades es deseable, pero todas son muy comunes y es frecuente verse forzado a trabajar con las restricciones impuestas por esta situación. En las secciones que siguen se estudian las etapas requeridas para establecer las bases que permiten entender los requerimientos de software a fin de que el proyecto comience en forma tal que se mantenga avanzando hacia una solución exitosa.

[6] La administración formal de los requerimientos sólo se practica para proyectos grandes que tienen cientos de requerimientos identificables. Para proyectos pequeños, esta actividad tiene considerablemente menos formalidad.

[7] Este enfoque es ampliamente recomendable para proyectos que adoptan la filosofía de desarrollo de software ágil.

Identificación de los participantes

Sommerville y Sawyer (Somerville & Sawyer, 1997) definen participante como "cualquier persona que se beneficie en forma directa o indirecta del sistema en desarrollo". Ya se identificaron los candidatos habituales: gerentes de operaciones del negocio, gerentes de producto, personal de mercadotecnia, clientes internos y externos, usuarios finales, consultores, ingenieros de producto, ingenieros de software e ingenieros de apoyo y mantenimiento, entre otros.

Cada participante tiene un punto de vista diferente respecto del sistema, obtienen distintos beneficios cuando éste se desarrolla con éxito y corre distintos riesgos si fracasa el esfuerzo de construcción. Durante la concepción, debe hacerse la lista de personas que harán aportes cuando se recaben los requerimientos. La lista inicial crecerá cuando se haga contacto con los participantes porque a cada uno se le hará la pregunta: "¿A quién más piensa que debe consultarse?".

Reconocer los múltiples puntos de vista

Debido a que existen muchos participantes distintos, los requerimientos del sistema se explorarán desde muchos puntos de vista diferentes. Por ejemplo, el grupo de mercadotecnia se interesa en funciones y características que estimularán el mercado potencial, lo que hará que el nuevo sistema sea fácil de vender. Los gerentes del negocio tienen interés en un conjunto de características para que se elabore dentro del presupuesto y que esté listo para ocupar nichos de mercado definidos. Los usuarios finales tal vez quieran características que les resulten familiares y que sean fáciles de aprender y usar. Los ingenieros de software quizá piensen en funciones invisibles para los participantes sin formación técnica, pero que permitan una infraestructura que dé apoyo a funciones y características más vendibles. Los ingenieros de apoyo tal vez se centren en la facilidad del software para recibir mantenimiento.

Cada uno de estos integrantes (y otros más) aportará información al proceso de ingeniería de los requerimientos. A medida que se recaba información procedente de múltiples puntos de vista, los requerimientos que surjan tal vez sean inconsistentes o estén en conflicto uno con otro. Debe clasificarse toda la información de los participantes (incluso los requerimientos inconsistentes y conflictivos) en forma que permita a quienes toman las decisiones escoger para el sistema un conjunto de requerimientos que tenga coherencia interna.

Trabajar hacia la colaboración

Si en un proyecto de software hay involucrados cinco participantes, tal vez se tengan cinco (o más) diferentes opiniones acerca del conjunto apropiado de requerimientos. En los primeros capítulos se mencionó que, para obtener un sistema exitoso, los clientes (y otros participantes) debían colaborar entre sí (sin pelear por insignificancias) y con los profesionales de la ingeniería de software. Pero, ¿cómo se llega a esta colaboración?

El trabajo del ingeniero de requerimientos es identificar las áreas de interés común (por ejemplo, requerimientos en los que todos los participantes estén de acuerdo) y las de conflicto o incongruencia (por ejemplo, requerimientos que desea un participante, pero que están en conflicto con las necesidades de otro). Es la última categoría la que, por supuesto, representa un reto.

La colaboración no significa necesariamente que todos los requerimientos los defina un comité. En muchos casos, los participantes colaboran con la aportación de su punto de vista respecto de los requerimientos, pero un influyente "campeón del proyecto" (por ejemplo, el director del negocio o un tecnólogo experimentado) toma la decisión final sobre los requerimientos que lo integrarán.

Hacer las primeras preguntas

Las preguntas que se hacen en la concepción del proyecto deben estar "libres del contexto" (Gause & Weinberg, 1989). El primer conjunto de ellas se centran en el cliente y en otros participantes, en las metas y beneficios generales. Por ejemplo, tal vez se pregunte:

- ¿Quién está detrás de la solicitud de este trabajo?
- ¿Quién usará la solución?
- ¿Cuál será el beneficio económico de una solución exitosa?
- ¿Hay otro origen para la solución que se necesita?

Estas preguntas ayudan a identificar a todos los participantes con interés en el software que se va a elaborar. Además, las preguntas identifican el beneficio mensurable de una implementación exitosa y las posibles alternativas para el desarrollo de software

personalizado. Las preguntas siguientes permiten entender mejor el problema y hacen que el cliente exprese sus percepciones respecto de la solución:

- ¿Cuál sería una "buena" salida generada por una solución exitosa?
- ¿Qué problemas resolvería esta solución?
- ¿Puede mostrar (o describir) el ambiente de negocios en el que se usaría la solución?
- ¿Hay aspectos especiales del desempeño o restricciones que afecten el modo en el que se enfoque la solución?

Las preguntas finales se centran en la eficacia de la actividad de comunicación en sí. Gause y Weinberg (Gause & Weinberg, 1989) las llaman "metapreguntas" y proponen la siguiente lista (abreviada):

- ¿Es usted la persona indicada para responder estas preguntas? ¿Sus respuestas son "oficiales"?
- ¿Mis preguntas son relevantes para el problema que se tiene?
- ¿Puede otra persona dar información adicional?
- ¿Debería yo preguntarle algo más?

Estas preguntas (y otras) ayudarán a "romper el hielo" y a iniciar la comunicación, que es esencial para una indagación exitosa. Pero una reunión de preguntas y respuestas no es un enfoque que haya tenido un éxito apabullante. En realidad, la sesión de preguntas y respuestas sólo debe usarse para el primer encuentro y luego ser reemplazada por un formato de indagación de requerimientos que combine elementos de solución de problemas, negociación y especificación.

2.4.3 Indagación de los requerimientos

La indagación de los requerimientos (actividad también llamada recabación de los requerimientos) combina elementos de la solución de problemas, elaboración, negociación y especificación. A fin de estimular un enfoque colaborativo y orientado al equipo, los participantes trabajan juntos para identificar el problema, proponer elementos de la solución, negociar distintas visiones y especificar un conjunto preliminar de requerimientos para la solución (Zahniser, 1990)[8].

[8] En ocasiones se denomina a este enfoque técnica facilitada de especificación de la aplicación (TFEA).

Recabación de los requerimientos en forma colaborativa

Se han propuesto muchos enfoques distintos para recabar los requerimientos en forma colaborativa. Cada uno utiliza un escenario un poco diferente, pero todos son variantes de los siguientes lineamientos básicos:

- Tanto ingenieros de software como otros participantes dirigen o intervienen en las reuniones.
- Se establecen reglas para la preparación y participación.
- Se sugiere una agenda con suficiente formalidad para cubrir todos los puntos importantes, pero con la suficiente informalidad para que estimule el libre flujo de ideas.
- Un "facilitador" (cliente, desarrollador o participante externo) controla la reunión.
- Se utiliza un "mecanismo de definición" (que pueden ser hojas de trabajo, tablas sueltas, etiquetas adhesivas, pizarrón electrónico, grupos de conversación o foro virtual).

La meta es identificar el problema, proponer elementos de la solución, negociar distintos enfoques y especificar un conjunto preliminar de requerimientos de la solución en una atmósfera que favorezca el logro de la meta. Para entender mejor el flujo de eventos conforme ocurren, se presenta un escenario breve que bosqueja la secuencia de hechos que llevan a la reunión para obtener requerimientos, a lo que sucede durante ésta y a lo que sigue después de ella.

Durante la concepción, hay preguntas y respuestas básicas que establecen el alcance del problema y la percepción general de lo que constituye una solución. Fuera de estas reuniones iniciales, el desarrollador y los clientes escriben una o dos páginas de "solicitud de producto". Se selecciona un lugar, fecha y hora para la reunión, se escoge un facilitador y se invita a asistir a integrantes del equipo de software y de otras organizaciones participantes. Antes de la fecha de la reunión, se distribuye la solicitud de producto a todos los asistentes.

Por ejemplo[9], considere un extracto de una solicitud de producto escrita por una persona de mercadotecnia involucrada en el proyecto **CasaSegura**. Esta persona escribe la siguiente narración sobre la función de seguridad en el hogar que va a ser parte de **CasaSegura**:

Nuestras investigaciones indican que el mercado para los sistemas de administración del hogar crece a razón de 40% anual. La primera función de CasaSegura que llevemos al mercado deberá ser la de seguridad del hogar. La mayoría de la gente está familiarizada con "sistemas de alarma", por lo que ésta deberá ser fácil de vender. La función de seguridad del hogar protegería, o reconocería, varias "situaciones" indeseables, como acceso ilegal, incendio y niveles de monóxido de carbono, entre otros. Emplearía sensores inalámbricos para detectar cada situación. Sería programada por el propietario y telefonearía en forma automática a una agencia de vigilancia cuando detectara una situación como las descritas.

En realidad, durante la reunión para recabar los requerimientos, otros contribuirían a esta narración y se dispondría de mucha más información. Pero aun con ésta habría ambigüedad, sería probable que existieran omisiones y ocurrieran errores. Por ahora bastará la "descripción funcional" anterior. Mientras se revisa la solicitud del producto antes de la reunión, se pide a cada asistente que elabore una lista de objetos que sean parte del ambiente que rodeará al sistema, los objetos que producirá éste y los que usará para realizar sus funciones. Además, se solicita a cada asistente que haga otra lista de servicios (procesos o funciones) que manipulen o interactúen con los objetos.

Por último, también se desarrollan listas de restricciones (por ejemplo, costo, tamaño, reglas del negocio, etc.) y criterios de desempeño (como velocidad y exactitud). Se informa a los asistentes que no se espera que las listas sean exhaustivas, pero sí que reflejen la percepción que cada persona tiene del sistema.

Entre los objetos descritos por **CasaSegura** tal vez estén incluidos el panel de control, detectores de humo, sensores en ventanas y puertas, detectores de movimiento, alarma, un evento (activación de un sensor), una pantalla, una computadora, números telefónicos, una llamada telefónica, etc. La lista de servicios puede incluir configurar el sistema, preparar la alarma, vigilar los sensores, marcar el teléfono, programar el panel de control y leer la pantalla (observe que los servicios actúan sobre los objetos). En forma similar, cada asistente

[9] Este ejemplo (con extensiones y variantes) se usa para ilustrar métodos importantes de la ingeniería de software. Como ejercicio, sería provechoso que el lector realizara su propia reunión para recabar requerimientos y que desarrollara un conjunto de listas para ella.

desarrollará una lista de restricciones (por ejemplo, el sistema debe reconocer cuando los sensores no estén operando, debe ser amistoso con el usuario, debe tener una interfaz directa con una línea telefónica estándar, etc.) y de criterios de desempeño (un evento en un sensor debe reconocerse antes de un segundo, debe implementarse un esquema de prioridad de eventos, etcétera).

Las listas de objetos pueden adherirse a las paredes del cuarto con el empleo de pliegos de papel grandes o con láminas adhesivas, o escribirse en un tablero. Alternativamente, las listas podrían plasmarse en un boletín electrónico, sitio web interno o en un ambiente de grupo de conversación para revisarlas antes de la reunión. Lo ideal es que cada entrada de las listas pueda manipularse por separado a fin de combinar las listas o modificar las entradas y agregar otras. En esta etapa, están estrictamente prohibidos las críticas y el debate.

Una vez que se presentan las listas individuales acerca de un área temática, el grupo crea una lista, eliminando las entradas redundantes o agregando ideas nuevas que surjan durante el análisis, pero no se elimina ninguna. Después de crear listas combinadas para todas las áreas temáticas, sigue el análisis, coordinado por el facilitador. La lista combinada se acorta, se alarga o se modifica su redacción para que refleje de manera apropiada al producto o sistema que se va a desarrollar. El objetivo es llegar a un consenso sobre la lista de objetos, servicios, restricciones y desempeño del sistema que se va a construir.

En muchos casos, un objeto o servicio descrito en la lista requerirá mayores explicaciones. Para lograr esto, los participantes desarrollan miniespecificaciones para las entradas en las listas[10]. Cada miniespecificación es una elaboración de un objeto o servicio. Por ejemplo, la correspondiente al objeto Panel de control de **CasaSegura** sería así:

El panel de control es una unidad montada en un muro, sus dimensiones aproximadas son de 9 por 5 pulgadas. Tiene conectividad inalámbrica con los sensores y con una PC. La interacción con el usuario tiene lugar por medio de un tablero que contiene 12 teclas. Una pantalla de cristal líquido de 3 por 3 pulgadas brinda retroalimentación al usuario. El software hace anuncios interactivos, como eco y funciones similares.

[10] En vez de crear una miniespecificación, muchos equipos de software eligen desarrollar escenarios del usuario llamados casos de uso.

Las miniespecificaciones se presentan a todos los participantes para que sean analizadas. Se hacen adiciones, eliminaciones y otras modificaciones. En ciertos casos, el desarrollo de las miniespecificaciones descubrirá nuevos objetos, servicios o restricciones, o requerimientos de desempeño que se agregarán a las listas originales. Durante todos los análisis, el equipo debe posponer los aspectos que no puedan resolverse en la reunión. Se conserva una lista de aspectos para volver después a dichas ideas.

CasaSegura

Conducción de una reunión para recabar los requerimientos

La escena: Sala de juntas. Está en marcha la primera reunión para recabar los requerimientos.

Participantes: Jamie Lazar, integrante del equipo de software; Vinod Raman, miembro del equipo de software; Ed Robbins, miembro del equipo de software; Doug Miller, gerente de ingeniería de software; tres trabajadores de mercadotecnia; un representante de ingeniería del producto, y un facilitador.

La conversación:

Facilitador (apunta en un pizarrón): De modo que ésa es la lista actual de objetos y servicios para la función de seguridad del hogar.

Persona de mercadotecnia: Eso la cubre, desde nuestro punto de vista.

Vinod: ¿No dijo alguien que quería que toda la funcionalidad de *CasaSegura* fuera accesible desde internet? Eso incluiría la función de seguridad, ¿o no?

Persona de mercadotecnia: Sí, así es... tendremos que añadir esa funcionalidad y los objetos apropiados.

Facilitador: ¿Agrega eso algunas restricciones?

Jamie: Sí, tanto técnicas como legales.

Representante del producto: ¿Qué significa eso?

Jamie: Nos tendríamos que asegurar de que un extraño no pueda ingresar al sistema, desactivarlo y robar en el lugar o hacer algo peor. Mucha responsabilidad sobre nosotros.

Doug: Muy cierto.

Mercadotecnia: Pero lo necesitamos así... sólo asegúrense de impedir que ingrese un extraño.

Ed: Eso es más fácil de decir que de hacer.

Facilitador (interrumpe): No quiero que debatamos esto ahora. Anotémoslo como un aspecto y continuemos.

(Doug, que es el secretario de la reunión, toma debida nota.)

Facilitador: Tengo la sensación de que hay más por considerar aquí.

(El grupo dedica los siguientes 20 minutos a mejorar y aumentar los detalles de la función de seguridad del hogar.)

Figura N°2-04 - Conducción de una reunión para recabar los requerimientos

Despliegue de la función de calidad

El despliegue de la función de calidad (DFC) es una técnica de administración de la calidad que traduce las necesidades del cliente en requerimientos técnicos para el software. El DFC "se concentra en maximizar la satisfacción del cliente a partir del proceso de ingeniería del software". Para lograr esto, el DFC pone el énfasis en entender lo que resulta valioso para el cliente y luego despliega dichos valores en todo el proceso de ingeniería. El DFC identifica tres tipos de requerimientos:

- **Requerimientos normales:** Objetivos y metas que se establecen para un producto o sistema durante las reuniones con el cliente. Si estos requerimientos están presentes, el cliente queda satisfecho. Ejemplos de requerimientos normales son los tipos de

gráficos pedidos para aparecer en la pantalla, funciones específicas del sistema y niveles de rendimiento definidos.

- **Requerimientos esperados:** Están implícitos en el producto o sistema y quizá sean tan importantes que el cliente no los mencione de manera explícita. Su ausencia causará mucha insatisfacción. Algunos ejemplos de requerimientos esperados son: fácil interacción humano/máquina, operación general correcta y confiable, y facilidad para instalar el software.
- **Requerimientos emocionantes:** Estas características van más allá de las expectativas del cliente y son muy satisfactorias si están presentes. Por ejemplo, el software para un nuevo teléfono móvil viene con características estándar, pero si incluye capacidades inesperadas (como pantalla sensible al tacto, correo de voz visual, etc.) agrada a todos los usuarios del producto.

Aunque los conceptos del DFC son aplicables en todo el proceso del software, hay técnicas específicas de aquél que pueden aplicarse a la actividad de indagación de los requerimientos. El DFC utiliza entrevistas con los clientes, observación, encuestas y estudio de datos históricos (por ejemplo, reportes de problemas) como materia prima para la actividad de recabación de los requerimientos. Después, estos datos se llevan a una tabla de requerimientos -llamada tabla de la voz del cliente- que se revisa con el cliente y con otros participantes. Luego se emplean varios diagramas, matrices y métodos de evaluación para extraer los requerimientos esperados y tratar de percibir requerimientos emocionantes.

Escenarios de uso

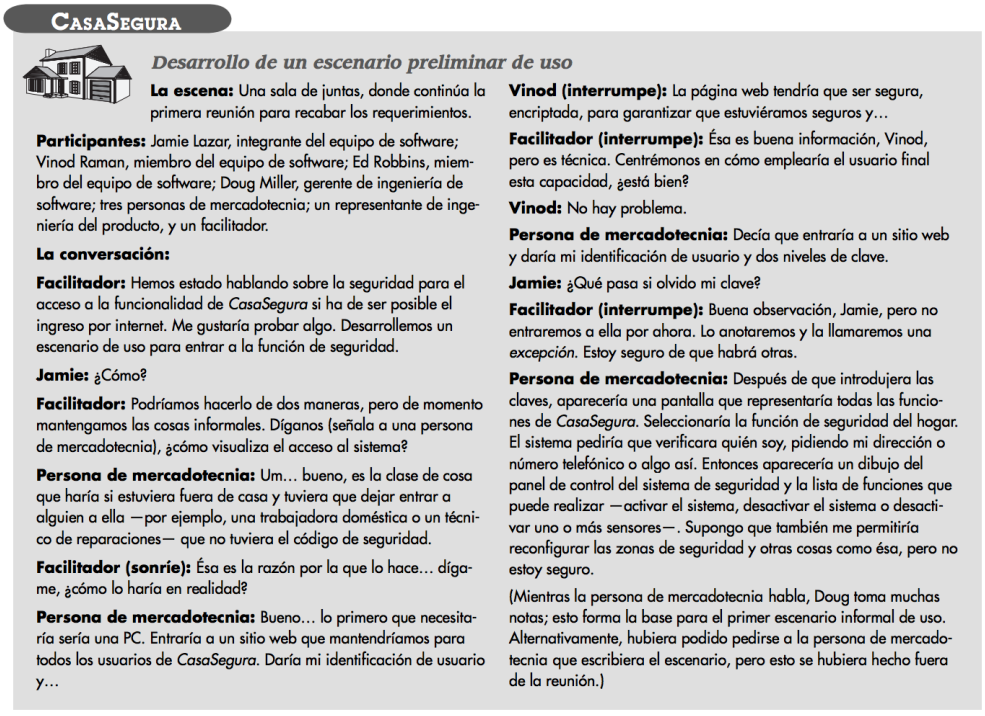
A medida que se reúnen los requerimientos, comienza a materializarse la visión general de funciones y características del sistema. Sin embargo, es difícil avanzar hacia actividades más técnicas de la ingeniería de software hasta no entender cómo emplearán los usuarios finales dichas funciones y características. Para lograr esto, los desarrolladores y usuarios crean un conjunto de escenarios que identifican la naturaleza de los usos para el sistema que se va a construir. Los escenarios, que a menudo se llaman casos de uso, proporcionan la descripción de la manera en la que se utilizará el sistema.

Indagación de los productos del trabajo

Los productos del trabajo generados como consecuencia de la indagación de los requerimientos variarán en función del tamaño del sistema o producto que se va a construir. Para la mayoría de sistemas, los productos del trabajo incluyen los siguientes:

- Un enunciado de la necesidad y su factibilidad.
- Un enunciado acotado del alcance del sistema o producto.
- Una lista de clientes, usuarios y otros participantes que intervienen en la indagación de los requerimientos.
- Una descripción del ambiente técnico del sistema.
- Una lista de requerimientos (de preferencia organizados por función) y las restricciones del dominio que se aplican a cada uno.
- Un conjunto de escenarios de uso que dan perspectiva al uso del sistema o producto en diferentes condiciones de operación.
- Cualesquiera prototipos desarrollados para definir requerimientos.

Cada uno de estos productos del trabajo es revisado por todas las personas que participan en la indagación de los requerimientos.



CASASEGURA

Desarrollo de un escenario preliminar de uso

La escena: Una sala de juntas, donde continúa la primera reunión para recabar los requerimientos.

Participantes: Jamie Lazar, integrante del equipo de software; Vinod Raman, miembro del equipo de software; Ed Robbins, miembro del equipo de software; Doug Miller, gerente de ingeniería de software; tres personas de mercadotecnia; un representante de ingeniería del producto, y un facilitador.

La conversación:

Facilitador: Hemos estado hablando sobre la seguridad para el acceso a la funcionalidad de *CasaSegura* si ha de ser posible el ingreso por internet. Me gustaría probar algo. Desarrollemos un escenario de uso para entrar a la función de seguridad.

Jamie: ¿Cómo?

Facilitador: Podríamos hacerlo de dos maneras, pero de momento mantengamos las cosas informales. Díganos (señala a una persona de mercadotecnia), ¿cómo visualiza el acceso al sistema?

Persona de mercadotecnia: Um… bueno, es la clase de cosa que haría si estuviera fuera de casa y tuviera que dejar entrar a alguien a ella —por ejemplo, una trabajadora doméstica o un técnico de reparaciones— que no tuviera el código de seguridad.

Facilitador (sonríe): Ésa es la razón por la que lo hace… dígame, ¿cómo lo haría en realidad?

Persona de mercadotecnia: Bueno… lo primero que necesitaría sería una PC. Entraría a un sitio web que mantendríamos para todos los usuarios de *CasaSegura*. Daría mi identificación de usuario y…

Vinod (interrumpe): La página web tendría que ser segura, encriptada, para garantizar que estuviéramos seguros y…

Facilitador (interrumpe): Ésa es buena información, Vinod, pero es técnica. Centrémonos en cómo emplearía el usuario final esta capacidad, ¿está bien?

Vinod: No hay problema.

Persona de mercadotecnia: Decía que entraría a un sitio web y daría mi identificación de usuario y dos niveles de clave.

Jamie: ¿Qué pasa si olvido mi clave?

Facilitador (interrumpe): Buena observación, Jamie, pero no entraremos a ella por ahora. Lo anotaremos y la llamaremos una *excepción*. Estoy seguro de que habrá otras.

Persona de mercadotecnia: Después de que introdujera las claves, aparecería una pantalla que representaría todas las funciones de *CasaSegura*. Seleccionaría la función de seguridad del hogar. El sistema pediría que verificara quién soy, pidiendo mi dirección o número telefónico o algo así. Entonces aparecería un dibujo del panel de control del sistema de seguridad y la lista de funciones que puede realizar —activar el sistema, desactivar el sistema o desactivar uno o más sensores—. Supongo que también me permitiría reconfigurar las zonas de seguridad y otras cosas como ésa, pero no estoy seguro.

(Mientras la persona de mercadotecnia habla, Doug toma muchas notas; esto forma la base para el primer escenario informal de uso. Alternativamente, hubiera podido pedirse a la persona de mercadotecnia que escribiera el escenario, pero esto se hubiera hecho fuera de la reunión.)

Figura N°2-05 - Desarrollo de un escenario preliminar de uso

2.4.4 Desarrollo de casos de uso

En un libro que analiza cómo escribir casos de uso eficaces, Alistair Cockburn (Cockburn, 2001) afirma que "un caso de uso capta un contrato que describe el comportamiento del sistema en distintas condiciones en las que el sistema responde a una petición de alguno de sus participantes". En esencia, un caso de uso narra una historia estilizada sobre cómo interactúa un usuario final (que tiene cierto número de roles posibles) con el sistema en circunstancias específicas. La historia puede ser un texto narrativo, un lineamiento de tareas o interacciones, una descripción basada en un formato o una representación diagramática. Sin importar su forma, un caso de uso ilustra el software o sistema desde el punto de vista del usuario final.

El primer paso para escribir un caso de uso es definir un conjunto de "actores" que estarán involucrados en la historia. Los actores son las distintas personas (o dispositivos) que usan el sistema o producto en el contexto de la función y comportamiento que va a describirse. Los actores representan los papeles que desempeñan las personas (o dispositivos) cuando opera el sistema. Con una definición más formal, un actor es cualquier cosa que se comunique con el sistema o producto y que sea externo a éste. Todo actor tiene uno o más objetivos cuando utiliza el sistema.

Es importante notar que un actor y un usuario final no necesariamente son lo mismo. Un usuario normal puede tener varios papeles diferentes cuando usa el sistema, mientras que un actor representa una clase de entidades externas (gente, con frecuencia pero no siempre) que sólo tiene un papel en el contexto del caso de uso. Por ejemplo, considere al operador de una máquina (un usuario) que interactúa con la computadora de control de una celda de manufactura que contiene varios robots y máquinas de control numérico. Después de una revisión cuidadosa de los requerimientos, el software para la computadora de control requiere cuatro diferentes modos (papeles) para la interacción: modo de programación, modo de prueba, modo de vigilancia y modo de solución de problemas. Por tanto, es posible definir cuatro actores: programador, probador, vigilante y solucionador de problemas. En ciertos casos, el operador de la máquina desempeñará todos los papeles. En otros, distintas personas tendrán el papel de cada actor.

Debido a que la indagación de los requerimientos es una actividad evolutiva, no todos los actores son identificados en la primera iteración. En ésta es posible identificar a los actores

principales, y a los secundarios cuando se sabe más del sistema. Los actores principales interactúan para lograr la función requerida del sistema y obtienen el beneficio previsto de éste. Trabajan con el software en forma directa y con frecuencia. Los actores secundarios dan apoyo al sistema, de modo que los primarios puedan hacer su trabajo.

Una vez identificados los actores, es posible desarrollar casos de uso. Jacobson sugiere varias preguntas[11] que debe responder un caso de uso:

- ¿Quién es el actor principal y quién(es) el(los) secundario(s)?
- ¿Cuáles son los objetivos de los actores?
- ¿Qué precondiciones deben existir antes de comenzar la historia?
- ¿Qué tareas o funciones principales son realizadas por el actor?
- ¿Qué excepciones deben considerarse al describir la historia?
- ¿Cuáles variaciones son posibles en la interacción del actor?
- ¿Qué información del sistema adquiere, produce o cambia el actor?
- ¿Tendrá que informar el actor al sistema acerca de cambios en el ambiente externo?
- ¿Qué información desea obtener el actor del sistema?
- ¿Quiere el actor ser informado sobre cambios inesperados?

En relación con los requerimientos básicos de **CasaSegura**, se definen cuatro actores: propietario de la casa (usuario), gerente de arranque (tal vez la misma persona que el propietario de la casa, pero en un papel diferente), sensores (dispositivos adjuntos al sistema) y subsistema de vigilancia y respuesta (estación central que vigila la función de seguridad de la casa de **CasaSegura**). Para fines de este ejemplo, consideraremos sólo al actor llamado propietario de la casa. Éste interactúa con la función de seguridad de la casa en varias formas distintas con el empleo del panel de control de la alarma o con una PC:

- Introduce una clave que permita todas las demás interacciones.
- Pregunta sobre el estado de una zona de seguridad.
- Interroga acerca del estado de un sensor.
- En una emergencia, oprime el botón de pánico.
- Activa o desactiva el sistema de seguridad.

[11] Las preguntas de Jacobson se han ampliado para que den una visión más completa del contenido del caso de uso.

Considerando la situación en la que el propietario de la casa usa el panel de control, a continuación se plantea el caso de uso básico para la activación del sistema[12]:

1. El propietario observa el panel de control de CasaSegura (ver la figura 2-06) para determinar si el sistema está listo para recibir una entrada. Si el sistema no está listo, se muestra el mensaje no está listo en la pantalla de cristal líquido y el propietario debe cerrar físicamente ventanas o puertas de modo que desaparezca dicho mensaje (el mensaje no está listo implica que un sensor está abierto; por ejemplo, que una puerta o ventana está abierta).
2. El propietario usa el teclado para introducir una clave de cuatro dígitos. La clave se compara con la que guarda el sistema como válida. Si la clave es incorrecta, el panel de control emitirá un sonido una vez y se reiniciará para recibir una entrada adicional. Si la clave es correcta, el panel de control espera otras acciones.
3. El propietario selecciona y teclea permanecer o fuera (ver la figura 2-06) para activar el sistema. La entrada permanecer activa sólo sensores perimetrales (se desactivan los sensores de detección de movimiento interior). La entrada fuera activa todos los sensores.
4. Cuando ocurre una activación, el propietario observa una luz roja de alarma.

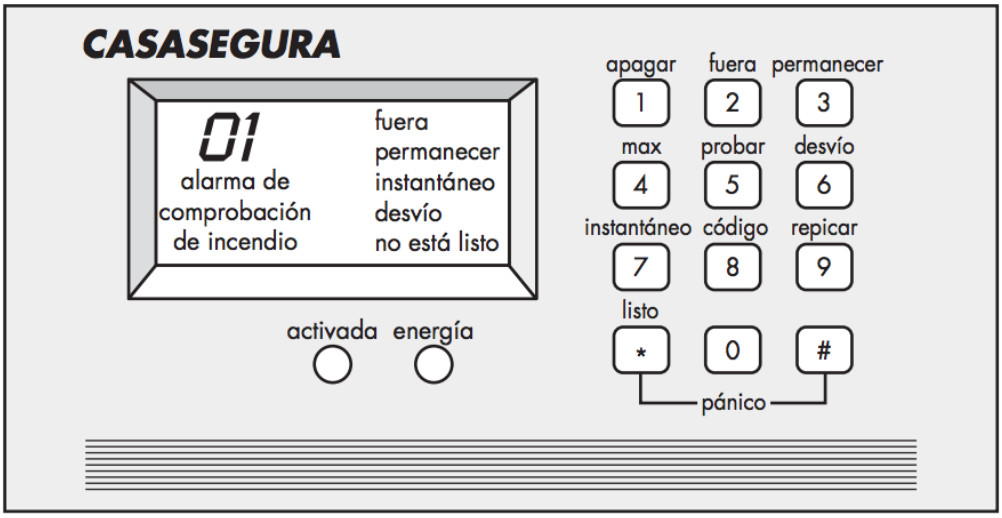


Figura Nº2-06 - Panel de control de CasaSegura

[12] Observe que este caso de uso difiere de la situación en la que se accede al sistema a través de internet. En este caso, la interacción es por medio del panel de control y no con la interfaz de usuario gráfica (GUI) que se da cuando se emplea una PC.

El caso de uso básico presenta una historia de alto nivel que describe la interacción entre el actor y el sistema. En muchas circunstancias, los casos de uso son más elaborados a fin de que brinden muchos más detalles sobre la interacción. Por ejemplo, Cockburn (Cockburn, 2001) sugiere el siguiente formato para hacer descripciones detalladas de casos de uso:

Caso de uso:	IniciarVigilancia
Actor principal:	Propietario
Objetivo en contexto:	Preparar el sistema para que vigile los sensores cuando el propietario salga de la casa o permanezca dentro.
Precondiciones:	El sistema se ha programado para recibir una clave y reconocer distintos sensores.
Disparador:	El propietario decide "preparar" el sistema, por ejemplo, para que encienda las funciones de alarma.

Escenario:

1. Propietario: observa el panel de control
2. Propietario: introduce una clave
3. Propietario: selecciona "permanecer" o "fuera"
4. Propietario: observa una luz roja de alarma que indica que CasaSegura ha sido activada.

Excepciones:

1. El panel de control no está listo: el propietario verifica todos los sensores para determinar cuáles están abiertos; los cierra.
2. La clave es incorrecta (el panel de control suena una vez): el propietario introduce la clave correcta.
3. La clave no es reconocida: debe contactarse el subsistema de vigilancia y respuesta para reprogramar la clave.
4. Se elige permanecer: el panel de control suena dos veces y se enciende un letrero luminoso que dice permanecer; se activan los sensores del perímetro.
5. Se selecciona fuera: el panel de control suena tres veces y se enciende un letrero luminoso que dice fuera; se activan todos los sensores.

Prioridad:	Esencial, debe implementarse
Cuándo estará disponible:	En el primer incremento
Frecuencia de uso:	Muchas veces por día
Canal para el actor:	A través de la interfaz del panel de control
Actores secundarios:	Técnico de apoyo, sensores

Canales para los actores secundarios:

Técnico de apoyo: línea telefónica

Sensores: interfaces cableadas y frecuencia de radio

Aspectos pendientes:

1. ¿Debe haber una forma de activar el sistema sin usar clave o con una clave abreviada?
2. ¿El panel de control debe mostrar mensajes de texto adicionales?
3. ¿De cuánto tiempo dispone el propietario para introducir la clave a partir del momento en el que se oprime la primera tecla?
4. ¿Hay una forma de desactivar el sistema antes de que se active en realidad?

Los casos de uso para otras interacciones de propietario se desarrollarían en una forma similar. Es importante revisar con cuidado cada caso de uso. Si algún elemento de la interacción es ambiguo, es probable que la revisión del caso de uso lo detecte.

CasaSegura

Desarrollo de un diagrama de caso de uso de alto nivel

La escena: Sala de juntas, continúa la reunión para recabar los requerimientos.

Participantes: Jamie Lazar, miembro del equipo de software; Vinod Roman, integrante del equipo de software; Ed Robbins, integrante del equipo de software; Doug Miller, gerente de ingeniería de software; tres miembros de mercadotecnia; un representante de ingeniería del producto; un facilitador.

La conversación:

Facilitador: Hemos pasado un buen tiempo hablando de la función de seguridad del hogar de *CasaSegura*. Durante el receso hice un diagrama de caso de uso para resumir los escenarios importantes que forman parte de esta función. Veámoslo.

(Todos los asistentes observan la figura 5.2.)

Jamie: Estoy aprendiendo la notación UML.[14] Veo que la función de seguridad del hogar está representada por el rectángulo grande con óvalos en su interior, ¿verdad? ¿Y los óvalos representan los casos de uso que hemos escrito?

Facilitador: Sí. Y las figuras pegadas representan a los actores —personas o cosas que interactúan con el sistema según los describe el caso de uso…—; ¡ah! usé el cuadrado para representar un actor que no es persona… en este caso, sensores.

Doug: ¿Es válido eso en UML?

Facilitador: La legalidad no es lo importante. El objetivo es comunicar información. Veo que usar una figura humana para representar un equipo sería erróneo. Así que adapté las cosas un poco. No pienso que genere problemas.

Vinod: Está bien, entonces tenemos narraciones de casos de uso para cada óvalo. ¿Necesitamos desarrollarlas con base en los formatos sobre los que he leído?

Facilitador: Es probable, pero eso puede esperar hasta que hayamos considerado otras funciones de *CasaSegura*.

Persona de mercadotecnia: Esperen, he estado observando este diagrama y de pronto me doy cuenta de que hemos olvidado algo.

Facilitador: ¿De verdad? Dime, ¿qué hemos olvidado?

(La reunión continúa.)

Figura Nº2-07 - Desarrollo de un diagrama de caso de uso de alto nivel

2.4.5 Elaboración del modelo de los requerimientos

Hay muchas formas diferentes de concebir los requerimientos para un sistema basado en computadora. Algunos profesionales del software afirman que es mejor seleccionar un modo de representación (por ejemplo, el caso de uso) y aplicarlo hasta excluir a todos los demás. Otros piensan que es más benéfico usar cierto número de modos de representación distintos para ilustrar el modelo de requerimientos. Los modos diferentes de representación fuerzan a considerar los requerimientos desde distintos puntos de vista, enfoque que tiene una probabilidad mayor de detectar omisiones, inconsistencia y ambigüedades.

Los elementos específicos del modelo de requerimientos están determinados por el método de análisis de modelado que se use.

Elementos basados en el escenario.

El sistema se describe desde el punto de vista del usuario con el empleo de un enfoque basado en el escenario. Por ejemplo, los casos de uso básico y sus diagramas correspondientes de casos de uso evolucionan hacia otros más elaborados que se basan en formatos. Los elementos del modelo de requerimientos basados en el escenario con frecuencia son la primera parte del modelo en desarrollo. Como tales, sirven como entrada para la creación de otros elementos de modelado. La figura 2-08 ilustra un diagrama de actividades UML[13] para indagar los requerimientos y representarlos con el empleo de casos de uso. Se aprecian tres niveles de elaboración que culminan en una representación basada en el escenario.

Elementos basados en clases: Cada escenario de uso implica un conjunto de objetos que se manipulan cuando un actor interactúa con el sistema. Estos objetos se clasifican en clases: conjunto de objetos que tienen atributos similares y comportamientos comunes. Por ejemplo, para ilustrar la clase Sensor de la función de seguridad de Casa Segura (ver la figura 2-09), puede utilizarse un diagrama de clase UML. Observe que el diagrama enlista los atributos de los sensores (por ejemplo, nombre, tipo, etc.) y las operaciones (por ejemplo, identificar y permitir) que se aplican para modificarlos. Además de los diagramas de clase, otros elementos de modelado del análisis ilustran la manera en la que las clases colaboran una con otra y las relaciones e interacciones entre ellas.

[13] El Lenguaje de Modelamiento Unificado (UML - Unified Modeling Language) es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software.

Elementos de comportamiento: El comportamiento de un sistema basado en computadora tiene un efecto profundo en el diseño que se elija y en el enfoque de implementación que se aplique. Por tanto, el modelo de requerimientos debe proveer elementos de modelado que ilustren el comportamiento. El diagrama de estado es un método de representación del comportamiento de un sistema que ilustra sus estados y los eventos que ocasionan que el sistema cambie de estado. Un estado es cualquier modo de comportamiento observable desde el exterior. Además, el diagrama de estado indica acciones (como la activación de un proceso, por ejemplo) tomadas como consecuencia de un evento en particular. Para ilustrar el uso de un diagrama de estado, considere el software incrustado dentro del panel de control de CasaSegura que es responsable de leer las entradas que hace el usuario. En la figura 2-10 se presenta un diagrama de estado UML simplificado. Además de las representaciones de comportamiento del sistema como un todo, también es posible modelar clases individuales.

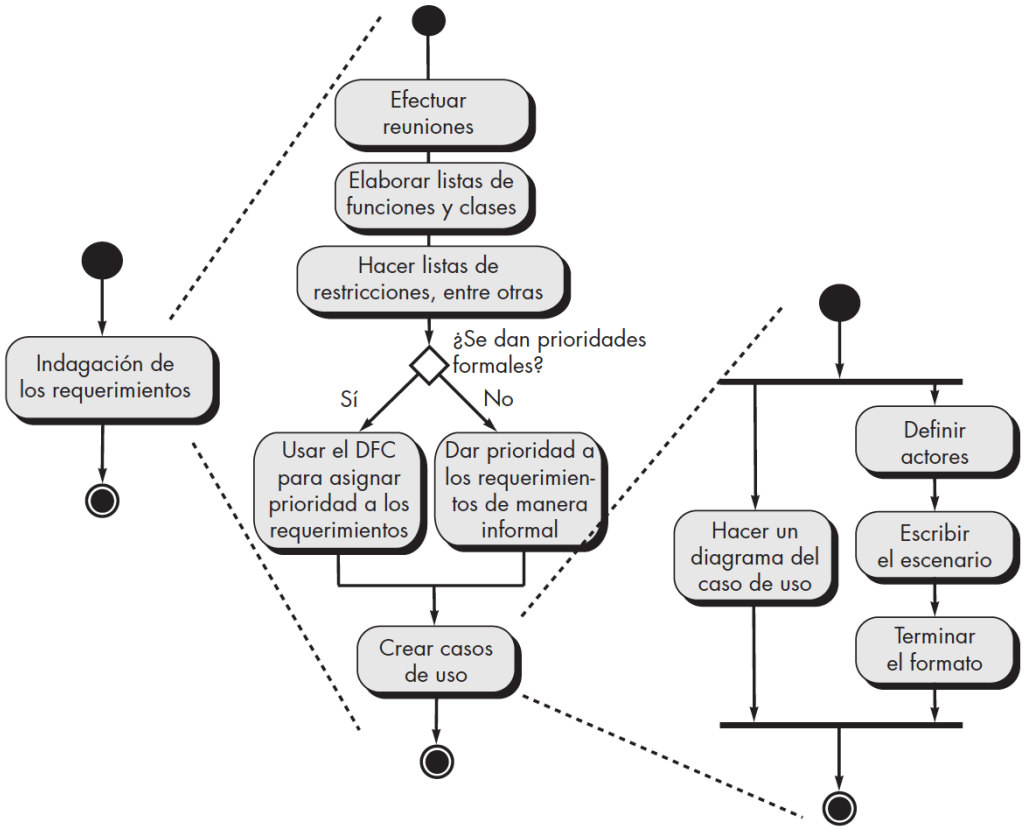


Figura N°2-08 - Diagrama de actividades del UML para indagar los requerimientos

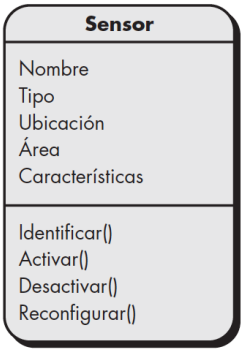


Figura Nº2-09 - Diagrama de clase para un sensor

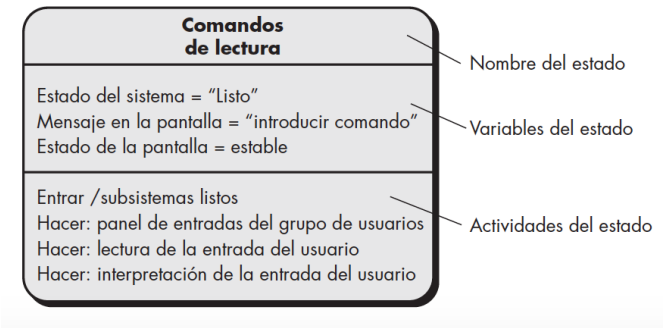


Figura Nº2-10 - Notación UML del diagrama de estado

Elementos orientados al flujo: La información se transforma cuando fluye a través de un sistema basado en computadora. El sistema acepta entradas en varias formas, aplica funciones para transformarla y produce salidas en distintos modos. La entrada puede ser una señal de control transmitida por un transductor, una serie de números escritos con el teclado por un operador humano, un paquete de información enviado por un enlace de red o un archivo grande de datos recuperado de un almacenamiento secundario. La transformación quizá incluya una sola comparación lógica, un algoritmo numérico complicado o un enfoque de regla de inferencia para un sistema experto. La salida quizá encienda un diodo emisor de luz o genere un informe de 200 páginas. En efecto, es posible crear un modelo del flujo para cualquier sistema basado en computadora, sin importar su tamaño y complejidad.

CasaSegura

Modelado preliminar del comportamiento

La escena: Sala de juntas, continúa la reunión de requerimientos.

Participantes: Jamie Lazar, integrante del equipo de software; Vinod Raman, miembro del equipo de software; Ed Robbins, integrante del equipo de software; Doug Miller, gerente de ingeniería de software; tres trabajadores de mercadotecnia; un representante de ingeniería del producto, y un facilitador.

La conversación:

Facilitador: Estamos por terminar de hablar sobre la funcionalidad de seguridad del hogar de *CasaSegura*. Pero antes, quisiera que analizáramos el comportamiento de la función.

Persona de mercadotecnia: No entiendo lo que quiere decir con *comportamiento*.

Ed (sonríe): Es cuando le das un "tiempo fuera" al producto si se porta mal.

Facilitador: No exactamente. Permítanme explicarlo.

(El facilitador explica al equipo encargado de recabar los requerimientos y los fundamentos de modelado del comportamiento.)

Persona de mercadotecnia: Esto parece un poco técnico. No estoy seguro de ser de ayuda aquí.

Facilitador: Seguro que lo serás. ¿Qué comportamiento se observa desde el punto de vista de un usuario?

Persona de mercadotecnia: Mmm... bueno, el sistema estará *vigilando* los sensores. *Leerá comandos* del propietario. *Mostrará* su estado.

Facilitador: ¿Ves?, lo puedes hacer.

Jamie: También estará *interrogando* a la PC para determinar si hay alguna entrada desde ella, por ejemplo, un acceso por internet o información sobre la configuración.

Vinod: Sí, en realidad, *configurar* el sistema es un estado por derecho propio.

Doug: Muchachos, lo hacen bien. Pensemos un poco más... ¿hay alguna forma de hacer un diagrama de todo esto?

Facilitador: Sí la hay, pero la dejaremos para la próxima reunión.

Figura N°2-11 - Modelado preliminar del comportamiento

Patrones de análisis

Cualquiera que haya hecho la ingeniería de los requerimientos en varios proyectos de software ha observado que ciertos problemas son recurrentes en todos ellos dentro de un dominio de aplicación específico[14]. Estos patrones de análisis sugieren soluciones (por ejemplo, una clase, función o comportamiento) dentro del dominio de la aplicación que pueden volverse a utilizar cuando se modelan muchas aplicaciones.

Geyer-Schulz y Hahsler sugieren dos beneficios asociados con el uso de patrones de análisis:

En primer lugar, los patrones de análisis aceleran el desarrollo de los modelos de análisis abstracto que capturan los principales requerimientos del problema concreto, debido a que proveen modelos de análisis reutilizables con ejemplos, así como una descripción de sus ventajas y limitaciones. En segundo lugar, los patrones de análisis facilitan la transformación

[14] En ciertos casos, los problemas vuelven a suceder sin importar el dominio de la aplicación. Por ejemplo, son comunes las características y funciones usadas para resolver problemas de la interfaz de usuario sin importar el dominio de la aplicación en consideración.

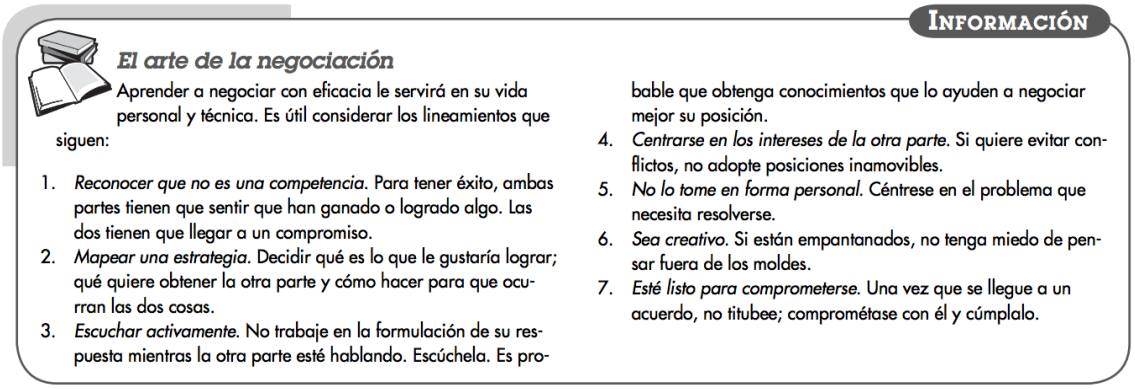
del modelo de análisis en un modelo del diseño, sugiriendo patrones de diseño y soluciones confiables para problemas comunes.

Los patrones de análisis se integran en el modelo del análisis, haciendo referencia al nombre del patrón. También se guardan en un medio de almacenamiento de modo que los ingenieros de requerimientos usen herramientas de búsqueda para encontrarlos y aplicarlos.

2.4.6 Requerimientos de las negociaciones

En un contexto ideal de la ingeniería de los requerimientos, las tareas de concepción, indagación y elaboración determinan los requerimientos del cliente con suficiente detalle como para avanzar hacia las siguientes actividades de la ingeniería de software. Desafortunadamente, esto rara vez ocurre. En realidad, se tiene que entrar en negociaciones con uno o varios participantes. En la mayoría de los casos, se pide a éstos que evalúen la funcionalidad, desempeño y otras características del producto o sistema, en contraste con el costo y el tiempo para entrar al mercado. El objetivo de esta negociación es desarrollar un plan del proyecto que satisfaga las necesidades del participante y que al mismo tiempo reflejan las restricciones del mundo real (por ejemplo, tiempo, personas, presupuesto, etc.) que se hayan establecido al equipo del software.

Las mejores negociaciones buscan un resultado "ganar-ganar". Es decir, los participantes ganan porque obtienen el sistema o producto que satisface la mayoría de sus necesidades y usted (como miembro del equipo de software) gana porque trabaja con presupuestos y plazos realistas y asequibles.



INFORMACIÓN

El arte de la negociación

Aprender a negociar con eficacia le servirá en su vida personal y técnica. Es útil considerar los lineamientos que siguen:

1. *Reconocer que no es una competencia.* Para tener éxito, ambas partes tienen que sentir que han ganado o logrado algo. Las dos tienen que llegar a un compromiso.
2. *Mapear una estrategia.* Decidir qué es lo que le gustaría lograr; qué quiere obtener la otra parte y cómo hacer para que ocurran las dos cosas.
3. *Escuchar activamente.* No trabaje en la formulación de su respuesta mientras la otra parte esté hablando. Escúchela. Es probable que obtenga conocimientos que lo ayuden a negociar mejor su posición.
4. *Centrarse en los intereses de la otra parte.* Si quiere evitar conflictos, no adopte posiciones inamovibles.
5. *No lo tome en forma personal.* Céntrese en el problema que necesita resolverse.
6. *Sea creativo.* Si están empantanados, no tenga miedo de pensar fuera de los moldes.
7. *Esté listo para comprometerse.* Una vez que se llegue a un acuerdo, no titubee; comprométase con él y cúmplalo.

Figura N°2-12 - El arte de la negociación

Boehm define un conjunto de actividades de negociación al principio de cada iteración del proceso de software. En lugar de una sola actividad de comunicación con el cliente, se definen las actividades siguientes:

1. Identificación de los participantes clave del sistema o subsistema.
2. Determinación de las "condiciones para ganar" de los participantes.
3. Negociación de las condiciones para ganar de los participantes a fin de reconciliarlas en un conjunto de condiciones ganar-ganar para todos los que intervienen (incluso el equipo de software).

La realización exitosa de estos pasos iniciales lleva a un resultado ganar-ganar, que se convierte en el criterio clave para avanzar hacia las siguientes actividades de la ingeniería de software.

CASASEGURA

El principio de una negociación

La escena: Oficina de Lisa Pérez, después de la primera reunión para recabar los requerimientos.

Participantes: Doug Miller, gerente de ingeniería de software, y Lisa Pérez, gerente de mercadotecnia.

La conversación:

Lisa: Pues escuché que la primera reunión salió realmente bien.

Doug: En realidad, sí. Enviaste buenos representantes... contribuyeron de verdad.

Lisa (sonríe): Sí; en realidad me dijeron que habían entrado y que no había sido una "actividad que les despejara la cabeza".

Doug (ríe): La próxima vez me aseguraré de quitarme la vena tecnológica... Mira, Lisa, creo que tenemos un problema para llegar a toda esa funcionalidad del sistema de seguridad para el hogar en las fechas que propone tu dirección. Sé que aún es temprano, pero hice un poco de planeación sobre las rodillas y...

Lisa (con el ceño fruncido): Lo debemos tener para esa fecha, Doug. ¿De qué funcionalidad hablas?

Doug: Supongo que podemos tener la funcionalidad completa en la fecha establecida, pero tendríamos que retrasar el acceso por internet hasta el segundo incremento.

Lisa: Doug, es el acceso por internet lo que da a *CasaSegura* su "súper" atractivo. Toda nuestra campaña de publicidad va a girar alrededor de eso. Lo tenemos que tener…

Doug: Entiendo la situación, de verdad. El problema es que para dar acceso por internet tendríamos que tener un sitio web por completo seguro y en operación. Esto requiere tiempo y personal. También tenemos que elaborar mucha funcionalidad adicional en la primera entrega… no creo que podamos hacerlo con los recursos que tenemos.

Lisa (todavía frunce el ceño): Ya veo, pero tienes que imaginar una manera de hacerlo. Tiene importancia crítica para las funciones de seguridad del hogar y también para otras… éstas podrían esperar hasta las siguientes entregas… estoy de acuerdo con eso.

Lisa y Doug parecen estar en suspenso, pero todavía deben negociar una solución a este problema. ¿Pueden "ganar" los dos en este caso? Si usted fuera el mediador, ¿qué sugeriría?

Figura N°2-13 - El principio de una negociación

2.4.7 Validación de los requerimientos

A medida que se crea cada elemento del modelo de requerimientos, se estudia para detectar inconsistencias, omisiones y ambigüedades. Los participantes asignan prioridades a los requerimientos representados por el modelo y se agrupan en paquetes de requerimientos que se implementarán como incrementos del software. La revisión del modelo de requerimientos aborda las preguntas siguientes:

- ¿Es coherente cada requerimiento con los objetivos generales del sistema o producto?
- ¿Se han especificado todos los requerimientos en el nivel apropiado de abstracción? Es decir, ¿algunos de ellos tienen un nivel de detalle técnico que resulta inapropiado en esta etapa?
- El requerimiento, ¿es realmente necesario o representa una característica agregada que tal vez no sea esencial para el objetivo del sistema?
- ¿Cada requerimiento está acotado y no es ambiguo?
- ¿Tiene atribución cada requerimiento? Es decir, ¿hay una fuente (por lo general una individual y específica) clara para cada requerimiento?
- ¿Hay requerimientos en conflicto con otros?
- ¿Cada requerimiento es asequible en el ambiente técnico que albergará el sistema o producto?
- Una vez implementado cada requerimiento, ¿puede someterse a prueba?
- El modelo de requerimientos, ¿refleja de manera apropiada la información, la función y el comportamiento del sistema que se va a construir?
- ¿Se ha "particionado" el modelo de requerimientos en forma que exponga información cada vez más detallada sobre el sistema?
- ¿Se ha usado el patrón de requerimientos para simplificar el modelo de éstos? ¿Se han validado todos los patrones de manera apropiada? ¿Son consistentes todos los patrones con los requerimientos del cliente?

Éstas y otras preguntas deben plantearse y responderse para garantizar que el modelo de requerimientos es una reflexión correcta sobre las necesidades del participante y que provee un fundamento sólido para el diseño.

2.4.8 Conclusiones del estado del arte

Las tareas de la ingeniería de requerimientos se realizan para establecer un fundamento sólido para el diseño y la construcción. La ingeniería de requerimientos ocurre durante las actividades de comunicación y modelado que se hayan definido para el proceso general del software. Los miembros del equipo de software llevan a cabo siete funciones de ingeniería de requerimientos: concepción, indagación, elaboración, negociación, especificación, validación y administración.

En la concepción del proyecto, los participantes establecen los requerimientos básicos del problema, definen las restricciones generales del proyecto, así como las características y funciones principales que debe presentar el sistema para cumplir sus objetivos. Esta información se mejora y amplía durante la indagación, actividad en la que se recaban los requerimientos y que hace uso de reuniones que lo facilitan, DFC y el desarrollo de escenarios de uso.

La elaboración amplía aún más los requerimientos en un modelo: una colección de elementos basados en escenarios, clases y comportamiento, y orientados al flujo. El modelo hace referencia a patrones de análisis: soluciones para problemas de análisis que se ha observado que son recurrentes en diferentes aplicaciones.

Conforme se identifican los requerimientos y se crea su modelo, el equipo de software y otros participantes negocian la prioridad, la disponibilidad y el costo relativo de cada requerimiento. Además, se valida cada requerimiento y su modelo como un todo comparado con las necesidades del cliente a fin de garantizar que va a construirse el sistema correcto.

2.5 MARCO DE INVESTIGACIÓN

En esta sección se propone y formula el marco de investigación que guiará este estudio. Para esto se desarrollan las preguntas de investigación que motivan la realización del estudio. Además se presentan las hipótesis que se desean someter a prueba en base a los resultados producto de la realización del estudio que en este documento se propone.

2.5.1 Preguntas de investigación

En base al estado del arte y las técnicas desarrolladas se evidencia que existe un interés en los procesos de ingeniería de requerimientos de los proyectos de ingeniería de software. Considerando los contratos de alta complejidad técnica que se celebran y la carencia técnica que cuentan en su contenido, es que se plantean las siguientes inquietudes:

- ¿Cómo se protegen las partes frente a los posibles abusos, falsedades o incumplimiento respecto de los contratos de software?
- ¿Cómo se realizan las definiciones de las funcionalidades del software?
- ¿Cómo se establecen las fechas de las entregas parciales y finales del software?
- ¿Cómo se definen los procesos de aceptación del software?
- ¿Se establecen criterios de calidad de software?
- ¿Existe legislación sobre la materia investigada a nivel nacional e internacional?
- ¿Existe una sóla línea de argumentación en la jurisprudencia sobre la materia o existen argumentos contradictorios?
- ¿Cuál es la controversia en los contratos de software?, ¿Cuáles son los problemas que se presentan?
- Y, en conclusión, tomando en consideración las respuestas obtenidas anteriormente, ¿existe alguna forma de reducir las falencias en los contratos de desarrollo y/o implementación de software?

2.5.2 Hipótesis

En función de los lineamientos propuestos por los trabajos recopilados y principalmente por la motivación que conduce la realización de este estudio con respecto a las causales que generan las controversias en los contratos ligados a la ingeniería de software, es que se formula la siguiente hipótesis que se desea validar con la realización de esta investigación.

Hipótesis: *"Si los contratos de desarrollo y/o implementación de software fueran elaborados por especialistas de los dos ámbitos por ambas entidades, se disminuiría el riesgo de fracaso en la entrega final del producto y cumplimiento efectivo del contrato celebrado"*.

2.6 RESUMEN

La práctica de la ingeniería de software incluye principios, conceptos, métodos y herramientas que los ingenieros de software aplican en todo el proceso de desarrollo. Todo proyecto de ingeniería de software es diferente. No obstante, existe un conjunto de principios generales que se aplican al proceso como un todo y a cada actividad estructural, sin importar cuál sea el proyecto o el producto.

Existe un conjunto de principios fundamentales que ayudan en la aplicación de un proceso de software significativo y en la ejecución de métodos de ingeniería de software eficaz. En el nivel del proceso, los principios fundamentales establecen un fundamento filosófico que guía al equipo de software cuando avanza por el proceso del software. En el nivel de la práctica, los principios fundamentales establecen un conjunto de valores y reglas que sirven como guía al analizar el diseño de un problema y su solución, al implementar ésta y al someterla a prueba para, finalmente, desplegar el software en la comunidad del usuario.

Los principios de comunicación se centran en la necesidad de reducir el ruido y mejorar el ancho de banda durante la conversación entre el desarrollador y el cliente. Ambas partes deben colaborar a fin de lograr la mejor comunicación.

Los principios de planeación establecen lineamientos para elaborar el mejor mapa del proceso hacia un sistema o producto terminado. El plan puede diseñarse sólo para un incremento del software, o para todo el proyecto. Sin que esto importe, debe definir lo que se hará, quién lo hará y cuándo se terminará el trabajo.

El modelado incluye tanto el análisis como el diseño, y describe representaciones cada vez más detalladas del software. El objetivo de los modelos es afirmar el entendimiento del trabajo que se va a hacer y dar una guía técnica a quienes implementarán el software. Los principios de modelado dan fundamento a los métodos y notación que se utilizan para crear representaciones del software.

La construcción incorpora un ciclo de codificación y pruebas en el que se genera código fuente para cierto componente y es sometido a pruebas. Los principios de codificación definen las acciones generales que deben tener lugar antes de que se escriba el código, mientras se escribe y una vez terminado. Aunque hay muchos principios para las pruebas,

sólo uno predomina: la prueba es el proceso que lleva a ejecutar un programa con objeto de encontrar un error.

El despliegue ocurre cuando se presenta al cliente un incremento de software, e incluye la entrega, apoyo y retroalimentación. Los principios clave para la entrega consideran la administración de las expectativas del cliente y darle información de apoyo adecuada sobre el software. El apoyo demanda preparación anticipada. La retroalimentación permite al cliente sugerir cambios que tengan valor para el negocio y que brinden al desarrollador información para el ciclo iterativo siguiente de ingeniería de software.

CAPÍTULO 3 METODOLOGÍA

En el nivel técnico, la ingeniería de software comienza con una serie de tareas de modelado que conducen a la especificación de los requerimientos y a la representación de un diseño del software que se va a elaborar. El modelo de requerimientos (un conjunto de modelos, en realidad) es la primera representación técnica de un sistema. En un libro fundamental sobre métodos para modelar los requerimientos, Tom DeMarco (DeMarco, 1979) describe el proceso de la manera siguiente:

Al mirar retrospectivamente los problemas y las fallas detectados en la fase de análisis, concluyó que es necesario agregar lo siguiente al conjunto de objetivos de dicha fase. Debe ser muy fácil dar mantenimiento a los productos del análisis. Esto se aplica en particular al Documento de Objetivos (especificación de los requerimientos del software). Los problemas grandes deben ser enfrentados con el empleo de un método eficaz para dividirlos. La especificación victoriana original resulta caducada. Deben usarse gráficas, siempre que sea posible. Es necesario diferenciar las consideraciones lógicas (esenciales) y las físicas (implementación). Finalmente, se necesita algo que ayude a dividir los requerimientos y a documentar dicha partición antes de elaborar la especificación, algunos medios para dar seguimiento a las interfaces y evaluar las nuevas herramientas para describir la lógica y la política, algo mejor que un texto narrativo.

Aunque DeMarco escribió hace más de un cuarto de siglo acerca de los atributos del modelado del análisis, sus comentarios aún son aplicables a los métodos y notaciones modernas del modelado de los requerimientos.

3.1 DEFINICIÓN CONCEPTUAL

El modelado de los requerimientos utiliza una combinación de texto y diagramas para ilustrarlos en forma que sea relativamente fácil de entender y, más importante, de revisar para corregir, completar y hacer congruente.

¿Quién lo hace? Un ingeniero de software (a veces llamado “analista”) construye el modelo con el uso de los requerimientos recabados del cliente.

¿Por qué es importante? Para validar los requerimientos del software se necesita estudiarlos desde diversos puntos de vista. En esta sección se considerará el modelado de los requerimientos desde tres perspectivas distintas: modelos basados en el escenario, modelos de datos (información) y modelos basados en la clase. Cada una representa a los requerimientos en una "dimensión" diferente, con lo que aumenta la probabilidad de detectar errores, de que afloren las inconsistencias y de que se revelen las omisiones.

¿Cuáles son los pasos? El modelado basado en escenarios es una representación del sistema desde el punto de vista del usuario. El modelado basado en datos recrea el espacio de información e ilustra los objetos de datos que manipulará el software y las relaciones entre ellos. El modelado orientado a clases define objetos, atributos y relaciones. Una vez que se crean los modelos preliminares, se mejoran y analizan para evaluar si están claros y completos, y si son consistentes.

¿Cuál es el producto final? Para construir el modelo de requerimientos, se escoge una amplia variedad de representaciones basadas en texto y en diagramas. Cada una de dichas representaciones, da una perspectiva de uno o más de los elementos del modelo.

¿Cómo me aseguro de que lo hice bien? Los productos del trabajo para modelar los requerimientos deben revisarse para saber si son correctos, completos y consistentes. Deben reflejar las necesidades de todos los participantes y establecer el fundamento desde el que se realizará el diseño.

3.1.1 Análisis de los requerimientos

El análisis de los requerimientos da como resultado la especificación de las características operativas del software, indica la interfaz de éste y otros elementos del sistema, y establece las restricciones que limitan al software. El análisis de los requerimientos permite al profesional (sin importar si se llama ingeniero de software, analista o modelista) construir sobre los requerimientos básicos establecidos durante las tareas de concepción, indagación y negociación, que son parte de la ingeniería de los requerimientos.

La acción de modelar los requerimientos da como resultado uno o más de los siguientes tipos de modelo:

- Modelos basados en el escenario de los requerimientos desde el punto de vista de distintos “actores” del sistema.
- Modelos de datos, que ilustran el dominio de información del problema.
- Modelos orientados a clases, que representan clases orientadas a objetos (atributos y operaciones) y la manera en la que las clases colaboran para cumplir con los requerimientos del sistema.
- Modelos orientados al flujo, que representan los elementos funcionales del sistema y la manera cómo transforman los datos a medida que se avanza a través del sistema.
- Modelos de comportamiento, que ilustran el modo en el que se comparte el software como consecuencia de “eventos” externos.

Estos modelos dan al diseñador del software, la información que se traduce en diseños de arquitectura, interfaces y componentes. Por último, el modelo de requerimientos (y la especificación de requerimientos de software) brinda al desarrollador y al cliente, los medios para evaluar la calidad una vez construido el software. En esta sección, nos centramos en el modelado basado en escenarios, técnica que cada vez es más popular entre la comunidad de la ingeniería de software; el modelado basado en datos, más especializado, apropiado en particular cuando debe crearse una aplicación o bien manipular un espacio complejo de información; y el modelado orientado a clases, representación de las clases orientada a objetos y a las colaboraciones resultantes que permiten que funcione el sistema.

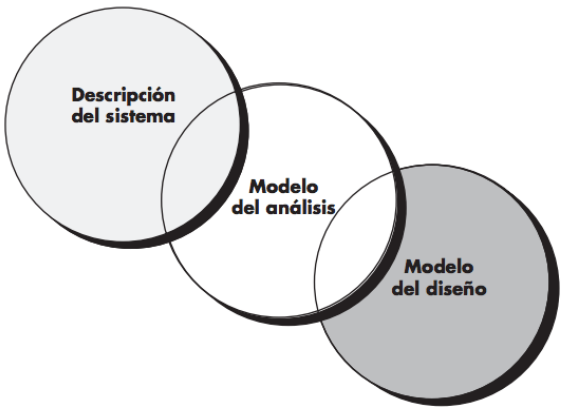


Figura N°3-01 - El modelo de requerimientos como puente entre la descripción del sistema y el modelo del diseño

Objetivos y filosofía general

Durante el modelado de los requerimientos, la atención se centra en qué, no en cómo. ¿Qué interacción del usuario ocurre en una circunstancia particular?, ¿qué objetos manipula el sistema?, ¿qué funciones debe realizar el sistema?, ¿qué comportamientos tiene el sistema?, ¿qué interfaces se definen? y ¿qué restricciones son aplicables?[15]

Anteriormente se menciono que en esta etapa tal vez no fuera posible tener la especificación completa de los requerimientos. El cliente quizá no esté seguro de qué es lo que requiere con precisión para ciertos aspectos del sistema. Puede ser que el desarrollador esté inseguro de que algún enfoque específico cumpla de manera apropiada la función y el desempeño. Estas realidades hablan a favor de un enfoque iterativo para el análisis y el modelado de los requerimientos. El analista debe modelar lo que se sabe y usar el modelo como base para el diseño del incremento del software[16].

El modelo de requerimientos debe lograr tres objetivos principales: 1) describir lo que requiere el cliente, 2) establecer una base para la creación de un diseño de software y 3) definir un conjunto de requerimientos que puedan validarse una vez construido el software. El modelo de análisis es un puente entre la descripción en el nivel del sistema que se centra en éste en lo general o en la funcionalidad del negocio que se logra con la aplicación de software, hardware, datos, personas y otros elementos del sistema y un diseño de software que describa la arquitectura de la aplicación del software, la interfaz del usuario y la estructura en el nivel del componente. Esta relación se ilustra en la figura 3-01.

Es importante observar que todos los elementos del modelo de requerimientos pueden rastrearse directamente hasta las partes del modelo del diseño. No siempre es posible la división clara entre las tareas del análisis y las del diseño en estas dos importantes actividades del modelado. Invariablemente, ocurre algo de diseño como parte del análisis y algo de análisis se lleva a cabo durante el diseño.

[15] Debe notarse que, a medida que los clientes tienen más conocimientos tecnológicos, hay una tendencia hacia la especificación del cómo tanto como del que. Sin embargo, la atención debe centrarse en el que.
[16] En un esfuerzo por entender mejor los requerimientos para el sistema, el equipo del software tiene la alternativa de escoger la creación de un prototipo.

Reglas prácticas del análisis

Arlow y Neustadt (Arlow & Neustadt, 2002) sugieren cierto número de reglas prácticas útiles que deben seguirse cuando se crea el modelo del análisis:

- El modelo debe centrarse en los requerimientos que sean visibles dentro del problema o dentro del dominio del negocio. El nivel de abstracción debe ser relativamente elevado. "No se empantane en los detalles" (Arlow & Neustadt, 2002) que traten de explicar cómo funciona el sistema.
- Cada elemento del modelo de requerimientos debe agregarse al entendimiento general de los requerimientos del software y dar una visión del dominio de la información, de la función y del comportamiento del sistema.
- Hay que retrasar las consideraciones de la infraestructura y otros modelos no funcionales hasta llegar a la etapa del diseño. Es decir, quizá se requiera una base de datos, pero las clases necesarias para implementarla, las funciones requeridas para acceder a ella y el comportamiento que tendrá cuando se use sólo deben considerarse después de que se haya terminado el análisis del dominio del problema.
- Debe minimizarse el acoplamiento a través del sistema. Es importante representar las relaciones entre las clases y funciones. Sin embargo, si el nivel de "interconectividad" es extremadamente alto, deben hacerse esfuerzos para reducirlo.
- Es seguro que el modelo de requerimientos agrega valor para todos los participantes. Cada actor tiene su propio uso para el modelo. Por ejemplo, los participantes de negocios deben usar el modelo para validar los requerimientos; los diseñadores deben usarlo como pase para el diseño; el personal de aseguramiento de la calidad lo debe emplear como ayuda para planear las pruebas de aceptación.
- Mantener el modelo tan sencillo como se pueda. No genere diagramas adicionales si no agregan nueva información. No utilice notación compleja si basta una sencilla lista.

Análisis del dominio

Al estudiar la ingeniería de requerimientos, se menciona que es frecuente que haya patrones de análisis que se repiten en muchas aplicaciones dentro de un dominio de negocio específico. Si éstos se definen y clasifican en forma tal que puedan reconocerse y aplicarse para resolver problemas comunes, la creación del modelo del análisis es más expedita. Más importante aún es que la probabilidad de aplicar patrones de diseño y

componentes de software ejecutable se incrementa mucho. Esto mejora el tiempo para llegar al mercado y reduce los costos de desarrollo.

Pero, ¿cómo se reconocen por primera vez los patrones de análisis y clases? ¿Quién los define, clasifica y prepara para usarlos en los proyectos posteriores? La respuesta a estas preguntas está en el análisis del dominio. Firesmith (Firesmith, 1993) lo describe del siguiente modo:

El análisis del dominio del software es la identificación, análisis y especificación de los requerimientos comunes, a partir de un dominio de aplicación específica, normalmente para usarlo varias veces en múltiples proyectos dentro del dominio de la aplicación. El análisis del dominio orientado a objetos es la identificación, análisis y especificación de capacidades comunes y reutilizables dentro de un dominio de aplicación específica en términos de objetos, clases, subensambles y estructuras comunes.

El dominio de aplicación[17] específica se extiende desde el control electrónico de aviones hasta la banca, de los juegos de video en multimedios al software incrustado en equipos médicos. La meta del análisis del dominio es clara: encontrar o crear aquellas clases o patrones de análisis que sean aplicables en lo general, de modo que puedan volverse a usar.

Con el empleo de la terminología que se introdujo antes en este libro, el análisis del dominio puede considerarse como una actividad sombrilla para el proceso del software. Esto significa que el análisis del dominio es una actividad de la ingeniería de software que no está conectada con ningún proyecto de software. En cierta forma, el papel del analista del dominio es similar al de un maestro herrero en un ambiente de manufactura pesada. El trabajo del herrero es diseñar y fabricar herramientas que utilicen muchas personas que hacen trabajos similares pero no necesariamente iguales. El papel del analista de dominio[18] es descubrir y definir patrones de análisis, clases de análisis e información relacionada que

[17] Un punto de vista complementario del análisis del dominio involucra el modelado de éste, de manera que los ingenieros del software y otros participantes aprendan más al respecto, no todas las clases de dominio necesariamente dan como resultado el desarrollo de clases reutilizables.

[18] No suponga que el ingeniero de software no necesita entender el dominio de la aplicación tan sólo porque hay un analista del dominio trabajando. Todo miembro del equipo del software debe entender algo del dominio en el que se va a colocar el software.

pueda ser utilizada por mucha gente que trabaja en aplicaciones similares, pero que no son necesariamente las mismas.

La figura 3-02 ilustra entradas y salidas clave para el proceso de análisis del dominio. Las fuentes de conocimiento del dominio se mapean con el fin de identificar los objetos que pueden reutilizarse a través del dominio.

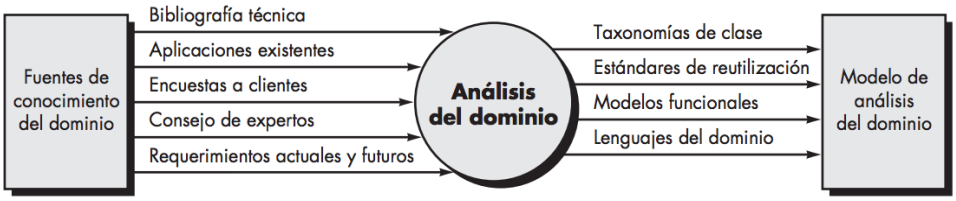


Figura N°3-02 - Entradas y salidas para el análisis del dominio

Enfoques del modelado de requerimientos

Un enfoque del modelado de requerimientos, llamado análisis estructurado, considera que los datos y los procesos que los transforman son entidades separadas. Los objetos de datos se modelan de modo que se definan sus atributos y relaciones. Los procesos que manipulan a los objetos de datos se modelan en forma que se muestre cómo transforman a los datos a medida que los objetos que se corresponden con ellos fluyen por el sistema.

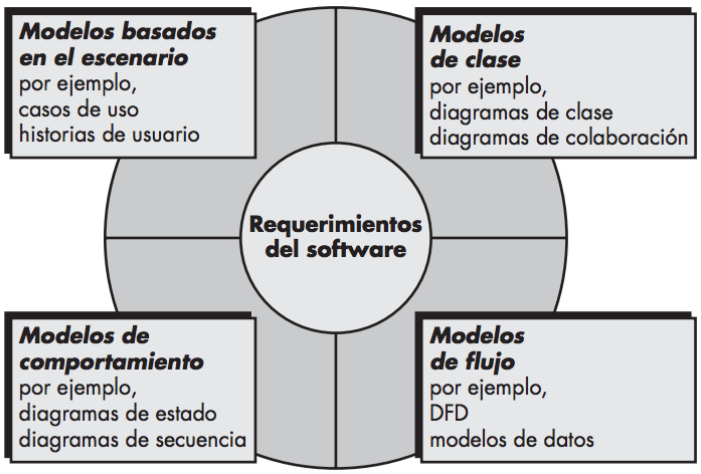


Figura N°3-03 - Elementos del modelo de análisis

Un segundo enfoque del modelado del análisis, llamado análisis orientado a objetos, se centra en la definición de las clases y en la manera en la que colaboran uno con el otro para cumplir los requerimientos. El UML y el proceso unificado están orientados a objetos, sobre todo.

Aunque el modelo de requerimientos propuesto en este libro combina características de ambos enfoques, los equipos de software escogen con frecuencia uno y excluyen todas las representaciones del otro. La pregunta no es cuál es mejor, sino qué combinación de representaciones proporcionará a los participantes el mejor modelo de requerimientos del software y el puente más eficaz para el diseño del mismo.

Cada elemento del modelo de requerimientos (ver la figura 3-03) presenta el problema desde diferentes puntos de vista. Los elementos basados en el escenario ilustran cómo interactúa el usuario con el sistema y la secuencia específica de actividades que ocurren cuando se utiliza el software. Los elementos basados en la clase modelan los objetos que el sistema manipulará, las operaciones que se aplicarán a ellos para realizar dicha manipulación, las relaciones (algunas jerárquicas) entre los objetos y las colaboraciones que ocurrirán entre las clases que se definan. Los elementos del comportamiento ilustran la forma en la que los eventos externos cambian el estado del sistema o las clases que residen dentro de éste. Por último, los elementos orientados al flujo representan al sistema como una transformación de la información e ilustran la forma en la que se transforman los objetos de datos cuando fluyen a través de las distintas funciones del sistema.

El modelado del análisis lleva a la obtención de cada uno de estos elementos de modelado. Sin embargo, el contenido específico de cada elemento (por ejemplo, los diagramas que se emplean para construir el elemento y el modelo) tal vez difiera de un proyecto a otro. Como se ha dicho varias veces en este libro, el equipo del software debe trabajar para mantenerlo sencillo. Sólo deben usarse elementos de modelado que agreguen valor al modelo.

3.1.2 Modelado basado en escenarios

Aunque el éxito de un sistema o producto basado en computadora se mide de muchas maneras, la satisfacción del usuario ocupa el primer lugar de la lista. Si se entiende cómo desean interactuar los usuarios finales (y otros actores) con un sistema, el equipo del

software estará mejor preparado para caracterizar adecuadamente los requerimientos y hacer análisis significativos y modelos del diseño. Entonces, el modelado de los requerimientos con UML comienza con la creación de escenarios en forma de casos de uso, diagramas de actividades y diagramas tipo carril de natación.

Creación de un caso preliminar de uso

Alistair Cockburn (Cockburn, 2001) caracteriza un caso de uso como un "contrato para el comportamiento". El "contrato" define la forma en la que un actor[19] utiliza un sistema basado en computadora para alcanzar algún objetivo. En esencia, un caso de uso capta las interacciones que ocurren entre los productores y consumidores de la información y el sistema en sí. En esta sección se estudiará la forma en la que se desarrollan los casos de uso como parte de los requerimientos de la actividad de modelado[20].

Anteriormente mencionamos que un caso de uso describe en lenguaje claro un escenario específico desde el punto de vista de un actor definido. Pero, ¿cómo se sabe sobre qué escribir, cuánto escribir sobre ello, cuán detallada hacer la descripción y cómo organizarla? Son preguntas que deben responderse si los casos de uso han de tener algún valor como herramienta para modelar los requerimientos.

¿Sobre qué escribir? Las dos primeras tareas de la ingeniería de requerimientos -concepción e indagación- dan la información que se necesita para comenzar a escribir casos de uso. Las reuniones para recabar los requerimientos, el DEC, y otros mecanismos para obtenerlos se utilizan para identificar a los participantes, definir el alcance del problema, especificar los objetivos operativos generales, establecer prioridades, delinear todos los requerimientos funcionales conocidos y describir las cosas (objetos) que serán manipuladas por el sistema.

Para comenzar a desarrollar un conjunto de casos de uso, se enlistan las funciones o actividades realizadas por un actor específico. Éstas se obtienen de una lista de las funciones requeridas del sistema, por medio de conversaciones con los participantes o con la evaluación de los diagramas de actividades desarrollados como parte del modelado de los requerimientos.

[19] Un actor no es una persona específica sino el rol que desempeña ésta (o un dispositivo) en un contexto específico. Un actor "llama al sistema para que entregue uno de sus servicios".
[20] Los casos de uso son una parte del modelado del análisis de importancia especial para las interfaces.

La función (subsistema) de vigilancia de CasaSegura estudiada en el recuadro identifica las funciones siguientes (lista abreviada) que va a realizar el actor propietario:

- Seleccionar cámara para ver.
- Pedir vistas reducidas de todas las cámaras.
- Mostrar vistas de las cámaras en una ventana de PC.
- Controlar el ángulo y acercamiento de una cámara específica.
- Grabar la salida de cada cámara en forma selectiva.
- Reproducir la salida de una cámara.
- Acceder por internet a la vigilancia con cámaras.

A medida que avanzan las conversaciones con el participante (quien juega el papel de propietario), el equipo que recaba los requerimientos desarrolla casos de uso para cada una de las funciones estudiadas. En general, los casos de uso se escriben primero en forma de narración informal. Si se requiere más formalidad, se reescribe el mismo caso con el empleo de un formato estructurado, similar al propuesto en el capítulo y que se reproduce en un recuadro más adelante, en esta sección.

Para ilustrar esto, considere la función acceder a la vigilancia con cámaras por internet-mostrar vistas de cámaras (AVC-MVC). El participante que tenga el papel del actor llamado propietario escribiría una narración como la siguiente:

Caso de uso: acceder a la vigilancia con cámaras por internet, mostrar vistas de cámaras (AVC-MVC)
Actor: propietario

Si estoy en una localidad alejada, puedo usar cualquier PC con un software de navegación apropiado para entrar al sitio web de Productos CasaSegura. Introduzco mi identificación de usuario y dos niveles de claves; una vez validadas, tengo acceso a toda la funcionalidad de mi sistema instalado. Para acceder a la vista de una cámara específica, selecciono "vigilancia" de los botones mostrados para las funciones principales. Luego selecciono "escoger una cámara" y aparece el plano de la casa. Después elijo la cámara que me interesa. Alternativamente, puedo ver la vista de todas las cámaras simultáneamente si selecciono "todas las cámaras".

Una vez que escojo una, selecciono "vista" y en la ventana que cubre la cámara aparece una vista con velocidad de un cuadro por segundo. Si quiero cambiar entre las cámaras, selecciono "escoger una cámara" y desaparece la vista original y de nuevo se muestra el plano de la casa. Después, selecciono la cámara que me interesa. Aparece una nueva ventana de vistas.

Una variación de la narrativa del caso de uso presenta la interacción como una secuencia ordenada de acciones del usuario. Cada acción está representada como enunciado declarativo. Al visitar la función ACS-DCV, se escribiría lo siguiente:

Caso de uso: acceder a la vigilancia con cámaras por internet, mostrar vistas de cámaras (AVC-MVC)
Actor: propietario

1. El propietario accede al sitio web Productos CasaSegura.
2. El propietario introduce su identificación de usuario.
3. El propietario escribe dos claves (cada una de al menos ocho caracteres de longitud).
4. El sistema muestra los botones de todas las funciones principales.
5. El propietario selecciona "vigilancia" de los botones de las funciones principales.
6. El propietario elige "seleccionar una cámara".
7. El sistema presenta el plano de la casa.
8. El propietario escoge el ícono de una cámara en el plano de la casa.
9. El propietario selecciona el botón "vista".
10. El sistema presenta la ventana de vista identificada con la elección de la cámara.
11. El sistema muestra un video dentro de la ventana a velocidad de un cuadro por segundo.

Es importante observar que esta presentación en secuencia no considera interacciones alternativas (la narración fluye con más libertad y representa varias alternativas). Los casos de este tipo en ocasiones se denominan escenarios primarios.

Mejora de un caso de uso preliminar
Para entender por completo la función que describe un caso de uso, es esencial describir interacciones alternativas. Después se evalúa cada paso en el escenario primario, planteando las preguntas siguientes:

- ¿El actor puede emprender otra acción en este punto?
- ¿Es posible que el actor encuentre alguna condición de error en este punto? Si así fuera, ¿cuál podría ser?
- En este punto, ¿es posible que el actor encuentre otro comportamiento (por ejemplo, alguno que sea invocado por cierto evento fuera del control del actor)? En ese caso, ¿cuál sería?

Las respuestas a estas preguntas dan como resultado la creación de un conjunto de escenarios secundarios que forman parte del caso de uso original, pero que representan comportamientos alternativos. Por ejemplo, consideremos los pasos 6 y 7 del escenario primario ya descrito:

6. El propietario elige "seleccionar una cámara".
7. El sistema presenta el plano de la casa.

¿El actor puede emprender otra acción en este punto? La respuesta es "sí". Al analizar la narración de flujo libre, el actor puede escoger mirar vistas de todas las cámaras simultáneamente. Entonces, un escenario secundario sería "observar vistas instantáneas de todas las cámaras".

¿Es posible que el actor encuentre alguna condición de error en este punto? Cualquier número de condiciones de error puede ocurrir cuando opera un sistema basado en computadora. En este contexto, sólo se consideran las condiciones que sean probables como resultado directo de la acción descrita en los pasos 6 o 7. De nuevo, la respuesta es "sí". Tal vez nunca se haya configurado un plano con iconos de cámara. Entonces, elegir "seleccionar una cámara" da como resultado una condición de error: "No hay plano configurado para esta casa[21]." Esta condición de error se convierte en un escenario secundario.

En este punto, ¿es posible que el actor encuentre otro comportamiento (por ejemplo, alguno que sea invocado por cierto evento fuera del control del actor)? Otra vez, la respuesta es "sí".

[21] En este caso, otro actor, administrador del sistema, tendría que configurar el plano de la casa, instalar e inicializar todas las cámaras (por ejemplo, asignar una identificación a los equipos) y probar cada una para garantizar que se encuentren accesibles por el sistema y a través del plano de la casa.

A medida que ocurran los pasos 6 y 7, el sistema puede hallar una condición de alarma. Esto dará como resultado que el sistema desplegará una notificación especial de alarma (tipo, ubicación, acción del sistema) y proporcionará al actor varias opciones relevantes según la naturaleza de la alarma. Como este escenario secundario puede ocurrir en cualquier momento para prácticamente todas las interacciones, no se vuelve parte del caso de uso AVC-MVC. En vez de ello, se desarrollará un caso de uso diferente -Condición de alarma encontrada- al que se hará referencia desde otros casos según se requiera.

Cada una de las situaciones descritas en los párrafos precedentes se caracteriza como una excepción al caso de uso. Una excepción describe una situación (ya sea condición de falla o alternativa elegida por el actor) que ocasiona que el sistema presente un comportamiento algo distinto.

Cockburn (Cockburn, 2001) recomienda el uso de una sesión de "lluvia de ideas" para obtener un conjunto razonablemente complejo de excepciones para cada caso de uso. Además de las tres preguntas generales ya sugeridas en esta sección, también deben explorarse los siguientes aspectos:

- ¿Existen casos en los que ocurra alguna "función de validación" durante este caso de uso? Esto implica que la función de validación es invocada y podría ocurrir una potencial condición de error.
- ¿Hay casos en los que una función (o actor) de soporte falle en responder de manera apropiada? Por ejemplo, una acción de usuario espera una respuesta pero la función que ha de responder se cae.
- ¿El mal desempeño del sistema da como resultado acciones inesperadas o impropias? Por ejemplo, una interfaz con base en web responde con demasiada lentitud, lo que da como resultado que un usuario haga selecciones múltiples en un botón de procesamiento. Estas selecciones se forman de modo equivocado y, en última instancia, generan un error.

La lista de extensiones desarrollada como consecuencia de preguntar y responder estas preguntas debe "racionalizarse" con el uso de los siguientes criterios: una excepción debe describirse dentro del caso de uso si el software la puede detectar y debe manejarla una

vez detectada. En ciertos casos, una excepción precipitará el desarrollo de otro caso de uso (el de manejar la condición descrita).

Escritura de un caso de uso formal

En ocasiones, para modelar los requerimientos es suficiente con los casos de uso informales presentados en la sección anterior. Sin embargo, cuando un caso de uso involucra una actividad crítica o cuando describe un conjunto complejo de etapas con un número significativo de excepciones, es deseable un enfoque más formal.

CASASEGURA

Formato de caso de uso para vigilancia

Caso de uso: Acceder a la vigilancia con cámaras por internet, mostrar vistas de cámaras (AVC-MVC).

Iteración: 2, última modificación: 14 de enero por V. Raman.

Actor principal: Propietario.

Objetivo en contexto: Ver la salida de las cámaras colocadas en la casa desde cualquier ubicación remota por medio de internet.

Precondiciones: El sistema debe estar configurado por completo; deben obtenerse las identificaciones y claves de usuario apropiadas.

Disparador: El propietario decide ver dentro de la casa mientras está fuera.

Escenario:

1. El propietario se registra en el sitio web *Productos CasaSegura*.
2. El propietario introduce su identificación de usuario.
3. El propietario proporciona dos claves (cada una con longitud de al menos ocho caracteres).
4. El sistema despliega todos los botones de las funciones principales.
5. El propietario selecciona "vigilancia" entre los botones de funciones principales.
6. El propietario escoge "seleccionar una cámara".
7. El sistema muestra el plano de la casa.
8. El propietario selecciona un ícono de cámara en el plano de la casa.
9. El propietario pulsa el botón "vista".
10. El sistema muestra la ventana de la vista de la cámara identificada.
11. El sistema presenta una salida de video dentro de la ventana de vistas, con una velocidad de un cuadro por segundo.

Excepciones:

1. La identificación o las claves son incorrectas o no se reconocen (véase el caso de uso **Validar identificación y claves**).
2. La función de vigilancia no está configurada para este sistema (el sistema muestra el mensaje de error apropiado; véase el caso de uso **Configurar la función de vigilancia**).
3. El propietario selecciona "Mirar vistas reducidas de todas las cámaras" (véase el caso de uso **Mirar vistas reducidas de todas las cámaras**).
4. No se dispone o no se ha configurado el plano de la casa (se muestra el mensaje de error apropiado y véase el caso de uso **Configurar plano de la casa**).
5. Se encuentra una condición de alarma (véase el caso de uso **Condición de alarma encontrada**).

Prioridad: Moderada, por implementarse después de las funciones básicas.

Cuándo estará disponible: En el tercer incremento.

Frecuencia de uso: Frecuencia moderada.

Canal al actor: A través de un navegador con base en PC y conexión a internet.

Actores secundarios: Administrador del sistema, cámaras.

Canales a los actores secundarios:

1. Administrador del sistema: sistema basado en PC.
2. Cámaras: conectividad inalámbrica.

Asuntos pendientes:

1. ¿Qué mecanismos protegen el uso no autorizado de esta capacidad por parte de los empleados de *Productos CasaSegura*?
2. Es suficiente la seguridad? El acceso ilegal a esta característica representaría una invasión grave de la privacidad.
3. ¿Será aceptable la respuesta del sistema por internet dado el ancho de banda que requieren las vistas de las cámaras?
4. ¿Desarrollaremos una capacidad que provea el video a una velocidad más alta en cuadros por segundo cuando se disponga de conexiones con un ancho de banda mayor?

Figura N°3-04 - Formato de caso de uso para vigilancia

El caso de uso AVC-MVC mostrado en la figura 3-04, sigue el guión común para los casos de uso formales. El objetivo en contexto identifica el alcance general del caso de uso. La precondición describe lo que se sabe que es verdadero antes de que inicie el caso de uso. El disparador (o trigger) identifica el evento o condición que "hace que comience el caso de uso". El escenario enlista las acciones específicas que requiere el actor, y las respuestas apropiadas del sistema. Las excepciones identifican las situaciones detectadas cuando se mejora el caso de uso preliminar. Pueden incluirse o no encabezados adicionales y se explican por sí mismos en forma razonable.

En muchos casos, no hay necesidad de crear una representación gráfica de un escenario de uso. Sin embargo, la representación con diagramas facilita la comprensión, en particular cuando el escenario es complejo. Como ya se dijo en este libro, UML cuenta con la capacidad de hacer diagramas de casos de uso. La figura 3-05 ilustra un diagrama de caso de uso preliminar para el producto CasaSegura. Cada caso de uso está representado por un óvalo. En esta sección sólo se estudia el caso de uso AVC-MVC.

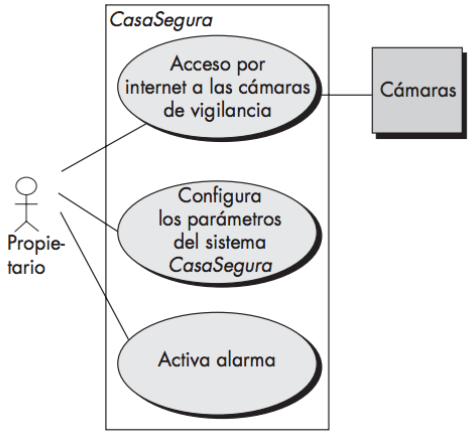


Figura N°3-05 - Diagrama de caso de uso preliminar para el sistema CasaSegura

Toda notación de modelado tiene sus limitaciones, y la del caso de uso no es la excepción. Como cualquier otra forma de descripción escrita, un caso de uso es tan bueno como lo sea(n) su(s) autor(es). Si la descripción es poco clara, el caso de uso será confuso o ambiguo. Un caso de uso se centra en los requerimientos funcionales y de comportamiento, y por lo general es inapropiado para requerimientos disfuncionales. Para situaciones en las que el modelo de requerimientos deba tener detalle y precisión significativos (por ejemplo, sistemas críticos de seguridad), tal vez no sea suficiente un caso de uso.

Sin embargo, el modelado basado en escenarios es apropiado para la gran mayoría de todas las situaciones que encontrará un ingeniero de software. Si se desarrolla bien, el caso de uso proporciona un beneficio sustancial como herramienta de modelado.

3.1.3 Modelos UML que proporcionan el caso de uso

Hay muchas situaciones de modelado de requerimientos en las que un modelo basado en texto -incluso uno tan sencillo como un caso de uso- tal vez no brinde información en forma clara y concisa. En tales casos, es posible elegir de entre una amplia variedad de modelos UML gráficos.

Desarrollo de un diagrama de actividades

El diagrama de actividad UML enriquece el caso de uso al proporcionar una representación gráfica del flujo de interacción dentro de un escenario específico. Un diagrama de actividades es similar a uno de flujo, y utiliza rectángulos redondeados para denotar una función específica del sistema, flechas para representar flujo a través de éste, rombos de decisión para ilustrar una ramificación de las decisiones (cada flecha que salga del rombo se etiqueta) y líneas continuas para indicar que están ocurriendo actividades en paralelo. En la figura 3-06 se presenta un diagrama de actividades para el caso de uso AVC-MVC.

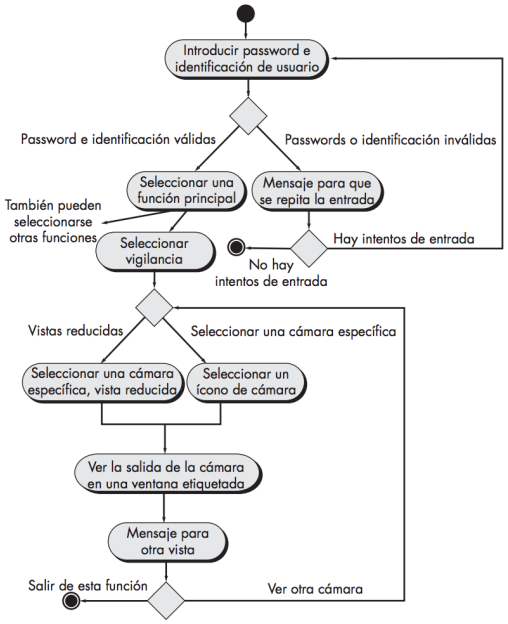


Figura N°3-06 - Diagrama de actividades para la función Acceder a la vigilancia con cámaras por internet, mostrar vistas de cámaras.

Por ejemplo, un usuario quizá sólo haga algunos intentos de introducir su identificación y password. Esto se representa por el rombo de decisión debajo de "Mensaje para que se repita la entrada".

Diagramas de canal (swimlane)

El diagrama de canal de UML es una variación útil del diagrama de actividades y permite representar el flujo de actividades descritas por el caso de uso; al mismo tiempo, indica qué actor (si hubiera muchos involucrados en un caso específico de uso) o clase de análisis (se estudia más adelante, en este capítulo) es responsable de la acción descrita por un rectángulo de actividad. Las responsabilidades se representan con segmentos paralelos que dividen el diagrama en forma vertical, como los canales o carriles de una alberca.

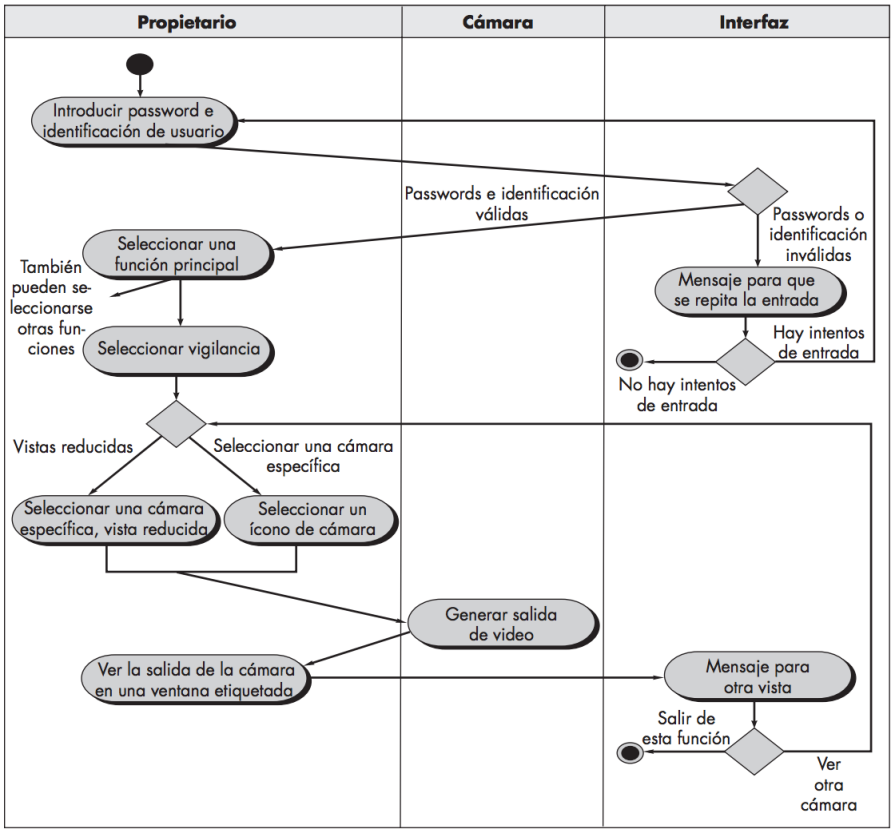


Figura N°3-07 - Diagrama de canal para la función Acceder a la vigilancia con cámaras por internet, mostrar vistas de cámaras

Son tres las clases de análisis: Propietario, Cámara e Interfaz, que tienen responsabilidad directa o indirecta en el contexto del diagrama de actividades representado en la figura 3-06. En relación con la figura 3-07, el diagrama de actividades se reacomodó para que las

actividades asociadas con una clase de análisis particular queden dentro del canal de dicha clase. Por ejemplo, la clase Interfaz representa la interfaz de usuario como la ve el propietario. El diagrama de actividades tiene dos mensajes que son responsabilidad de la interfaz: "mensaje para que se repita la entrada" y "mensaje para otra vista". Estos mensajes y las decisiones asociadas con ellos caen dentro del canal Interfaz. Sin embargo, las flechas van de ese canal de regreso al de Propietario, donde ocurren las acciones de éste.

Los casos de uso, junto con los diagramas de actividades y de canal, están orientados al procedimiento. Representan la manera en la que los distintos actores invocan funciones específicas (u otros pasos del procedimiento) para satisfacer los requerimientos del sistema. Pero una vista del procedimiento de los requerimientos representa una sola dimensión del sistema.

3.1.4 Conceptos del modelado de datos

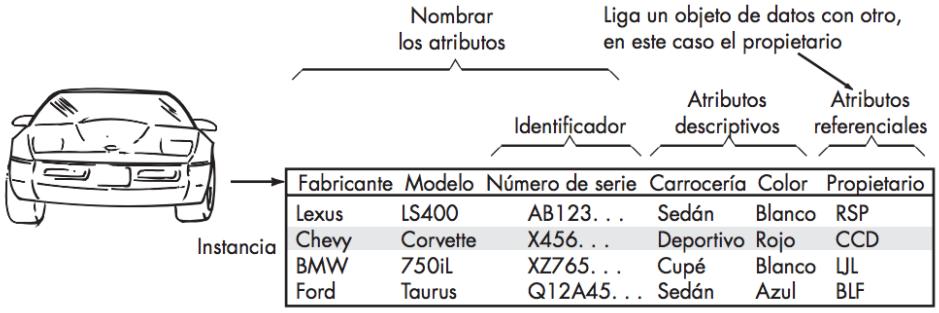
Si los requerimientos del software incluyen la necesidad de crear, ampliar o hacer interfaz con una base de datos, o si deben construirse y manipularse estructuras de datos complejas, el equipo del software tal vez elija crear un modelo de datos como parte del modelado general de los requerimientos. Un ingeniero o analista de software define todos los objetos de datos que se procesan dentro del sistema, la relación entre ellos y otro tipo de información que sea pertinente para las relaciones. El diagrama entidad-relación (DER) aborda dichos aspectos y representa todos los datos que se introducen, almacenan, transforman y generan dentro de una aplicación.

Objetos de datos

Un objeto de datos es una representación de información compuesta que debe ser entendida por el software. Información compuesta quiere decir algo que tiene varias propiedades o atributos diferentes. Por tanto, el ancho (un solo valor) no sería un objeto de datos válido, pero las dimensiones (que incorporan altura, ancho y profundidad) sí podrían definirse como un objeto.

Un objeto de datos puede ser una entidad externa (por ejemplo, cualquier cosa que produzca o consuma información), una cosa (por ejemplo, un informe o pantalla), una ocurrencia (como una llamada telefónica) o evento (por ejemplo, una alarma), un rol (un vendedor), una unidad organizacional (por ejemplo, el departamento de contabilidad), un

lugar (como una bodega) o estructura (como un archivo). Por ejemplo, una persona o un auto pueden considerarse como objetos de datos en tanto cada uno se define en términos de un conjunto de atributos. La descripción del objeto de datos incorpora a ésta todos sus atributos.



Fabricante	Modelo	Número de serie	Carrocería	Color	Propietario
Lexus	LS400	AB123. . .	Sedán	Blanco	RSP
Chevy	Corvette	X456. . .	Deportivo	Rojo	CCD
BMW	750iL	XZ765. . .	Cupé	Blanco	LJL
Ford	Taurus	Q12A45. . .	Sedán	Azul	BLF

Figura N°3-08 - Representación tabular de objetos de datos

Un objeto de datos contiene sólo datos -dentro de él no hay referencia a las operaciones que se apliquen sobre los datos-. Entonces, el objeto de datos puede representarse en forma de tabla, como la que se muestra en la figura 3-08. Los encabezados de la tabla reflejan atributos del objeto. En este caso, un auto se define en términos de fabricante, modelo, número de serie, tipo de carrocería, color y propietario. El cuerpo de la tabla representa instancias específicas del objeto de datos. Por ejemplo, un Chevy Corvette es una instancia del objeto de datos auto.

Relaciones

Los objetos de datos están conectados entre sí de diferentes maneras. Considere dos objetos de datos, persona y auto. Estos objetos se representan con la notación simple que se ilustra en la figura 3-09a. Se establece una conexión entre persona y auto porque ambos objetos están relacionados. Pero, ¿cuál es esa relación? Para determinarlo, debe entenderse el papel de las personas (propiedad, en este caso) y los autos dentro del contexto del software que se va a elaborar. Se establece un conjunto de parejas objeto/relación que definan las relaciones relevantes. Por ejemplo:

- Una persona posee un auto.
- Una persona es asegurada para que maneje un auto.

Las relaciones posee y es asegurada para que maneje definen las conexiones relevantes entre persona y auto. La figura 3-09b ilustra estas parejas objeto-relación. Las flechas en esa figura dan información importante sobre la dirección de la relación y es frecuente que reduzcan las ambigüedades o interpretaciones erróneas.

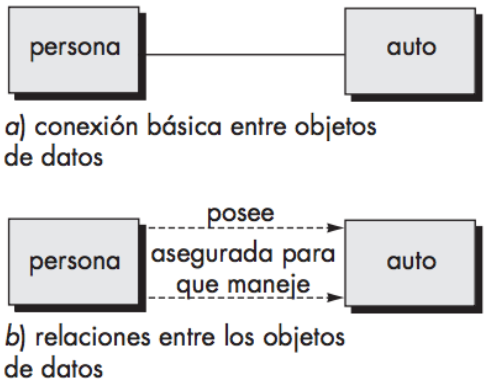


Figura N°3-09 - Relaciones entre objetos de datos

3.1.5 Modelado basado en clases

El modelado basado en clases representa los objetos que manipulará el sistema, las operaciones (también llamadas métodos o servicios) que se aplicarán a los objetos para efectuar la manipulación, las relaciones (algunas de ellas jerárquicas) entre los objetos y las colaboraciones que tienen lugar entre las clases definidas. Los elementos de un modelo basado en clases incluyen las clases y los objetos, atributos, operaciones, modelos clase-responsabilidad-colaborador (CRC), diagramas de colaboración y paquetes. En las secciones siguientes se presenta una serie de lineamientos informales que ayudarán a su identificación y representación.

Identificación de las clases de análisis

Al mirar una habitación, se observa un conjunto de objetos físicos que se identifican, clasifican y definen fácilmente (en términos de atributos y operaciones). Pero cuando se "ve" el espacio del problema de una aplicación de software, las clases (y objetos) son más difíciles de concebir.

Se comienza por identificar las clases mediante el análisis de los escenarios de uso desarrollados como parte del modelo de requerimientos y la ejecución de un "análisis gramatical" sobre los casos de uso desarrollados para el sistema que se va a construir. Las

clases se determinan subrayando cada sustantivo o frase que las incluya para introducirlo en una tabla simple. Deben anotarse los sinónimos. Si la clase (sustantivo) se requiere para implementar una solución, entonces forma parte del espacio de solución; de otro modo, si sólo es necesaria para describir la solución, es parte del espacio del problema.

Pero, ¿qué debe buscarse una vez identificados todos los sustantivos? Las clases de análisis se manifiestan en uno de los modos siguientes:

- Entidades externas (por ejemplo, otros sistemas, dispositivos y personas) que producen o consumen la información que usará un sistema basado en computadora.
- Cosas (reportes, pantallas, cartas, señales, etc.) que forman parte del dominio de información para el problema.
- Ocurrencias o eventos (como una transferencia de propiedad o la ejecución de una serie de movimientos de un robot) que suceden dentro del contexto de la operación del sistema.
- Roles (gerente, ingeniero, vendedor, etc.) que desempeñan las personas que interactúan con el sistema.
- Unidades organizacionales (división, grupo, equipo, etc.) que son relevantes para una aplicación.
- Lugares (piso de manufactura o plataforma de carga) que establecen el contexto del problema y la función general del sistema.
- Estructuras (sensores, vehículos de cuatro ruedas, computadoras, etc.) que definen una clase de objetos o clases relacionadas de éstos.

Esta clasificación sólo es una de muchas propuestas en la bibliografía. Por ejemplo, Budd (Budd, 1996) sugiere una taxonomía de clases que incluye productores (fuentes) y consumidores (sumideros) de datos, administradores de datos, vista, clases de observador y clases de auxiliares.

También es importante darse cuenta de lo que no son las clases u objetos. En general, una clase nunca debe tener un "nombre de procedimiento imperativo". Por ejemplo, si los desarrolladores del software de un sistema de imágenes médicas definieron un objeto con el nombre InvertirImagen o incluso InversióndeImagen, cometerían un error sutil. La Imagen obtenida del software podría ser, por supuesto, una clase (algo que es parte del dominio de

la información). La inversión de la imagen es una operación que se aplica al objeto. Es probable que la inversión esté definida como una operación para el objeto Imagen, pero no lo estaría como clase separada con la connotación "inversión de imagen". Como afirma Cashman (Cashman, 1989): "el intento de la orientación a objetos es contener, pero mantener separados, los datos y las operaciones sobre ellos". Para ilustrar cómo podrían definirse las clases del análisis durante las primeras etapas del modelado, considere un análisis gramatical (los sustantivos están subrayados, los verbos aparecen en cursivas) de una narración de procesamiento[22] para la función de seguridad de CasaSegura.

Seguridad de CasaSegura
La función de seguridad CasaSegura permite que el propietario configure el sistema de seguridad cuando se instala, vigila todos los sensores conectados al sistema de seguridad e interactúa con el propietario a través de internet, una PC o panel de control.

Durante la instalación, la PC de CasaSegura se utiliza para programar y configurar el sistema. Se asigna a cada sensor un número y tipo, se programa un password maestro para activar y desactivar el sistema y se introducen números telefónicos para marcar cuando ocurre un evento de sensor.

Cuando se reconoce un evento de sensor, el software invoca una alarma audible instalada en el sistema. Después de un tiempo de retraso que especifica el propietario durante las actividades de configuración del sistema, el software marca un número telefónico de un servicio de monitoreo, proporciona información acerca de la ubicación y reporta la naturaleza del evento detectado. El número telefónico se vuelve a marcar cada 20 segundos hasta que se obtiene la conexión telefónica.

El propietario recibe información de seguridad a través de un panel de control, la PC o un navegador, lo que en conjunto se llama interfaz. La interfaz despliega mensajes de aviso e información del estado del sistema en el panel de control, la PC o la ventana del navegador. La interacción del propietario tiene la siguiente forma.

[22] Una narración de procesamiento es similar al caso de uso en su estilo, pero algo distinto en su propósito. La narración de procesamiento hace una descripción general de la función que se va a desarrollar. No es un escenario escrito desde el punto de vista de un actor. No obstante, es importante observar que el análisis gramatical también puede emplearse para todo caso de uso desarrollado como parte de la obtención de requerimientos (indagación).

Con los sustantivos se proponen varias clases potenciales:

Clase potencial	Clasificación general
propietario	rol de entidad externa
sensor	entidad externa
panel de control	entidad externa
instalación	ocurrencia
sistema (alias sistema de seguridad)	cosa
número, tipo	no objetos, atributos de sensor
password maestro	cosa
número telefónico	cosa
evento de sensor	ocurrencia
alarma audible	entidad externa
servicio de monitoreo	unidad organizacional o entidad externa

Figura N°3-10 - Clases potenciales

La lista continuará hasta que se hayan considerado todos los sustantivos en la narrativa de procesamiento. Observe que cada entrada en la lista se llama objeto potencial. El lector debe considerar cada una antes de tomar la decisión final.

Coad y Yourdon (Coad & Yourdon, 1991) sugieren seis características de selección que deben usarse cuando se considere cada clase potencial para incluirla en el modelo de análisis:

1. Información retenida. La clase potencial será útil durante el análisis sólo si debe recordarse la información sobre ella para que el sistema pueda funcionar.
2. Servicios necesarios. La clase potencial debe tener un conjunto de operaciones identificables que cambien en cierta manera el valor de sus atributos.
3. Atributos múltiples. Durante el análisis de los requerimientos, la atención debe estar en la información "principal"; en realidad, una clase con un solo atributo puede ser útil durante el diseño, pero es probable que durante la actividad de análisis se represente mejor como un atributo de otra clase.
4. Atributos comunes. Para la clase potencial se define un conjunto de atributos y se aplican éstos a todas las instancias de la clase.
5. Operaciones comunes. Se define un conjunto de operaciones para la clase potencial y éstas se aplican a todas las instancias de la clase.

6. Requerimientos esenciales. Las entidades externas que aparezcan en el espacio del problema y que produzcan o consuman información esencial para la operación de cualquier solución para el sistema casi siempre se definirán como clases en el modelo de requerimientos.

Para que se considere una clase legítima para su inclusión en el modelo de requerimientos, un objeto potencial debe satisfacer todas (o casi todas) las características anteriores. La decisión de incluir clases potenciales en el modelo de análisis es algo subjetiva, y una evaluación posterior tal vez haga que un objeto se descarte o se incluya de nuevo. Sin embargo, el primer paso del modelado basado en clases es la definición de éstas, y deben tomarse las medidas respectivas (aun las subjetivas). Con esto en mente, se aplicarán las características de selección a la lista de clases potenciales de CasaSegura:

Clase potencial	**Número de característica que se aplica**
propietario	rechazada: 1 y 2 fallan, aunque la 6 aplica
sensor	aceptada: se aplican todas
panel de control	aceptada: se aplican todas
instalación	rechazada
sistema (alias sistema de seguridad)	aceptada: se aplican todas
número, tipo	rechazada: 3 fallan, atributos de sensores
password maestro	rechazada: 3 falla
número telefónico	rechazada: 3 falla
evento de sensor	aceptada: se aplican todas
alarma audible	aceptada: se aplican 2, 3, 4, 5 y 6
servicio de monitoreo	rechazada: 1 y 2 fallan aunque la 6 aplica

Debe notarse que: 1) la lista anterior no es exhaustiva; para completar el modelo tendrían que agregarse clases adicionales; 2) algunas de las clases potenciales rechazadas se convertirán en atributos para otras que sí fueron aceptadas (por ejemplo, número y tipo son atributos de Sensor, y password maestro y número telefónico pueden convertirse en atributos de Sistema); 3) diferentes enunciados del problema harían que se tomaran decisiones distintas para "aceptar o rechazar" (por ejemplo, si cada propietario tuviera una clave individual o se identificara con reconocimiento de voz, la clase Propietario satisfaría las características 1 y 2, y se aceptaría).

Especificación de atributos

Los atributos describen una clase que ha sido seleccionada para incluirse en el modelo de requerimientos. En esencia, son los atributos los que definen la clase (esto aclara lo que significa la clase en el contexto del espacio del problema). Por ejemplo, si se fuera a construir un sistema que analiza estadísticas de jugadores de béisbol profesionales, los atributos de la clase Jugador serían muy distintos de los que tendría la misma clase cuando se usara en el contexto del sistema de pensiones de dicho deporte. En la primera, atributos tales como nombre, porcentaje de bateo, porcentaje de fildeo, años jugados y juegos jugados serían relevantes. Para la segunda, algunos de los anteriores sí serían significativos, pero otros se sustituirían (o se crearían) por atributos tales como salario promedio, crédito hacia el retiro completo, opciones del plan de pensiones elegido, dirección de correo, etcétera.

Para desarrollar un conjunto de atributos significativos para una clase de análisis, debe estudiarse cada caso de uso y seleccionar aquellas "cosas" que "pertenezcan" razonablemente a la clase. Además, debe responderse la pregunta siguiente para cada clase: "¿qué aspectos de los datos (compuestos o elementales) definen por completo esta clase en el contexto del problema en cuestión?".

Para ilustrarlo, se considera la clase Sistema definida para CasaSegura. El propietario configura la función de seguridad para que refleje la información de los sensores, la respuesta de la alarma, la activación o desactivación, la identificación, etc. Estos datos compuestos se representan del modo siguiente:

información de identificación = identificación del sistema + número telefónico de verificación + estado del sistema
información de respuesta de la alarma = tiempo de retraso + número telefónico
información de activación o desactivación = password maestro + número de intentos permisibles + password temporal

Cada uno de los datos a la derecha del signo igual podría definirse más, hasta un nivel elemental, pero para nuestros propósitos constituye una lista razonable de atributos para la clase Sistema (parte sombreada de la figura 3-11). Los sensores forman parte del sistema general CasaSegura, pero no están enlistados como datos o atributos en la figura 3-11. Sensor ya se definió como clase, y se asociarán múltiples objetos Sensor con la clase Sistema. En general, se evita definir algo como atributo si más de uno va a asociarse con la clase.

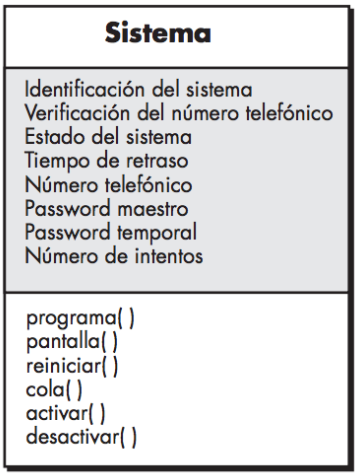


Figura N°3-11 - Diagrama de clase para la clase sistema

Hasta no hacer más investigaciones, es probable que la operación programar( ) se divida en cierto número de suboperaciones específicas adicionales que se requieren para configurar el sistema. Por ejemplo, programar() implica la especificación de números telefónicos, la configuración de las características del sistema (por ejemplo, crear la tabla de sensores, introducir las características de la alarma, etc.) y la introducción de la(s) clave(s). Pero, de momento, programar() se especifica como una sola operación.

CASASEGURA

Modelos de clase

La escena: Cubículo de Ed, cuando comienza el modelado de los requerimientos.

Participantes: Jamie, Vinod y Ed, todos ellos miembros del equipo de ingeniería de software para *CasaSegura*.

La conversación:

[Ed ha estado trabajando para obtener las clases a partir del formato del caso de uso para AVC-MVC (presentado en un recuadro anterior de este capítulo) y expone a sus colegas las que ha obtenido].

Ed: Entonces, cuando el propietario quiere escoger una cámara, la tiene que elegir del plano. Definí una clase llamada **Plano**. Éste es el diagrama.
(Observan la figura 6.10.)

Jamie: Entonces, **Plano** es un objeto que agrupa paredes, puertas, ventanas y cámaras. Eso significa esas líneas con leyendas, ¿verdad?

Ed: Sí, se llaman "asociaciones". Una clase se asocia con otra de acuerdo con las asociaciones que se ven (las asociaciones se estudian en la sección 6.5.5).

Vinod: Es decir, el plano real está constituido por paredes que contienen en su interior cámaras y sensores. ¿Cómo sabe el plano dónde colocar estos objetos?

Ed: No lo sabe, pero las otras clases sí. Mira los atributos de, digamos, **SegmentodePared**, que se usa para construir una pared. El segmento de muro tiene coordenadas de inicio y final, y la operación *draw( )* hace el resto.

Jamie: Y lo mismo vale para las ventanas y puertas. Parece como si cámara tuviera algunos atributos adicionales.

Ed: Sí. Los necesito para dar información del alcance y el acercamiento.

Vinod: Tengo una pregunta. ¿Por qué tiene la cámara una identificación pero las demás no? Veo que tienes un atributo llamado **ParedSiguiente**. ¿Cómo sabe **SegmentodePared** cuál será la pared siguiente?

Ed: Buena pregunta, pero, como dijimos, ésa es una decisión de diseño, por lo que la voy a retrasar hasta...

Jamie: Momento... Apuesto a que ya lo has imaginado.

Ed (sonríe con timidez): Es cierto, voy a usar una estructura de lista que modelaré cuando vayamos a diseñar. Si somos puristas en cuanto a separar el análisis y el diseño, el nivel de detalle podría parecer sospechoso.

Jamie: Me parece muy bien, pero tengo más preguntas.

Figura N°3-12 - Modelos de clase

Además del análisis gramatical, se obtiene más perspectiva sobre otras operaciones si se considera la comunicación que ocurre entre los objetos. Éstos se comunican con la transmisión de mensajes entre sí. Antes de continuar con la especificación de operaciones, se estudiará esto con más detalle.

3.1.6 Modelado clase-responsabilidad-colaborador (CRC)

El modelado clase-responsabilidad-colaborador (CRC) proporciona una manera sencilla de identificación y organización de las clases que son relevantes para los requerimientos de un sistema o producto. Ambler (Ambler, 1995) describe el modelado CRC en la siguiente forma: Un modelo CRC en realidad es un conjunto de tarjetas índice estándar que representan clases. Las tarjetas se dividen en tres secciones. En la parte superior de la tarjeta se escribe el nombre de la clase, en la parte izquierda del cuerpo se enlistan las responsabilidades de la clase y en la derecha, los colaboradores.

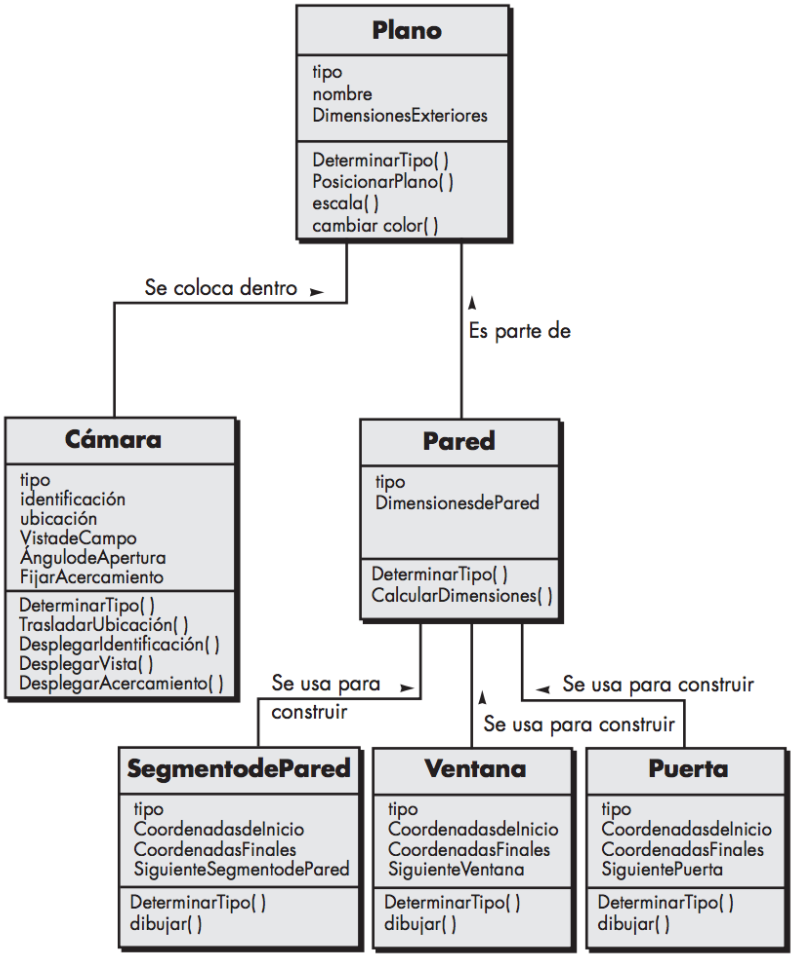


Figura N°3-13 - Diagrama de clase para plano

En realidad, el modelo CRC hace uso de tarjetas índice reales o virtuales. El objetivo es desarrollar una representación organizada de las clases. Las responsabilidades son los atributos y operaciones relevantes para la clase. En pocas palabras, una responsabilidad es "cualquier cosa que la clase sepa o haga" (Ambler, 1995). Los colaboradores son aquellas clases que se requieren para dar a una clase la información necesaria a fin de completar una responsabilidad. En general, una colaboración implica una solicitud de información o de cierta acción.

En la figura 3-14 se ilustra una tarjeta CRC índice sencilla para la clase Plano: la lista de responsabilidades en la tarjeta CRC es preliminar y está sujeta a agregados o modificaciones. Las clases Pared y Cámara se anotan frente a la responsabilidad que requerirá su colaboración.

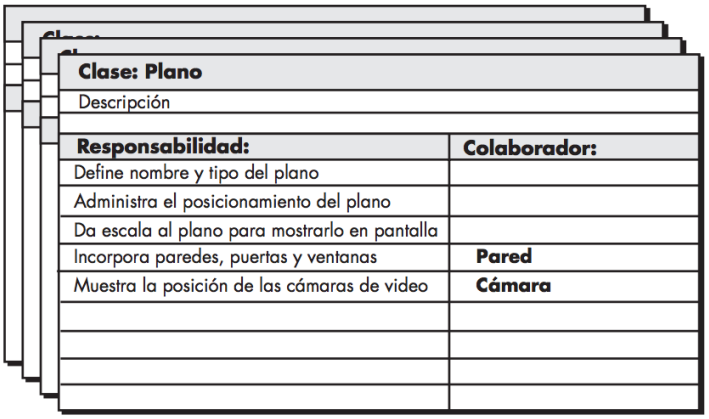


Figura N°3-14 - Modelo de tarjeta CRC índice

Clases. Al inicio de este capítulo se presentaron lineamientos básicos para identificar clases y objetos. La taxonomía de tipos de clase presentada en la sección anterior puede ampliarse con las siguientes categorías:

- Clases de entidad, también llamadas clases modelo o de negocio, se extraen directamente del enunciado del problema (por ejemplo, Plano y Sensor). Es común que estas clases representen cosas almacenadas en una base de datos y que persistan mientras dure la aplicación (a menos que se eliminen en forma específica).
- Clases de frontera se utilizan para crear la interfaz (por ejemplo, pantallas interactivas o reportes impresos) que el usuario mira y con la que interactúa cuando utiliza el

software. Los objetos de entidad contienen información que es importante para los usuarios, pero no se muestran por sí mismos. Las clases de frontera se diseñan con la responsabilidad de administrar la forma en la que se presentan a los usuarios los objetos de entidad. Por ejemplo, una clase de frontera llamada VentanadeCámara tendría la responsabilidad de desplegar la salida de una cámara de vigilancia para el sistema CasaSegura.

- Clases de controlador administran una "unidad de trabajo" de principio a fin. Es decir, las clases de controlador están diseñadas para administrar 1) la creación o actualización de objetos de entidad, 2) las instancias de los objetos de frontera en tanto obtienen información de los objetos de entidad, 3) la comunicación compleja entre conjuntos de objetos y 4) la validación de datos comunicados entre objetos o entre el usuario y la aplicación. En general, las clases de controlador no se consideran hasta haber comenzado la actividad de diseño.

Responsabilidades

En las secciones anteriores se presentaron los lineamientos básicos para identificar responsabilidades (atributos y operaciones). Wirfs-Brock sugiere cinco lineamientos para asignar responsabilidades a las clases:

1. La inteligencia del sistema debe estar distribuida entre las clases para enfrentar mejor las necesidades del problema. Toda aplicación contiene cierto grado de inteligencia, es decir, lo que el sistema sabe y lo que puede hacer. Esta inteligencia se distribuye entre las clases de diferentes maneras. Las clases "tontas" (aquellas que tienen pocas responsabilidades) pueden modelarse para que actúen como subordinadas de ciertas clases "inteligentes" (las que tienen muchas responsabilidades). Aunque este enfoque hace directo el flujo del control en un sistema, tiene algunas desventajas: concentra toda la inteligencia en pocas clases, lo que hace que sea más difícil hacer cambios, y tiende a que se requieran más clases y por ello más trabajo de desarrollo.

   Si la inteligencia del sistema tiene una distribución más pareja entre las clases de una aplicación, cada objeto sabe algo, sólo hace unas cuantas cosas (que por lo general están bien identificadas) y la cohesión del sistema mejora. Esto facilita el mantenimiento del software y reduce el efecto de los resultados colaterales del cambio.

Para determinar si la inteligencia del sistema está distribuida en forma apropiada, deben evaluarse las responsabilidades anotadas en cada modelo de tarjeta CRC índice a fin de definir si alguna clase tiene una lista demasiado larga de responsabilidades. Esto indica una concentración de inteligencia. Además, las responsabilidades de cada clase deben tener el mismo nivel de abstracción. Por ejemplo, entre las operaciones enlistadas para una clase agregada llamada RevisarCuenta, un revisor anota dos responsabilidades: hacer el balance de la cuenta y eliminar comprobaciones concluidas. La primera operación (responsabilidad) implica un procedimiento matemático complejo y lógico. La segunda es una simple actividad de oficina. Como estas dos operaciones no están en el mismo nivel de abstracción, eliminar comprobaciones concluidas debe colocarse dentro de las responsabilidades de RevisarEntrada, clase que está incluida en la clase agregada RevisarCuenta.

2. Cada responsabilidad debe enunciarse del modo más general posible. Este lineamiento implica que las responsabilidades generales (tanto atributos como operaciones) deben residir en un nivel elevado de la jerarquía de clases (porque son generales y se aplicarán a todas las subclases).
3. La información y el comportamiento relacionado con ella deben residir dentro de la misma clase. Esto logra el principio orientado a objetos llamado encapsulamiento. Los datos y los procesos que los manipulan deben empacarse como una unidad cohesiva.
4. La información sobre una cosa debe localizarse con una sola clase, y no distribuirse a través de muchas. Una sola clase debe tener la responsabilidad de almacenar y manipular un tipo específico de información. En general, esta responsabilidad no debe ser compartida por varias clases. Si la información está distribuida, es más difícil dar mantenimiento al software y más complicado someterlo a prueba.
5. Cuando sea apropiado, las responsabilidades deben compartirse entre clases relacionadas. Hay muchos casos en los que varios objetos relacionados deben tener el mismo comportamiento al mismo tiempo. Por ejemplo, considere un juego de video que deba tener en la pantalla las clases siguientes: Jugador, CuerpodelJugador, BrazosdelJugador, PiernasdelJugador y CabezadelJugador. Cada una de estas clases tiene sus propios atributos (como posición, orientación, color y velocidad) y todas deben actualizarse y desplegarse a medida que el usuario manipula una palanca de juego. Las responsabilidades actualizar() y desplegar()

deben, por tanto, ser compartidas por cada uno de los objetos mencionados. Jugador sabe cuando algo ha cambiado y requiere actualizarse(). Colabora con los demás objetos para obtener una nueva posición u orientación, pero cada objeto controla su propio despliegue en la pantalla.

Colaboraciones

Una clase cumple sus responsabilidades en una de dos formas: 1) usa sus propias operaciones para manipular sus propios atributos, con lo que satisface una responsabilidad particular o 2) colabora con otras clases. Wirfs-Brock define las colaboraciones del modo siguiente:

Las colaboraciones representan solicitudes que hace un cliente a un servidor para cumplir con sus responsabilidades. Una colaboración es la materialización del contrato entre el cliente y el servidor. Decimos que un objeto colabora con otro si, para cumplir una responsabilidad, necesita enviar al otro objeto cualesquiera mensajes. Una sola colaboración fluye en una dirección: representa una solicitud del cliente al servidor. Desde el punto de vista del cliente, cada una de sus colaboraciones está asociada con una responsabilidad particular implementada por el servidor.

Las colaboraciones se identifican determinando si una clase puede cumplir cada responsabilidad. Si no es así, entonces necesita interactuar con otra clase. Ésa es una colaboración.

Como ejemplo, considere la función de seguridad de CasaSegura. Como parte del procedimiento de activación, el objeto PaneldeControl debe determinar si están abiertos algunos sensores. Se define una responsabilidad llamada determinar-estado-delsensor( ). Si los sensores están abiertos, PaneldeControl debe fijar el atributo estado como “no está listo”. La información del sensor se adquiere de cada objeto Sensor. Por tanto, la responsabilidad determinar-estadodelsensor( ) se cumple sólo si PaneldeControl trabaja en colaboración con Sensor.

Para ayudar a identificar a los colaboradores, se estudian tres relaciones generales diferentes entre las clases: 1) la relación es-parte-de, 2) la relación tiene-conocimiento-de y 3) la relación depende-de. En los párrafos siguientes se analizan brevemente cada una de

estas tres responsabilidades generales. Todas las clases que forman parte de una clase agregada se conectan a ésta por medio de una relación es-parte-de. Considere las clases definidas por el juego mencionado antes, la clase CuerpodelJugador es-parte-de Jugador, igual que BrazosdelJugador, PiernasdelJugador y CabezadelJugador. En UML, estas relaciones se representan como el agregado que se ilustra en la figura 3-15.

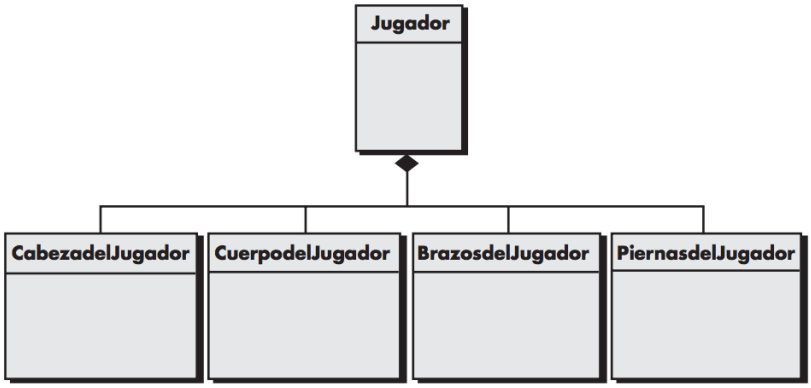


Figura N°3-15 - Una clase agregada compuesta

Cuando una clase debe adquirir información de otra, se establece la relación tiene-conocimiento-de. La responsabilidad determinar-estado-delsensor() ya mencionada es un ejemplo de ello.

La relación depende-de significa que dos clases tienen una dependencia que no se determina por tiene-conocimiento-de ni por es-parte-de. Por ejemplo, CabezadelJugador siempre debe estar conectada a CuerpodelJugador (a menos que el juego de video sea particularmente violento), pero cada objeto puede existir sin el conocimiento directo del otro. Un atributo del objeto CabezadelJugador, llamado posición-central, se determina a partir de la posición central de CuerpodelJugador. Esta información se obtiene por medio de un tercer objeto, Jugador, que la obtiene de CuerpodelJugador. Entonces, CabezadelJugador depende-de CuerpodelJugador.

En todos los casos, el nombre de la clase colaboradora se registra en el modelo de tarjeta CRC índice, junto a la responsabilidad que produce la colaboración. Por tanto, la tarjeta índice contiene una lista de responsabilidades y las colaboraciones correspondientes que hacen que se cumplan.

Cuando se ha desarrollado un modelo CRC completo, los participantes lo revisan con el empleo del enfoque siguiente:

1. Se da a todos los participantes que intervienen en la revisión (del modelo CRC) un subconjunto del modelo de tarjetas índice CRC. Deben separarse aquellas que colaboran (de modo que ningún revisor deba tener dos tarjetas que colaboren).
2. Todos los escenarios de casos de uso (y los diagramas correspondientes) deben organizarse en dos categorías.
3. El líder de la revisión lee el caso de uso en forma deliberada. Cuando llega a un objeto con nombre, entrega una ficha a la persona que tenga la tarjeta índice de la clase correspondiente. Por ejemplo, un caso de uso de CasaSegura contiene la narración siguiente:

   El propietario observa el panel de control de CasaSegura para determinar si el sistema está listo para recibir una entrada. Si el sistema no está listo, el propietario debe cerrar físicamente las puertas y ventanas de modo que el indicador listo aparezca [un indicador no está listo implica que un sensor se encuentra abierto, es decir, que una puerta o ventana está abierta].

   Cuando en la narración del caso de uso el líder de la revisión llega a "panel de control", entrega la ficha a la persona que tiene la tarjeta índice PaneldeControl. La frase "implica que un sensor está abierto" requiere que la tarjeta índice contenga una responsabilidad que validará esta implicación (esto lo logra la responsabilidad determinar-estado-delsensor(). Junto a la responsabilidad, en la tarjeta índice se encuentra el Sensor colaborador. Entonces, la ficha pasa al objeto Sensor.
4. Cuando se pasa la ficha, se pide al poseedor de la tarjeta Sensor que describa las responsabilidades anotadas en la tarjeta. El grupo determina si una (o más) de las responsabilidades satisfacen el requerimiento del caso de uso.
5. Si las responsabilidades y colaboraciones anotadas en las tarjetas índice no se acomodan al caso de uso, éstas se modifican. Lo anterior tal vez incluya la definición de nuevas clases (y las tarjetas CRC índice correspondientes) o la especificación en las tarjetas existentes de responsabilidades o colaboraciones nuevas o revisadas.

Este modo de operar continúa hasta terminar el caso de uso. Cuando se han revisado todos los casos de uso, continúa el modelado de los requerimientos.

Asociaciones y dependencias

En muchos casos, dos clases de análisis se relacionan de cierto modo con otra, en forma muy parecida a como dos objetos de datos se relacionan entre sí. En UML, estas relaciones se llaman asociaciones. Al consultar la figura 3-12, la clase Plano se define con la identificación de un conjunto de asociaciones entre Plano y otras dos clases, Cámara y Pared. La clase Pared está asociada con tres clases que permiten que se construya ésta, y que son SegmentodePared, Ventana y Puerta. En ciertos casos, una asociación puede definirse con más detalle si se indica multiplicidad. Un objeto Pared se construye a partir de uno o más objetos SegmentodePared. Además, el objeto Pared puede contener 0 o más objetos Ventana y 0 o más objetos Puerta. Estas restricciones de multiplicidad se ilustran en la figura 3-16, donde "uno o más" se representa con 1...*, y para "0 o más" se usa 0...*. En LMU, el asterisco indica una frontera ilimitada en ese rango[23].

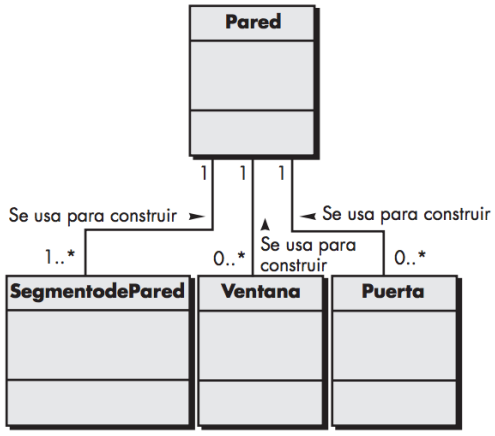


Figura N°3-16 - Multiplicidad

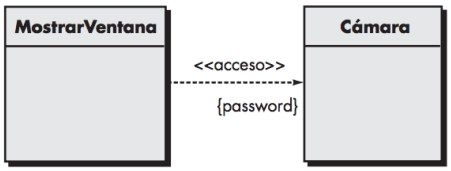


Figura N°3-17 - Dependencias

Sucede con frecuencia que entre dos clases de análisis existe una relación cliente-servidor. En tales casos, una clase cliente depende de algún modo de la clase servidor, y se establece una relación de dependencia. Las dependencias están definidas por un estereotipo. Un estereotipo es un "mecanismo extensible" [Arl02] dentro del UML que permite definir un elemento especial de modelado con semántica y especialización determinadas. En UML, los estereotipos se representan entre paréntesis dobles angulares (por ejemplo, <>).

[23] Como parte de una asociación, pueden indicarse otras relaciones de multiplicidad: una a una, una a muchas, muchas a muchas, una a un rango específico con límites inferior y superior, y otras.

Como ilustración de una dependencia simple dentro del sistema de vigilancia CasaSegura, un objeto Cámara (la clase servidora, en este caso) proporciona una imagen a un objeto MostrarVentana (la clase cliente). La relación entre estos dos objetos no es una asociación simple sino de dependencia. En el caso de uso escrito para la vigilancia (que no se presenta aquí), debe darse una clave especial a fin de observar ubicaciones específicas de las cámaras. Una forma de lograr esto es hacer que Cámara pida un password y luego asegure el permiso a MostrarVentana para que presente el video. Esto se representa en la figura 3-17, donde <> implica que el uso de la salida de cámara se controla con una clave especial.

Paquetes de análisis

Una parte importante del modelado del análisis es la categorización. Es decir, se clasifican distintos elementos del modelo de análisis (por ejemplo, casos de uso y clases de análisis) de manera que se agrupen en un paquete -llamado paquete de análisis- al que se da un nombre representativo.

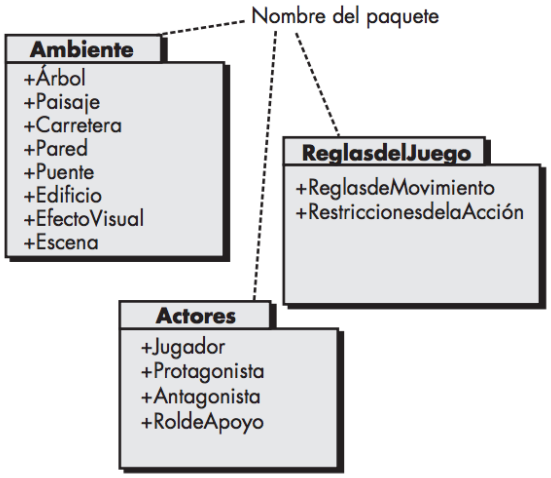


Figura N°3-18 - Paquetes

Para ilustrar el uso de los paquetes de análisis, considere el juego de video que se mencionó antes. A medida que se desarrolla el modelo de análisis para el juego de video, se obtiene un gran número de clases. Algunas se centran en el ambiente del juego -las escenas visuales que el usuario ve cuando lo usa-. En esta categoría quedan clases tales como Árbol, Paisaje, Carretera, Pared, Puente, Edificio y EfectoVisual. Otras se centran en los caracteres dentro del juego y describen sus características físicas, acciones y restricciones. Pueden definirse clases como Jugador (ya descrita), Protagonista, Antagonista y

RolesdeApoyo. Otras más describen las reglas del juego -cómo se desplaza un jugador por el ambiente-. Candidatas para esto son clases como ReglasdeMovimiento y RestriccionesdelaAcción. Pueden existir muchas otras categorías. Estas clases se agrupan en los paquetes de análisis que se observan en la figura 3-18.

El signo más (suma) que precede al nombre de la clase de análisis en cada paquete, indica que las clases tienen visibilidad pública, por lo que son accesibles desde otros paquetes. Aunque no se aprecia en la figura, hay otros símbolos que preceden a un elemento dentro de un paquete. El signo menos (resta) indica que un elemento queda oculto desde todos los demás paquetes. Y el símbolo # señala que un elemento es accesible sólo para los paquetes contenidos dentro de un paquete dado.

3.1.7 Conclusiones

El objetivo del modelado de los requerimientos es crear varias representaciones que describan lo que necesita el cliente, establecer una base para generar un diseño de software y definir un conjunto de requerimientos que puedan ser validados una vez construido el software. El modelo de requerimientos cruza la brecha entre la representación del sistema que describe el sistema en su conjunto y la funcionalidad del negocio, y un diseño de software que describe la arquitectura de la aplicación del software, la interfaz de usuario y la estructura de componentes.

Los modelos basados en el escenario ilustran los requerimientos del software desde el punto de vista del usuario. El caso de uso -descripción, hecha con una narración o un formato, de una interacción entre un actor y el software- es el principal elemento del modelado. El caso de uso se obtiene durante la indagación de los requerimientos y define las etapas clave de una función o interacción específica. El grado de formalidad del caso de uso y su nivel de detalle varía, pero el resultado final da las entradas necesarias a todas las demás actividades del modelado. Los escenarios también pueden ser descritos con el uso de un diagrama de actividades -representación gráfica parecida a un diagrama de flujo que ilustra el flujo del procesamiento dentro de un escenario específico-. Los diagramas de canal (swimlane) ilustran la forma en la que se asigna el flujo del procesamiento a distintos actores o clases.

El modelado de datos se utiliza para describir el espacio de información que será construido o manipulado por el software. El modelado de datos comienza con la representación de los objetos de datos -información compuesta que debe ser entendida por el software-. Se identifican los atributos de cada objeto de datos y se describen las relaciones entre estos objetos.

El modelado basado en clases utiliza información obtenida de los elementos del modelado basado en el escenario y en datos, para identificar las clases de análisis. Se emplea un análisis gramatical para obtener candidatas a clase, atributos y operaciones, a partir de narraciones basadas en texto. Se definen criterios para definir una clase. Para definir las relaciones entre clases, se emplean tarjetas índice clase-responsabilidad-colaborador. Además, se aplican varios elementos de la notación UML para definir jerarquías, relaciones, asociaciones, agregaciones y dependencias entre clases. Se emplean paquetes de análisis para clasificar y agrupar clases, de manera que sean más manejables en sistemas grandes.

3.2 DISEÑO EXPERIMENTAL

Para ciertos tipos de software, el caso de uso puede ser la única representación para modelar los requerimientos que se necesite. Para otros, se escoge un enfoque orientado a objetos y se desarrollan modelos basados en clase. Pero en otras situaciones, los requerimientos de las aplicaciones complejas demandan el estudio de la manera como se transforman los objetos de datos cuando se mueven a través del sistema; cómo se comporta una aplicación a consecuencia de eventos externos; si el conocimiento del dominio existente puede adaptarse al problema en cuestión; o, en el caso de sistemas y aplicaciones basados en web, cómo unificar el contenido y la funcionalidad para dar al usuario final la capacidad de navegar con éxito por una webapp a fin de lograr sus objetivos.

3.2.1 Requerimientos que modelan las estrategias

Un punto de vista del modelado de los requerimientos, llamada análisis estructurado, considera como entidades separadas los datos y los procesos que los transforman. Los objetos de datos se modelan en una forma que define sus atributos y relaciones. Los procesos que manipulan objetos de datos se modelan de una forma que muestra cómo transforman los datos cuando los objetos de datos fluyen por el sistema. Un segundo enfoque del modelado de análisis, llamado análisis orientado a objetos, se centra en la definición de clases y en el modo en el que colaboran una con otra para cumplir con los requerimientos del cliente.

Aunque el modelo de análisis que se propone en este libro combina características de ambos enfoques, es frecuente que los equipos del software elijan uno de ellos y excluyan las representaciones del otro. La pregunta no es cuál es mejor, sino qué combinación de representaciones dará a los participantes el mejor modelo de los requerimientos del software y cuál será el mejor puente para cruzar la brecha hacia el diseño del software.

3.2.2 Modelado orientado al flujo

Aunque algunos ingenieros de software perciben el modelado orientado al flujo como una técnica obsoleta, sigue siendo una de las notaciones más usadas actualmente para hacer el análisis de los requerimientos[24]. Si bien el diagrama de flujo de datos (DFD) y la información relacionada no son una parte formal del UML, se utilizan para complementar los diagramas de éste y amplían la perspectiva de los requerimientos y del flujo del sistema. El DFD adopta

[24] El modelado del flujo de datos es una actividad fundamental del análisis estructurado.

un punto de vista del tipo entrada-proceso-salida para el sistema. Es decir, los objetos de datos entran al sistema, son transformados por elementos de procesamiento y los objetos de datos que resultan de ello salen del software. Los objetos de datos se representan con flechas con leyendas y las transformaciones, con círculos (también llamados burbujas).

El DFD se presenta en forma jerárquica. Es decir, el primer modelo de flujo de datos (en ocasiones llamado DFD de nivel 0 o diagrama de contexto) representa al sistema como un todo. Los diagramas posteriores de flujo de datos mejoran el diagrama de contexto y dan cada vez más detalles en los niveles siguientes.

Creación de un modelo de flujo de datos

El diagrama de flujo de datos permite desarrollar modelos del dominio de la información y del dominio funcional. A medida que el DFD se mejora con mayores niveles de detalle, se efectúa la descomposición funcional implícita del sistema. Al mismo tiempo, la mejora del DFD da como resultado el refinamiento de los datos conforme avanzan por los procesos que constituyen la aplicación. Unos cuantos lineamientos sencillos ayudan muchísimo durante la elaboración del diagrama de flujo de los datos: 1) el nivel 0 del diagrama debe ilustrar el software o sistema como una sola burbuja; 2) debe anotarse con cuidado las entradas y salidas principales; 3) la mejora debe comenzar por aislar procesos candidatos, objetos de datos y almacenamiento de éstos, para representarlos en el siguiente nivel; 4) todas las flechas y burbujas deben etiquetarse con nombres significativos; 5) de un nivel a otro, debe mantenerse la continuidad del flujo de información[25], y 6) debe mejorarse una burbuja a la vez. Existe la tendencia natural a complicar innecesariamente el diagrama de flujo de los datos. Esto sucede cuando se trata de ilustrar demasiados detalles en una etapa muy temprana o representar aspectos de procedimiento del software en lugar del flujo de la información.

Para ilustrar el uso del DFD y la notación relacionada, consideremos de nuevo la función de seguridad de CasaSegura. En la figura 30 se muestra un DFD de nivel 0 para dicha función. Las entidades externas principales (cuadrados) producen información para uso del sistema y consumen información generada por éste. Las flechas con leyendas representan objetos de datos o jerarquías de éstos. Por ejemplo, los comandos y datos del usuario agrupan todos los comandos de configuración, todos los comandos de activación/desactivación, todas las

[25] Es decir, los objetos de datos que entran al sistema o a cualquier transformación en cierto nivel deben ser los mismos objetos de datos (o sus partes constitutivas) que entran a la transformación en un nivel mejorado.

diferentes interacciones y todos los datos que se introducen para calificar o expandir un comando.

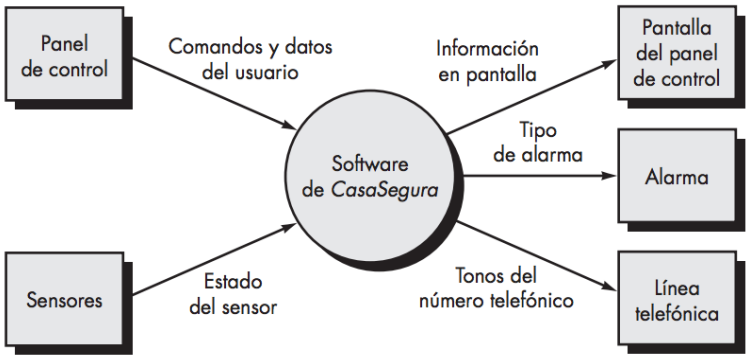


Figura N°3-19 - DFD en el nivel de contexto para la función de seguridad de CasaSegura

Ahora debe expandirse el DFD de nivel 0 a un modelo de flujo de datos de nivel 1. Pero, ¿cómo hacerlo? Según el enfoque sugerido en la sección anterior, debe aplicarse un "análisis gramatical" a la narración del caso de uso que describe la burbuja en el nivel del contexto. Es decir, se aíslan todos los sustantivos (y frases sustantivadas) y verbos (y frases verbales) en la narración del procesamiento de CasaSegura obtenida durante la primera reunión realizada para recabar los requerimientos. Recordemos el análisis gramatical del texto que narra el procesamiento presentado anteriormente:

La función de seguridad CasaSegura permite que el propietario configure el sistema de seguridad cuando se instala, vigila todos los sensores conectados al sistema de seguridad e interactúa con el propietario a través de internet, una PC o un panel de control.

Durante la instalación, la PC de CasaSegura se utiliza para programar y configurar el sistema. Se asigna a cada sensor un número y un tipo, se programa una clave maestra para activar y desactivar el sistema, y se introducen números telefónicos para marcar cuando ocurre un evento de sensor.

Cuando se reconoce un evento de sensor, el software invoca una alarma audible instalada en el sistema. Después de un tiempo de retraso que especifica el propietario durante las actividades de configuración del sistema, el software marca un número telefónico de un servicio de monitoreo, proporciona información acerca de la ubicación y reporta la naturaleza

del evento detectado. El número telefónico vuelve a marcarse cada 20 segundos hasta que se obtiene la conexión telefónica.

El propietario recibe información de seguridad a través de un panel de control, la PC o un navegador, lo que en conjunto se llama interfaz. La interfaz despliega en el panel de control, en la PC o en la ventana del navegador mensajes de aviso e información del estado del sistema. La interacción del propietario tiene la siguiente forma…

En relación con el análisis gramatical, los verbos son los procesos de CasaSegura y se representarán como burbujas en un DFD posterior. Los sustantivos son entidades externas (cuadros), datos u objetos de control (flechas) o almacenamiento de datos (líneas dobles). De lo estudiado en la sección anterior se recuerda que los sustantivos y verbos se asocian entre sí (por ejemplo, a cada sensor se asigna un número y tipo; entonces, número y tipo son atributos del objeto de datos sensor). De modo que al realizar un análisis gramatical de la narración de procesamiento en cualquier nivel del DFD, se genera mucha información útil sobre la manera de proceder para la mejora del nivel siguiente. En la figura 31 se presenta un DFD de nivel 1 con el empleo de esta información. El proceso en el nivel de contexto que se ilustra en la figura 30 ha sido expandido a seis procesos derivados del estudio del análisis gramatical. De manera similar, el flujo de información entre procesos del nivel 1 ha surgido de dicho análisis. Además, entre los niveles 0 y 1 se mantiene la continuidad del flujo de información.

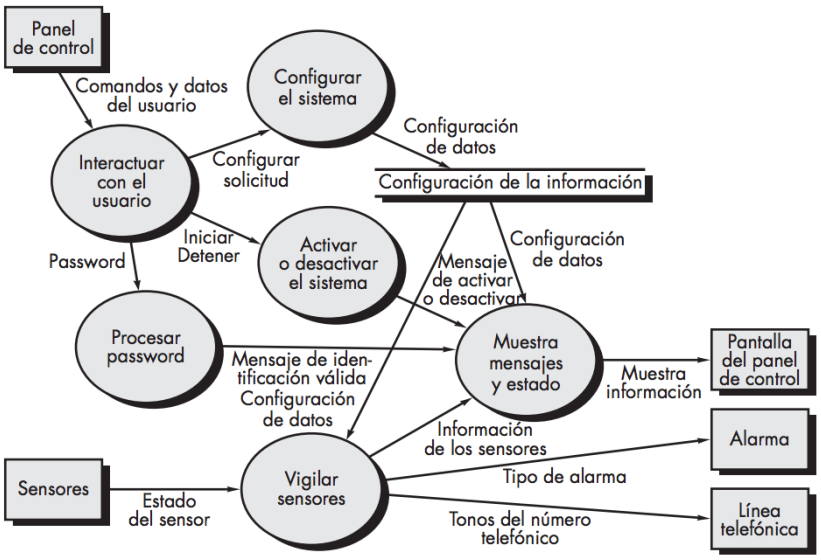


Figura N°3-20 - DFD de nivel 1 para la función de seguridad de CasaSegura

Los procesos representados en el nivel 1 del DFD pueden mejorarse más hacia niveles inferiores. Por ejemplo, el proceso vigilar sensores se mejora en el DFD de nivel 2, como se aprecia en la figura 3-21. De nuevo, observe que entre los niveles se ha mantenido la continuidad del flujo de información.

La mejoría de los DFD continúa hasta que cada burbuja realiza una función simple. Es decir, hasta que el proceso representado por la burbuja ejecuta una función que se implementaría fácilmente como componente de un programa.

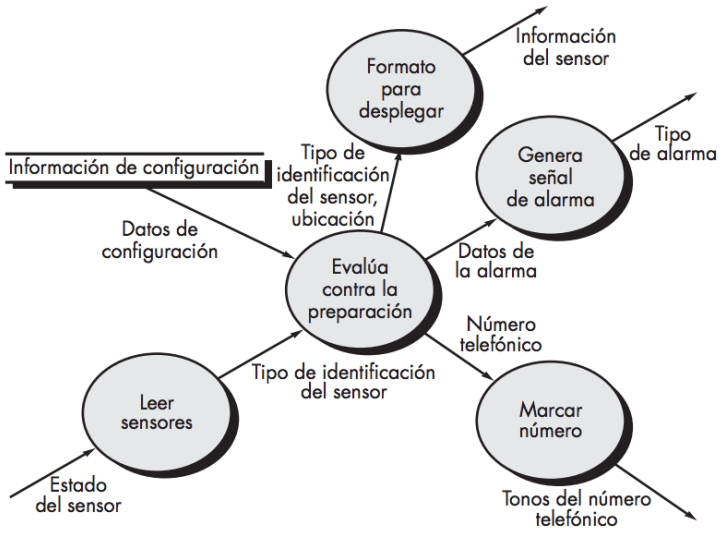


Figura N°3-21 - DFD de nivel 2 que mejora el proceso vigilar sensores

Creación de un modelo de flujo de control

Para ciertos tipos de aplicaciones, el modelo de datos y el diagrama de flujo de datos es todo lo que se necesita para obtener una visión significativa de los requerimientos del software. Sin embargo, como ya se dijo, un gran número de aplicaciones son "motivadas" por eventos y no por datos, producen información de control en lugar de reportes o pantallas, y procesan información con mucha atención en el tiempo y el desempeño. Tales aplicaciones requieren el uso del modelado del flujo de control, además de modelar el flujo de datos.

Se dijo que un evento o aspecto del control se implementa como valor booleano (por ejemplo, verdadero o falso, encendido o apagado, 1 o 0) o como una lista discreta de condiciones (vacío, bloqueado, lleno, etc.). Se sugieren los lineamientos siguientes para seleccionar eventos candidatos potenciales:

- Enlistar todos los sensores que son “leídos” por el software.
- Enlistar todas las condiciones de interrupción.
- Enlistar todos los “interruptores” que son activados por un operador.
- Enlistar todas las condiciones de los datos.
- Revisar todos los “aspectos de control” como posibles entradas o salidas de especificación del control, según el análisis gramatical de sustantivos y verbos que se aplicó a la narración del procesamiento.
- Describir el comportamiento de un sistema con la identificación de sus estados, identificar cómo se llega a cada estado y definir las transiciones entre estados.
- Centrarse en las posibles omisiones, error muy común al especificar el control; por ejemplo, se debe preguntar: “¿hay otro modo de llegar a este estado o de salir de él?”

Entre los muchos eventos y aspectos del control que forman parte del software de CasaSegura, se encuentran evento de sensor (por ejemplo, un sensor se descompone), bandera de cambio (señal para que la pantalla cambie) e interruptor iniciar/detener (señal para encender o apagar el sistema).

La especificación de control

Una especificación de control (CSPEC) representa de dos maneras distintas el comportamiento del sistema (en el nivel desde el que se hizo referencia a él). La CSPEC contiene un diagrama de estado que es una especificación secuencial del comportamiento. También puede contener una tabla de activación del programa, especificación combinatoria del comportamiento. La figura 3-22 ilustra un diagrama de estado preliminar para el nivel 1 del modelo de flujo de control para CasaSegura. El diagrama indica cómo responde el sistema a eventos conforme pasa por los cuatro estados definidos en este nivel. Con la revisión del diagrama de estado se determina el comportamiento del sistema, y, lo que es más importante, se investiga si existen “agujeros” en el comportamiento especificado.

Por ejemplo, el diagrama de estado (véase la figura 3-22) indica que las transiciones del estado Ocioso ocurren si el sistema se reinicia, se activa o se apaga. Si el sistema se activa (por ejemplo, se enciende el sistema de alarma), ocurre una transición al estado VigilanciadelEstadodelSistema, los mensajes en la pantalla cambian como se muestra y se invoca el proceso SistemadeVigilanciayControl.

Fuera del estado SistemadeVigilanciayControl ocurren dos transiciones: 1) cuando se desactiva el sistema hay una transición de regreso al estado Ocioso; 2) cuando se dispara un sensor en el estado ActivarAlarma. Durante la revisión se consideran todas las transiciones y el contenido de todos los estados.

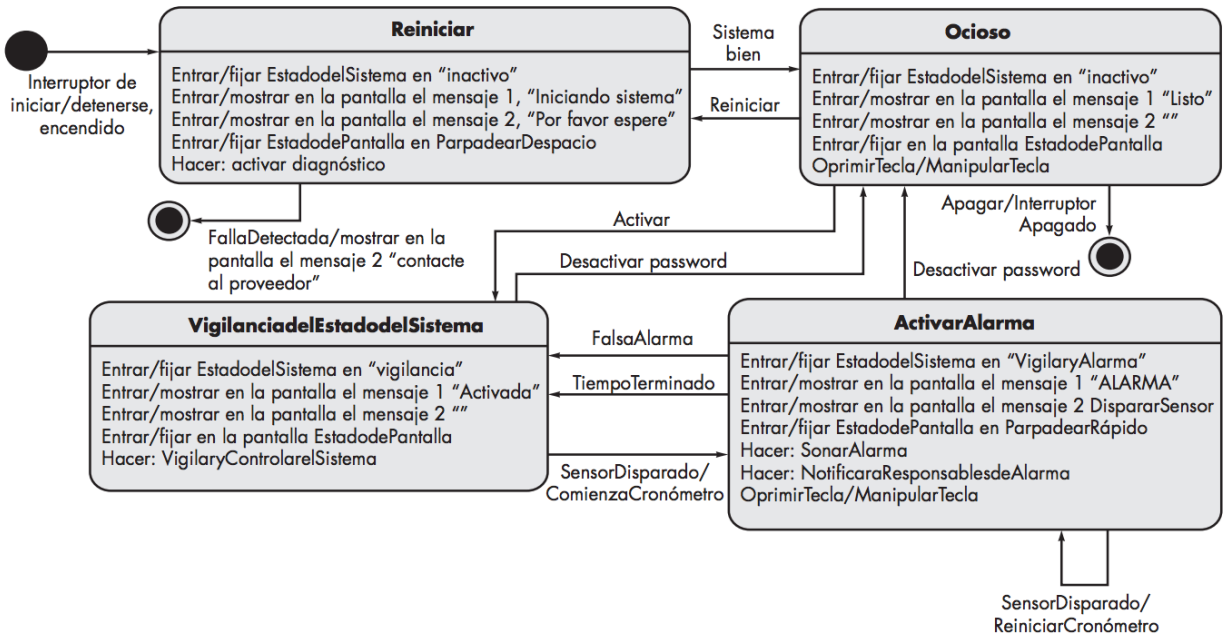


Figura Nº3-22 - Diagrama de estado para la función de seguridad de CasaSegura

La tabla de activación del proceso (TAP) es un modo algo distinto de representar el comportamiento. La TAP representa la información contenida en el diagrama de estado en el contexto de los procesos, no de los estados. Es decir, la tabla indica cuáles procesos (burbujas) serán invocados en el modelo del flujo cuando ocurra un evento. La TAP se usa como guía para un diseñador que debe construir una ejecución que controle los procesos representados en este nivel.

En la figura 3-23 se aprecia una TAP para el nivel 1 del modelo de flujo del software de CasaSegura. La CSPEC describe el comportamiento del sistema, pero no da información acerca del funcionamiento interno de los procesos que se activan como resultado de dicho comportamiento.

eventos de entrada						
evento de sensor	0	0	0	0	1	0
bandera de parpadeo	0	0	1	1	0	0
interruptor de iniciar o detener	0	1	0	0	0	0
estado de la acción en pantalla terminado	0	0	0	1	0	0
en marcha	0	0	1	0	0	0
tiempo terminado	0	0	0	0	0	1
salida						
señal de alarma	0	0	0	0	1	0
activación del proceso						
vigilar y controlar el sistema	0	1	0	0	1	1
activar o desactivar el sistema	0	1	0	0	0	0
mostrar mensajes y estado	1	0	1	1	1	1
interactuar con el usuario	1	0	0	1	0	1

Figura N°3-23 - Tabla de activación del proceso para la función de seguridad de CasaSegura

La especificación del proceso

La especificación del proceso (PSPEC) se utiliza para describir todos los procesos del modelo del flujo que aparecen en el nivel final de la mejora. El contenido de la especificación del proceso incluye el texto narrativo, una descripción del lenguaje de diseño del programa[26] del algoritmo del proceso, ecuaciones matemáticas, tablas o diagramas de actividad UML. Si se da una PSPEC que acompañe a cada burbuja del modelo del flujo, se crea una "miniespecificación" que sirve como guía para diseñar la componente del software que implementará la burbuja. Para ilustrar el uso de la PSPEC, considere la transformación procesar password representada en el modelo de flujo de la figura 31. La PSPEC de esta función adopta la forma siguiente:

PSPEC: procesar password (en el panel de control). La transformación procesar password realiza la validación en el panel de control para la función de seguridad de CasaSegura. Procesar password recibe un password de cuatro dígitos de la función interactuar con usuario. Primero, el password se compara con el password maestro almacenado dentro del sistema. Si el password maestro coincide, se pasa a la función mostrar mensaje y estado. Si el password maestro no coincide, los cuatro dígitos se comparan con una tabla de passwords secundarios (que deben asignarse para recibir invitados o trabajadores que necesiten entrar a la casa cuando el propietario no esté presente). Si el password coincide

[26] El lenguaje de diseño del programa (LDP) mezcla la sintaxis del lenguaje de programación con el texto narrativo a fin de dar detalles del diseño del procedimiento.

con una entrada de la tabla, se pasa a la función mostrar mensaje y estado. Si no coinciden, se pasa a la función de mostrar mensaje y estado.

Si en esta etapa se desean detalles algorítmicos adicionales, también puede incluirse una representación del lenguaje de diseño del programa (LDP) como parte de la PSPEC. Sin embargo, muchos profesionales piensan que la versión LDP debe posponerse hasta comenzar el diseño de los componentes.

3.2.3 Modelo de requerimientos para WebApps

Es frecuente que los desarrolladores de web manifiesten escepticismo cuando se plantea la idea del análisis de los requerimientos para webapps. Acostumbran decir: "después de todo, el proceso de desarrollo en web debe ser ágil y el análisis toma tiempo. Nos hará ser lentos justo cuando necesitemos diseñar y construir la webapp".

El análisis de los requerimientos lleva tiempo, pero resolver el problema equivocado toma aún más tiempo. La pregunta que debe responder todo desarrollador en web es sencilla: ¿estás seguro de que entiendes los requerimientos del problema? Si la respuesta es un "sí" inequívoco, entonces tal vez sea posible omitir el modelado de los requerimientos, pero si la respuesta es "no", entonces ésta debe llevarse a cabo.

¿Cuánto análisis es suficiente?

El grado en el que se profundice en el modelado de los requerimientos para las webapps depende de los factores siguientes:

- Tamaño y complejidad del incremento de la webapp.
- Número de participantes (el análisis ayuda a identificar los requerimientos conflictivos que provienen de distintas fuentes).
- Tamaño del equipo de la webapp.
- Grado en el que los miembros del equipo han trabajado juntos antes (el análisis ayuda a desarrollar una comprensión común del proyecto).
- Medida en la que el éxito de la organización depende directamente del éxito de la webapp.

El inverso de los puntos anteriores es que a medida que el proyecto se hace más chico, que el número de participantes disminuye, que el equipo de desarrollo es más cohesivo y que la aplicación es menos crítica, es razonable aplicar un enfoque más ligero para el análisis.

Aunque es una buena idea analizar el problema antes de que comience el diseño, no es verdad que todo el análisis deba preceder a todo el diseño. En realidad, el diseño de una parte específica de la webapp sólo demanda un análisis de los requerimientos que afectan a sólo esa parte de la webapp. Como un ejemplo proveniente de CasaSegura, podría diseñarse con validez la estética general del sitio web (formatos, colores, etc.) sin tener que analizar los requerimientos funcionales de las capacidades de comercio electrónico. Sólo se necesita analizar aquella parte del problema que sea relevante para el trabajo de diseño del incremento que se va a entregar.

Entrada del modelado de los requerimientos

Previamente se analizó una versión ágil del proceso de software general que puede aplicarse cuando se hace la ingeniería de las webapps. El proceso incorpora una actividad de comunicación que identifica a los participantes y las categorías de usuario, el contexto del negocio, las metas definidas de información y aplicación, requerimientos generales de webapps y los escenarios de uso, información que se convierte en la entrada del modelado de los requerimientos. Esta información se representa en forma de descripciones hechas en lenguaje natural, a grandes rasgos, en bosquejos y otras representaciones no formales.

El análisis toma esta información, la estructura con el empleo de un esquema de representación definido formalmente (donde sea apropiado) y luego produce como salida modelos más rigurosos. El modelo de requerimientos brinda una indicación detallada de la verdadera estructura del problema y da una perspectiva de la forma de la solución.

Algunos aspectos de esta información faltante emergerían de manera natural durante el diseño. Los ejemplos quizá incluyan el formato específico de los botones de función, su aspecto y percepción estética, el tamaño de las vistas instantáneas, la colocación del ángulo de las cámaras y el plano de la casa, o incluso minucias tales como las longitudes máxima y mínima de las claves. Algunos de estos aspectos son decisiones de diseño (como el aspecto de los botones) y otros son requerimientos (como la longitud de las claves) que no influyen de manera fundamental en los primeros trabajos de diseño.

Pero cierta información faltante sí podría influir en el diseño general y se relaciona más con la comprensión real de los requerimientos. Por ejemplo:

- P1: ¿Cuál es la resolución del video de salida que dan las cámaras de CasaSegura?
- P2: ¿Qué ocurre si se encuentra una condición de alarma mientras la cámara está siendo vigilada?
- P3: ¿Cómo maneja el sistema las cámaras con vistas panorámicas y de acercamiento?
- P4: ¿Qué información debe darse junto con la vista de la cámara (por ejemplo, ubicación, fecha y hora, último acceso, etcétera)?

Ninguna de estas preguntas fue identificada o considerada en el desarrollo inicial del caso de uso; no obstante, las respuestas podrían tener un efecto significativo en los diferentes aspectos del diseño.

Por tanto, es razonable concluir que aunque la actividad de comunicación provea un buen fundamento para entender, el análisis de los requerimientos mejora este entendimiento al dar una interpretación adicional. Como la estructura del problema se delinea como parte del modelo de requerimientos, invariablemente surgen preguntas. Son éstas las que llenan los huecos y, en ciertos casos, en realidad ayudan a encontrarlos.

En resumen, la información obtenida durante la actividad de comunicación será la entrada del modelo de los requerimientos, cualquiera que sea, desde un correo electrónico informal hasta un proyecto detallado con escenarios de uso exhaustivos y especificaciones del producto.

Salida del modelado de los requerimientos

El análisis de los requerimientos provee un mecanismo disciplinado para representar y evaluar el contenido y funcionamiento de las webapp, los modos de interacción que hallarán los usuarios y el ambiente e infraestructura en las que reside la webapp.

Cada una de estas características se representa como un conjunto de modelos que permiten que los requerimientos de la webapp sean analizados en forma estructurada. Si bien los modelos específicos dependen en gran medida de la naturaleza de la webapp, hay cinco clases principales de ellos:

- **Modelo de contenido:** identifica el espectro completo de contenido que dará la webapp. El contenido incluye datos de texto, gráficos e imágenes, video y sonido.
- **Modelo de interacción:** describe la manera en que los usuarios interactúan con la webapp.
- **Modelo funcional:** define las operaciones que se aplicarán al contenido de la webapp y describe otras funciones de procesamiento que son independientes del contenido pero necesarias para el usuario final.
- **Modelo de navegación:** define la estrategia general de navegación para la webapp.
- **Modelo de configuración:** describe el ambiente e infraestructura en la que reside la webapp.

Es posible desarrollar cada uno de estos modelos con el empleo de un esquema de representación (llamado con frecuencia "lenguaje") que permite que su objetivo y estructura se comuniquen y evalúen con facilidad entre los miembros del equipo de ingeniería de web y otros participantes. En consecuencia, se identifica una lista de aspectos clave (como errores, omisiones, inconsistencias, sugerencias de mejora o modificaciones, puntos de aclaración, etc.) para trabajar sobre ellos.

Modelo del contenido de las webapps

El modelo de contenido incluye elementos estructurales que dan un punto de vista importante de los requerimientos del contenido de una webapp. Estos elementos estructurales agrupan los objetos del contenido y todas las clases de análisis, entidades visibles para el usuario que se crean o manipulan cuando éste interactúa con la webapp.

El contenido puede desarrollarse antes de la implementación de la webapp, mientras ésta se construye o cuando ya opera. En cualquier caso, se incorpora por referencia de navegación en la estructura general de la webapp. Un objeto de contenido es una descripción de un producto en forma de texto, un artículo que describe un evento deportivo, una fotografía tomada en éste, la respuesta de un usuario en un foro de análisis, una representación animada de un logotipo corporativo, una película corta de un discurso o una grabación en audio para una presentación con diapositivas. Los objetos de contenido pueden almacenarse como archivos separados, incrustarse directamente en páginas web u

obtenerse en forma dinámica de una base de datos. En otras palabras, un objeto de contenido es cualquier aspecto de información cohesiva que se presente al usuario final.

Los objetos de contenido se determinan directamente a partir de casos de uso, estudiando la descripción del escenario respecto de referencias directas e indirectas al contenido. Por ejemplo, se establece en CasaSeguraAsegurada.com una webapp que da apoyo a CasaSegura. Un caso de uso, Comprar componentes seleccionados de CasaSegura, describe el escenario que se requiere para comprar un componente de CasaSegura y que contiene la siguiente oración:

Podré obtener información descriptiva y de precios de cada componente del producto.

El modelo de contenido debe ser capaz de describir el objeto de contenido Componente. En muchas circunstancias, para definir los requerimientos para el contenido que debe diseñarse e implementarse, es suficiente una lista sencilla de los objetos de contenido, junto con la descripción breve de cada uno. Sin embargo, en ciertos casos, el modelo de contenido se beneficia de un análisis más rico que ilustre en forma gráfica las relaciones entre los objetos de contenido y la jerarquía que mantiene una webapp.

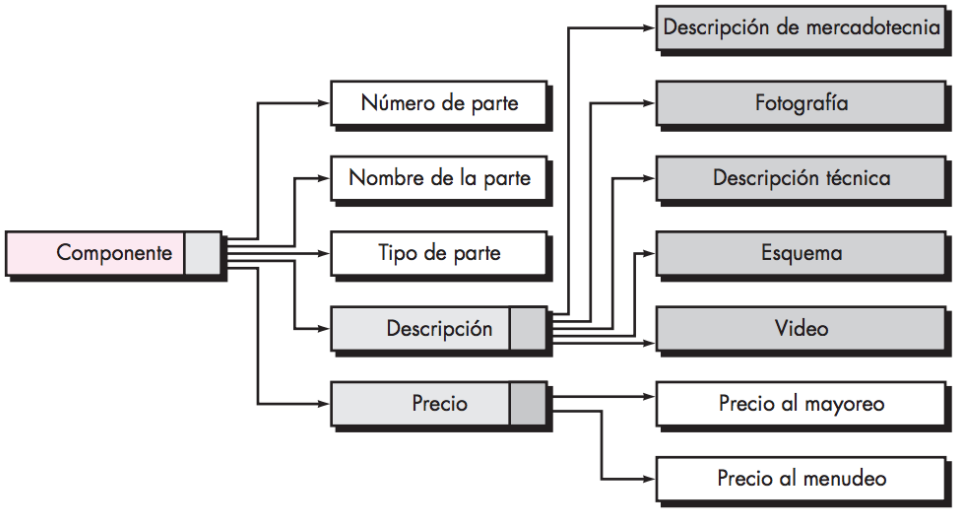


Figura N°3-24 - Árbol de datos para el componente CasaSeguraAsegurada.com

Por ejemplo, tome en cuenta el árbol de datos [Sri01] creado por el componente CasaSeguraAsegurada.com que aparece en la figura 35. El árbol representa una jerarquía de

información que se utiliza para describir un componente. Los aspectos de datos simples o compuestos (uno o más valores de los datos) se representan con rectángulos sin sombra. Los objetos de contenido se representan con rectángulos con sombra. En la figura, descripción está definida por cinco objetos (los rectángulos sombreados). En ciertos casos, uno o más de estos objetos se mejorará más conforme se expanda el árbol de datos.

Es posible crear un árbol de datos para cualquier contenido que se componga de múltiples objetos de contenido y aspectos de datos. El árbol de datos se desarrolla como un esfuerzo para definir relaciones jerárquicas entre los objetos de contenido y para dar un medio de revisión del contenido a fin de que se descubran las omisiones e inconsistencias antes de que comience el diseño. Además, el árbol de datos sirve como base para diseñar el contenido.

Modelo de la interacción para webapps

La gran mayoría de webapps permiten una "conversación" entre un usuario final y funcionalidad, contenido y comportamiento de la aplicación. Esta conversación se describe con el uso de un modelo de interacción que se compone de uno o más de los elementos siguientes: 1) casos de uso, 2) diagramas de secuencia, 3) diagramas de estado y 4) prototipos de la interfaz de usuario.

En muchas instancias, basta un conjunto de casos de uso para describir la interacción en el nivel del análisis (durante el diseño se introducirán más mejoras y detalles). Sin embargo, cuando la secuencia de interacción es compleja e involucra múltiples clases de análisis o muchas tareas, es conveniente ilustrarla de forma más rigurosa mediante un diagrama.

El formato de la interfaz de usuario, el contenido que presenta, los mecanismos de interacción que implementa y la estética general de las conexiones entre el usuario y la webapp tienen mucho que ver con la satisfacción de éste y con el éxito conjunto del software. Aunque se afirme que la creación de un prototipo de interfaz de usuario es una actividad de diseño, es una buena idea llevarla a cabo durante la creación del modelo de análisis. Entre más pronto se revise la representación física de la interfaz de usuario, más probable es que los consumidores finales obtengan lo que desean.

Como hay muchas herramientas para construir webapps baratas y poderosas en sus funciones, es mejor crear el prototipo de la interfaz con el empleo de ellas. El prototipo debe implementar los vínculos de navegación principales y representar la pantalla general en forma muy parecida a la que se construirá. Por ejemplo, si van a ponerse a disposición del usuario final cinco funciones principales del sistema, el prototipo debe representarlas tal como las verá cuando entre por primera vez a la webapp. ¿Se darán vínculos gráficos? ¿Dónde se desplegará el menú de navegación? ¿Qué otra información verá el usuario? Preguntas como éstas son las que debe responder el prototipo.

Modelo funcional para las webapps

Muchas webapps proporcionan una amplia variedad de funciones de computación y manipulación que se asocian directamente con el contenido (porque lo utilizan o porque lo producen) y es frecuente que sean un objetivo importante de la interacción entre el usuario y la webapp. Por esta razón, deben analizarse los requerimientos funcionales y modelarlos cuando sea necesario.

El modelo funcional enfrenta dos elementos de procesamiento de la webapp, cada uno de los cuales representa un nivel distinto de abstracción del procedimiento: 1) funciones observables por los usuarios que entrega la webapp a éstos y 2) las operaciones contenidas en las clases de análisis que implementan comportamientos asociados con la clase.

La funcionalidad observable por el usuario agrupa cualesquiera funciones de procesamiento que inicie directamente el usuario. Por ejemplo, una webapp financiera tal vez implemente varias funciones de finanzas (como una calculadora de ahorros para una colegiatura universitaria o un fondo para el retiro). Estas funciones en realidad se implementan con el uso de operaciones dentro de clases de análisis, pero desde el punto de vista del usuario final; el resultado visible es la función (más correctamente, los datos que provee la función).

En un nivel más bajo de abstracción del procedimiento, el modelo de requerimientos describe el procesamiento que se realizará por medio de operaciones de clase de análisis. Estas operaciones manipulan los atributos de clase y se involucran como clases que colaboran entre sí para lograr algún comportamiento que se desea.

Sin que importe el nivel de abstracción del procedimiento, el diagrama de actividades UML se utiliza para representar detalles de éste. En el nivel de análisis, los diagramas de actividades deben usarse sólo donde la funcionalidad sea relativamente compleja. Gran parte de la complejidad de muchas webapps ocurre no en las funciones que proveen, sino en la naturaleza de la información a que se accede y en las formas en las que se manipula.

Un ejemplo de complejidad relativa de la funcionalidad para CasaSeguraAsegurada.com se aborda en un caso de uso llamado Obtener recomendaciones para la distribución de sensores en mi espacio. El usuario ya ha desarrollado la distribución del espacio que se vigilará y en este caso de uso selecciona dicha distribución y solicita ubicaciones recomendables para los sensores dentro de ella. CasaSeguraAsegurada.com responde con la representación gráfica de la distribución por medio de información adicional acerca de la ubicación recomendable para los sensores. La interacción es muy sencilla, el contenido es algo más complejo, pero la funcionalidad subyacente es muy sofisticada. El sistema debe realizar un análisis relativamente complejo de la planta del piso para determinar el conjunto óptimo de sensores. Debe examinar las dimensiones de la habitación, la ubicación de puertas y ventanas, y coordinar éstas con la capacidad y especificaciones de los sensores. ¡No es una tarea fácil! Para describir el procesamiento de este caso de uso se utiliza un conjunto de diagramas de actividades.

El segundo ejemplo es el caso de uso Controlar cámaras. En éste, la interacción es relativamente sencilla, pero existe el potencial de una funcionalidad compleja, dado que dicha operación “sencilla” requiere una comunicación compleja con dispositivos ubicados en posiciones remotas y a los que se accede por internet. Una complicación adicional se relaciona con la negociación del control cuando varias personas autorizadas tratan de vigilar o controlar un mismo sensor al mismo tiempo.

La figura 36 ilustra el diagrama de actividades para la operación TomarControldeCámara que forma parte de la clase de análisis Cámara usada dentro del caso de uso Controlar cámaras. Debe observarse que con el flujo de procedimiento se invocan dos operaciones adicionales: SolicitarBloqueodeCámara(), que trata de bloquear la cámara para este usuario, y ObtenerUsuarioActualdeCámara(), que recupera el nombre del usuario que controla en ese momento la cámara. Los detalles de construcción indican cómo se invocan estas

operaciones, y los de la interfaz para cada operación no se señalan hasta que comienza el diseño de la webapp.

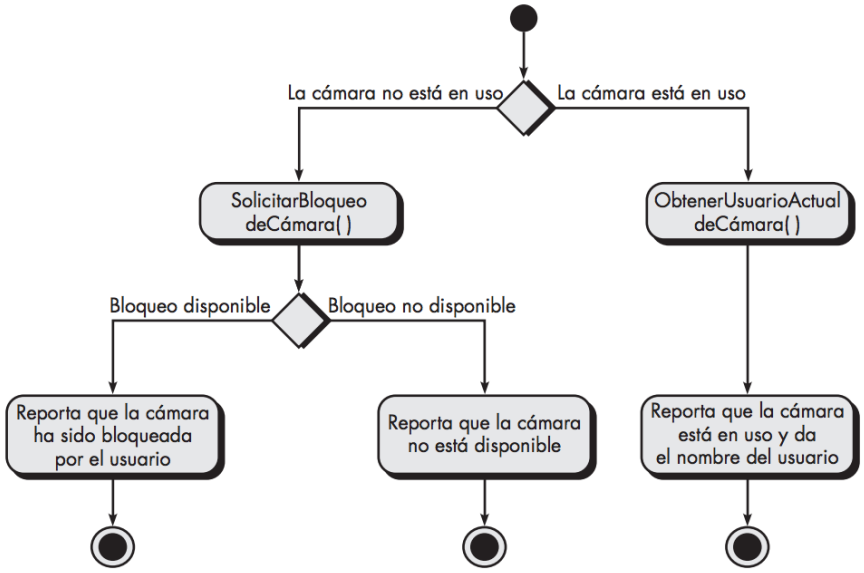


Figura N°3-25 - Diagrama de actividades para la operación TomarControldeCámara()

Modelos de configuración para las webapps

En ciertos casos, el modelo de configuración no es sino una lista de atributos del lado del servidor y del lado del cliente. Sin embargo, para webapps más complejas, son varias las dificultades de configuración (por ejemplo, distribuir la carga entre servidores múltiples, arquitecturas caché, bases de datos remotas, distintos servidores que atienden a varios objetos en la misma página web, etc.) que afectan el análisis y diseño. El diagrama de despliegue UML se utiliza en situaciones en las que deben considerarse arquitecturas de configuración compleja.

Para CasaSeguraAsegurada.com, deben especificarse el contenido y funcionalidad públicos a fin de que sean accesibles a través de todos los clientes principales de web (como aquéllos con 1 por ciento o más de participación en el mercado)[27]. A la inversa, es aceptable restringir las funciones más complejas de control y vigilancia (que sólo es accesible para los usuarios tipo Propietario) a un conjunto más pequeño de clientes. El modelo de

[27] La determinación de la participación en el mercado para los navegadores es notoriamente problemática y varía en función de cuál fuente se utilice. No obstante, en el momento de escribir este libro, Internet Explorer y Firefox eran los únicos que sobrepasaban 30 por ciento, y Mozilla, Opera y Safari los únicos que superaban de manera consistente 1 por ciento.

configuración para CasaSeguraAsegurada.com también especificará la operación cruzada con las bases de datos de productos y aplicaciones de vigilancia.

Modelado de la navegación Para modelar la navegación se considera cómo navegará cada categoría de usuario de un elemento de la webapp (como un objeto de contenido) a otro. La mecánica de navegación se define como parte del diseño. En esa etapa debe centrarse la atención en los requerimientos generales de navegación. Deben considerarse las preguntas siguientes:

- ¿Ciertos elementos deben ser más fáciles de alcanzar (requieren menos pasos de navegación) que otros? ¿Cuál es la prioridad de presentación?
- ¿Debe ponerse el énfasis en ciertos elementos para forzar a los usuarios a navegar en esa dirección?
- ¿Cómo deben manejarse los errores en la navegación?
- ¿Debe darse prioridad a la navegación hacia grupos de elementos relacionados y no hacia un elemento específico?
- ¿La navegación debe hacerse por medio de vínculos, acceso basado en búsquedas o por otros medios?
- ¿Debe presentarse a los usuarios ciertos elementos con base en el contexto de acciones de navegación previas?
- ¿Debe mantenerse un registro de usuarios de la navegación?
- ¿Debe estar disponible un mapa completo de la navegación (en oposición a un solo vínculo para “regresar” o un apuntador dirigido) en cada punto de la interacción del usuario?
- ¿El diseño de la navegación debe estar motivado por los comportamientos del usuario más comunes y esperados o por la importancia percibida de los elementos definidos de la webapp?
- ¿Un usuario puede “guardar” su navegación previa a través de la webapp para hacer expedito el uso futuro?
- ¿Para qué categoría de usuario debe diseñarse la navegación óptima?
- ¿Cómo deben manejarse los vínculos externos hacia la webapp? ¿Con la superposición de la ventana del navegador existente? ¿Como nueva ventana del navegador? ¿En un marco separado?

Estas preguntas y muchas otras deben plantearse y responderse como parte del análisis de la navegación.

Usted y otros participantes también deben determinar los requerimientos generales para la navegación. Por ejemplo, ¿se dará a los usuarios un "mapa del sitio" y un panorama de toda la estructura de la webapp? ¿Un usuario puede hacer una "visita guiada" que resalte los elementos más importantes (objetos y funciones de contenido) con que se disponga? ¿Podrá acceder un usuario a los objetos o funciones de contenido con base en atributos definidos de dichos elementos (por ejemplo, un usuario tal vez desee acceder a todas las fotografías de un edificio específico o a todas las funciones que permiten calcular el peso)?

3.2.4 Conclusiones

Los modelos orientados al flujo se centran en el flujo de objetos de datos a medida que son transformados por las funciones de procesamiento. Derivados del análisis estructurado, los modelos orientados al flujo usan el diagrama de flujo de datos, notación de modelación que ilustra la manera en la que se transforma la entrada en salida cuando los objetos de datos se mueven a través del sistema. Cada función del software que transforme datos es descrita por la especificación o narrativa de un proceso. Además del flujo de datos, este elemento de modelación también muestra el flujo del control, representación que ilustra cómo afectan los eventos al comportamiento de un sistema.

El modelado del comportamiento ilustra el comportamiento dinámico. El modelo de comportamiento utiliza una entrada basada en el escenario, orientada al flujo y elementos basados en clases para representar los estados de las clases de análisis y al sistema como un todo. Para lograr esto, se identifican los estados y se definen los eventos que hacen que una clase (o el sistema) haga una transición de un estado a otro, así como las acciones que ocurren cuando se efectúa dicha transición. Los diagramas de estado y de secuencia son la notación que se emplea para modelar el comportamiento.

Los patrones de análisis permiten a un ingeniero de software utilizar el conocimiento del dominio existente para facilitar la creación de un modelo de requerimientos. Un patrón de análisis describe una característica o función específica del software que puede describirse con un conjunto coherente de casos de uso. Especifica el objetivo del patrón, la motivación para su uso, las restricciones que limitan éste, su aplicabilidad en distintos dominios de

problemas, la estructura general del patrón, su comportamiento y colaboraciones, así como información suplementaria.

El modelado de los requerimientos para las webapps utiliza la mayoría, si no es que todos, los elementos de modelado que se estudian en el libro. Sin embargo, dichos elementos se aplican dentro de un conjunto de modelos especializados que se abocan al contenido, interacción, función, navegación y configuración cliente-servidor en la que reside la webapp

3.3 DESCRIPCIÓN DE LA SOLUCIÓN

Para diseñar y elaborar la propuesta de solución de un anexo técnico para el análisis de requerimientos, se estudiaron las diferentes actividades y tareas que se llevan a cabo en esta fase, así como los diferentes aspectos que influyen en las mismas. Esto se realizó, con el propósito de identificar problemáticas y oportunidades, para recrear funcionalidades computacionales que facilitarán dichas actividades.

Además, en esta sección se exponen los aspectos más relevantes del modelo de análisis de requerimientos resultado de la investigación, y cómo se pueden realizar algunos procedimientos de este modelo a través de técnicas computacionales. También se exponen las características que deben poseer las fases de recolección, de manera que satisfagan el modelo propuesto, y cómo este es aplicable de manera transversal, a cada uno de los diferentes niveles en los cuales puede residir el proceso de ingeniería de requerimientos.

3.3.1 Recolección de requerimientos

Como se mencionó anteriormente, las actividades que se realizan en la fase de recolección de requerimientos, son determinantes a la hora de realizar el análisis de los mismos, ya que constituyen la base para dicho análisis. Para la fase de recolección de requerimientos, se definieron un conjunto básico de actividades necesario para satisfacer las restricciones del análisis. Este conjunto de actividades, se basa en dos enfoques del proceso de recolección. El primero, propuesto por Zair Abdelouahab y Marcia Carvalho (Abdelouahab & Carvalho, 2001), y el segundo, propuesto por Ian Sommerville (Sommerville, 2015). Ambos enfoques son abiertos y modularizados, y favorecen el desarrollo de esta fase a través de actividades desarrolladas en etapas estándares. El conjunto de actividades que se debe llevar a cabo en esta fase son:

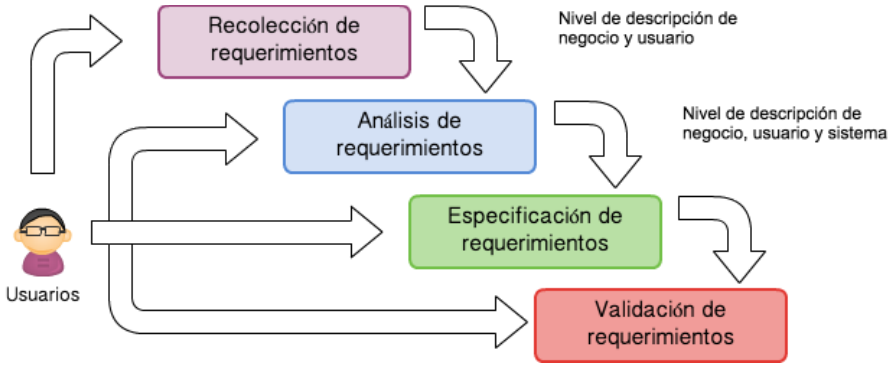


Figura N°3-26 - Flujo de recolección de requerimientos

En la figura 3-26, se puede observar un modelo de proceso para la recolección de requerimientos, constituido por un conjunto de actividades encaminadas a soportar y facilitar la comprensión del análisis de requerimientos. No existe un único método, restrictivo a través del cual se deban llevar a cabo las actividades descritas en la figura 3-26. Las empresas que cuentan con un proceso definido y maduro en el cual se realizan en la actualidad estas actividades, no poseen ningún inconveniente para aplicar el modelo de análisis de requerimientos. Sin embargo, las empresas que no cuentan con un proceso de recolección definido (nivel 0 ó 1), y por tanto, no realizan en la actualidad estas actividades; deben utilizar alguna herramienta, técnica o estrategia a través de la cual se lleven a cabo las actividades formuladas en el modelo.

A continuación, se exponen algunas sugerencias de estrategias para llevar a cabo cada una de esas actividades.

- **Recolección de objetivos del sistema:** es la primera actividad que se debe llevar a cabo en esta fase. En ella, se busca identificar los objetivos del sistema, que corresponden a los requerimientos del sistema descritos en un nivel de descripción de negocio.

  A continuación se explican algunas de las estrategias recomendadas para llevar a cabo esta actividad:

  > Identificación de objetivos iniciales: consiste en extraer los objetivos del sistema, a partir del análisis de la documentación existente o de entrevistas realizadas a los stakeholders. El análisis de documentación existente se realiza examinando, ya sea las entrevistas realizadas a los usuarios del sistema en búsqueda de palabras (verbos) que representen una acción; o, analizando cómo expresan sus requerimientos en forma de operaciones, procesos o diagramas de flujo, en cualquier documento (normalmente informal) que exista acerca del dominio del negocio donde la aplicación será desarrollada.

  > Preguntas específicas: consiste en establecer un conjunto de preguntas específicas previamente diseñadas por el ingeniero de requerimientos (u otro participante que asuma ese rol), con el fin de aplicar en una entrevista o cuestionario a los diferentes stakeholders, de manera que se identifiquen los objetivos del sistema.

  > Plantillas: esta estrategia propone el uso de una plantilla para la recolección de los objetivos del sistema. Esta, debe contener campos para ingresar información, como

un número de identificación del objetivo, descripción del objetivo y otros definidos por el ingeniero de requerimientos.

> Estrategia lingüística: consiste en la formalización de un lenguaje sobre el cual se especifique y recolecte la información correspondiente a los objetivos del sistema.

- **Descomposición de los objetivos del sistema en requerimientos de usuario:** esta actividad persigue como objetivo el descomponer los objetivos previamente establecidos, en requerimientos más detallados (requerimientos de usuario), enfocados en la funcionalidad del negocio sobre el cual se está trabajando. Esta etapa es realizada para cada uno de los objetivos de negocio identificados en la fase anterior. Para las empresas que no cuenten con un proceso maduro, y se encuentren en lo niveles inferiores (nivel 0 y 1); se recomienda el uso de escenarios y/o casos de uso. Para cada una de estas herramientas, se analizarán algunas de las estrategias que se pueden utilizar para llevarlas a cabo.
- **Casos de uso o plantillas:** esta estrategia propone especificar casos de uso por medio de una plantilla, al igual que lo descrito para los objetivos del sistema. Esta debe contener la información correspondiente al requerimiento que se desea especificar. Esta son plantilla contiene los campos estándar para los casos de uso (entendidos como requerimientos funcionales del sistema). Para esta plantilla, se utiliza una notación que permite implementar patrones lingüísticos a través una notación específica. Esta notación está compuesta así: Las palabras al interior de < > deben ser reemplazadas de la manera apropiada, y las que se encuentran al interior de { } y separadas por coma, representan opciones excluyentes de manera que sólo una puede ser seleccionada

Tabla N°3-01 - Plantilla para la descripción de requerimientos funcionales

<table>
<tr><td>Requerimiento funcional</td><td>< Nombre descriptivo del requerimiento funcional ></td></tr>
<tr><td>Versión</td><td>< Número de la versión actual del requerimiento indicando la fecha del registro ></td></tr>
<tr><td>Autor</td><td>< Nombre de los autores involucrados en la generación del requerimiento ></td></tr>
<tr><td>Fuente</td><td>< De donde se obtuvo referencias sobre el requerimiento (empresa, expertos, libros, etc) ></td></tr>
<tr><td>Propósito</td><td>< Propósito de la realización del requerimiento funcional dentro del sistema ></td></tr>
<tr><td>Descripción</td><td>< Detalle de la funcionalidad del requerimiento funcional ></td></tr>
<tr><td>Alcance y nivel</td><td>< Hasta qué nivel se desarrollará el requerimiento y cuál será su alcance dentro del proyecto ></td></tr>
<tr><td>Condiciones de aprobación</td><td>< Condiciones con las cuales se dará por aprobado el requerimiento funcional ></td></tr>
<tr><td>Condiciones de rechazo</td><td>< Condiciones con las cuales se rechazará el requerimiento funcional ></td></tr>
<tr><td>Actores</td><td>< Actores involucrados con este requerimiento y sus permisos correspondientes ></td></tr>
<tr><td>Evento disparador</td><td>< Evento o acción que da inicio este requerimiento funcional ></td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><th>Paso</th><th>Acción</th></tr>
<tr><td>1</td><td></td></tr>
<tr><td>2</td><td></td></tr>
<tr><td>3</td><td></td></tr>
<tr><td>...</td><td></td></tr>
<tr><td>1000</td><td></td></tr>
</table>
</td></tr>
<tr><td>Condición posterior</td><td>< Condición posterior del caso de uso del requerimiento funcional ></td></tr>
</table>

<table>
<tr><td>Excepciones y extensiones</td><td>
<table>
<tr><th>Paso</th><th>Acción</th></tr>
<tr><td>1</td><td></td></tr>
<tr><td>2</td><td></td></tr>
<tr><td>3</td><td></td></tr>
<tr><td>...</td><td></td></tr>
<tr><td>1000</td><td></td></tr>
</table>
</td></tr>
<tr><td colspan="2">INFORMACIÓN RELACIONADA</td></tr>
<tr><td>Prioridad</td><td>< Grado de prioridad del requerimiento funcional dentro del desarrollo del sistema > (Crítico, Alto Medio, Bajo)</td></tr>
<tr><td>Frecuencia</td><td>< Cantidad de veces que este requerimiento será utilizado en el sistema ></td></tr>
<tr><td>Desempeño</td><td>
<table>
<tr><th>Acción</th><th>Medida</th></tr>
<tr><td>1</td><td></td></tr>
<tr><td>2</td><td></td></tr>
<tr><td>3</td><td></td></tr>
<tr><td>...</td><td></td></tr>
<tr><td>1000</td><td></td></tr>
</table>
</td></tr>
<tr><td>Canales hacia los actores</td><td>< Archivos, interactivos, bases de datos, etc ></td></tr>
<tr><td>Características abiertas</td><td>< Listado de características que pueden afectar las decisiones sobre el caso de uso ></td></tr>
<tr><td>Requerimientos superiores</td><td>< Listado de requerimientos que ocupan este requerimiento funcional ></td></tr>
<tr><td>Requerimientos inferiores</td><td>< Listado de requerimientos que se ocupan en este requerimiento funcional ></td></tr>
<tr><td>Comentarios</td><td>< Observaciones adicionales sobre el requerimiento funcional ></td></tr>
</table>

La correspondiente definición de cada uno de los campos de esta plantilla es:

- **Versión:** De acuerdo a las recomendaciones de la IEEE, y de compañías líderes en procesos de software como Rational, un requerimiento debe permitir manejar sus distintas versiones de manera que se pueda analizar la evolución del mismo a través del tiempo. Para todo requerimiento, este campo debe estar ocupado por la versión actual, con su correspondiente número y fecha.
- **Autor y fuente:** Contiene el nombre de la organización y el autor del requerimiento.
- Propósito: Contiene la descripción del porqué el requerimiento consignado es necesario para alcanzar los objetivos del negocio.
- **Descripción:** Para los requerimientos funcionales, esta plantilla contiene un patrón lingüístico que indica que debe ser llenado y los eventos que disparan el requerimiento.
- **Alcance y nivel:** Cuál sistema debe ser considerado como caja negra para este requerimiento.
- **Precondición:** Se ingresan las condiciones necesarias que se deben tener para llevar a cabo el requerimiento que se está describiendo.
- **Condición de aprobación:** Condición que indica si la ejecución del requerimiento fue exitosa.
- **Condición de rechazo:** Condición que indica que la ejecución del requerimiento fue abortada o falló.
- **Actores:** Actor primario, y/o lista de actores secundarios del requerimiento.
- **Evento disparador:** Evento que dispara la realización del requerimiento.
- **Secuencia normal:** Aquí se deposita la secuencia de interacciones que tiene el usuario con el sistema, en orden de llevar a cabo una funcionalidad u operación. Los pasos que se llevan a cabo en pueden a su vez contener sub-pasos o secuencias anidadas, asumiendo que sólo una de estas se puede llevar a cabo.
- **Condiciones posteriores:** Son los estados a los cuales se debe llegar o los resultados que se deben obtener después de ejecutar la funcionalidad descrita en el requerimiento.
- **Excepciones y extensiones:** Durante la interacción descrita en la secuencia ordinaria se pueden presentar excepciones o extensiones condicionales, debido a flujos alternativos de dicha interacción entre el usuario y el sistema. Es este campo se

especifica la secuencia que se tomaría si se presenta la excepción o se indica si la funcionalidad descrita en el requerimiento se aborta.

- **Prioridad:** Indica qué tan importante es el requerimiento, que grado de prioridad tiene el requerimiento para los usuarios y clientes.
- **Frecuencia:** Frecuencia con la que se lleva a cabo la ejecución del requerimiento en un intervalo de tiempo. Se indica la tolerancia a fallos que tiene el requerimiento de acuerdo al número de eventos que se utiliza el requerimiento y el tiempo que el mismo debe estar disponible.
- **Desempeño:** Aquí se especifica un tiempo de tolerancia respecto al tiempo de respuesta del sistema para alguno o todos los pasos de la secuencia ordinaria.
- **Canales hacia los actores:** Canales como archivos, interactivo, base de datos a través de los cuales se expresa el resultado de la ejecución del requerimiento
- **Características abiertas:** Lista de características que pueden afectar las decisiones sobre el caso de uso
- **Requerimientos superiores:** Listado de requerimientos que ocupan este requerimiento funcional.
- **Requerimientos inferiores:** Listado de requerimientos que se ocupan en este requerimiento funcional.
- **Comentarios:** En este campo se ingresa toda la información que no se pudo agregar a ninguno de los campos.

CAPÍTULO 4 CONCLUSIONES

En este capítulo se presentan las conclusiones respectivas a todo el proceso realizado en este proyecto. Lo cual implica recapitular y concluir en base a los objetivos planteados, indicando los objetivos realizados y en qué medida estos se llevaron a cabo en tu totalidad. Luego, se expresan las implicancias futuras que el desarrollo de este trabajo y los resultados encontrados podrían tener a nivel teórico y práctico. Posteriormente se proponen los alcances inherentes a los resultados encontrados producto de la ejecución del diseño experimental planteado. En base a estos, se propone el trabajo futuro que se define para avanzar hacia la obtención de un estudio más detallado o en pro de resultados considerando aspectos que no fueron abordados en este proyecto. Finalmente, se proponen las apreciaciones personales del autor con respecto al estudio propuesto y al desarrollo del proyecto.

4.1 OBJETIVOS

En esta sección se presentan las conclusiones respectivas a los objetivos planteados que guían el desarrollo de este proyecto. A continuación analizaremos el objetivo general del proyecto sintetizando de forma integral los pasos necesarios para alcanzarlo.

Objetivo general

A continuación se presenta el objetivo general que guía las pautas de este proyecto de investigación, el cual es:

1. Dar a conocer cómo los contratos de software han cambiado la esquematización de los contratos convencionales solo por el hecho de incorporar en su estructuración productos intangibles y no especificados completamente.

En función de este objetivo, complementado con la motivación y el levantamiento del estado del arte realizado en este proyecto, se definen tres preguntas de investigación respecto de la celebración de contratos de desarrollo de software: ¿son los contratos de software de la misma índole que un contrato convencional?, ¿de qué tipo de contrato se trata? ¿deben ser estos contratos elaborados por quien conoce de leyes, quien conoce de software o ambos especialistas?

En base a estas preguntas, considerando la motivación planteada, se determina una hipótesis que componen el marco de trabajo de este estudio, la cual es:

H: Si los contratos de desarrollo y/o implementación de software fueran elaborados por especialistas legales y técnicos se disminuiría el riesgo de fracaso en la entrega final del producto y cumplimiento efectivo del contrato celebrado.

Luego, realizando la ejecución del diseño experimental se obtienen los datos necesarios para someter a prueba la hipótesis en función de los análisis considerados.

Finalmente, se realiza un prototipo de plantilla para toma de requerimientos funcionales como no funcionales, centrándose en las controversias observadas.

Los resultados de dicha investigación determinaron que en términos de toma de requerimientos mediante una plantilla pre-elaborada favorece el entendimiento del problema y garantiza de mejor manera el cumplimiento de la funcionalidad, dado que se obtienen las pruebas de aprobación y de rechazo de cada una de los requerimientos.

4.2 IMPLICANCIAS TEÓRICAS Y PRÁCTICAS

En esta sección se presentan las implicancias e impactos teóricos y prácticos asociados a los resultados y productos obtenidos en función de la ejecución de este proyecto. Esto considerando la implicancia en relación al sistema de evaluación desarrollado y también en base a los resultados obtenidos en función del análisis de los datos recolectados.

4.2.1 Implicancias teóricas

Dentro de las implicancias teóricas que podemos obtener de la investigación desarrollada, son principalmente orientadas a temáticas sobre posibles aspectos legales que podrían regular el proceso de contratación y/o asesoramiento técnico en áreas de la ingeniería de software.

4.2.2 Implicancias prácticas

Dentro de las implicancias prácticas que obtenemos de la investigación desarrollada, podemos mencionar:

- Definiciones claras de las funcionalidades
- Trazabilidad de tiempos
- Pruebas de aceptación establecidas
- Control de cambios

4.3 TRABAJO FUTURO

En función del trabajo realizado se identifican diversas consideraciones para el desarrollo de potenciales trabajo en función de este. Luego, se estructura esta sección en los trabajos futuros a realizar producto del software desarrollado como herramienta para el apoyo de la evaluación propuesta en este proyecto y los trabajos derivados del diseño experimental propuesto.

4.3.1 Trabajo futuro en la ingeniería de requerimientos

Con respecto a la ingeniería de requerimientos propuesta se considera la aplicación de los siguientes trabajos futuros:

Relacionados con las personas involucradas

Las vías que pueden dificultar la determinación de los requisitos son:

- Los usuarios no tienen claro lo que desean
- Los usuarios no se involucran en la elaboración de requisitos escritos
- Los usuarios insisten en nuevos requisitos después de que el coste y la programación se hayan fijado
- La comunicación con los usuarios es lenta
- Los usuarios no participan en revisiones o son incapaces de hacerlo.
- Los usuarios no comprenden los problemas técnicos
- Los usuarios no entienden el proceso del desarrollo

Esto puede conducir a la situación donde las exigencias del consumidor cambian, incluso cuando el desarrollo del producto ya está en marcha, esta situación es tan común en el día a día de los proyectos de ingeniería de software, que existe una ingeniería del cambio de requerimientos para abordar de mejor manera estas exigencias de parte del consumidor.

Relacionados con los analistas

La correcta redacción de las especificaciones de requisitos del software es imprescindible para el correcto desarrollo del proyecto. Por ello, en su redacción hay que evitar:

- Uso de terminología ambigua en la redacción de los documentos de requisitos
- Sobre-especificación de los requisitos

- Escritura poco legible, voz pasiva, abuso de negaciones
- Uso de verbos en condicional, expresiones subjetivas
- Ausencia de términos y verbos del dominio de la aplicación

Relacionados con los desarrolladores

Los problemas posibles causados por los desarrolladores durante análisis de requisitos son:

- El personal técnico y los usuarios finales pueden tener diversos vocabularios y pueden llegar a creer incorrectamente que están de acuerdo, no dándose cuenta del desacuerdo hasta que se provee el producto final
- Los desarrolladores pueden intentar encajar el sistema en un modelo existente, en vez de desarrollar un sistema adaptado a las necesidades del cliente
- El análisis de requisitos se puede realizar a menudo por los ingenieros o programadores, en vez de personal con las habilidades de relación con la gente y el conocimiento apropiados para entender las necesidades de un cliente correctamente.

4.3.2 Trabajo en la línea de investigación

Considerando los fenómenos observados y las conclusiones desarrolladas en base a los resultados obtenidos, es posible indagar en mayor medida en las preguntas de investigación que guían y soportan el estudio propuesto.

En base a lo anterior, se propone realizar una ampliación a las plantillas de requerimientos funcionales y no funcionales para garantizar una mejor abstracción de las necesidades del cliente y, por consecuencia, conseguir un análisis más detallado de las necesidad del cliente.

Adicional a lo anterior, se propone realizar un anexo técnico con aspectos regulatorios en el ámbito del cumplimiento de los requerimientos, indicando el tipo de pruebas que se realizarán y bajo qué tipo de ambiente se encontrará.

BIBLIOGRAFÍA

Abdelouahab, Z. y Carvalho, M. (2001). **Um método para elicitação e Modelagem de Requisitos Baseado em Objetivos**, http://www.inf.puc-rio.br/~wer01/Eli-Req-5.pdf.

Álvarez G. (2003), **Curso de Investigación Jurídica**, Editorial Lexis Nexis.

Ambler, S. (1995). **Using Use-Cases, Software Development. páginas 53-61**.

Arlow, J. y Neustadt, I. (2002). **UML and the Unified Process**. Editorial Addison-Wesley.

Balestrini M. (2006). **Cómo se elabora el Proyecto de Investigación (Para los Estudios Formulativos o Exploratorios, Descriptivos, Diagnósticos, Evaluativos, Formulación de Hipótesis Causales, Experimentales y los Proyectos Factibles)**. Editorial BL Consultores Asociados.

Budd, T. (1996). **An Introduction to Object-Oriented Programming, segunda edición**. Editorial Addison-Wesley.

Cashman, M. (1989). **Object Oriented Domain Analysis**. ACM Software Engineering Notes, vol. 14, núm. 6, p. 67.

Coad, P. y Yourdon, E. (1991). **Object-Oriented Analysis, segunda edición**. Editorial Prentice Hall.

Cockburn, A. (2001). **Writing Effective Use-Cases**. Editorial Addison-Wesley.

Congreso de la República de Chile, (Código Civil) **Código Civil**, Banco del Congreso Nacional de Chile.

Congreso de la República de Chile, (Ley 17.336), **Ley de Propiedad Intelectual**, Banco del Congreso Nacional de Chile.

Congreso de la República de Chile, (Ley 19.039), **Ley de Propiedad Industrial**, Banco del Congreso Nacional de Chile.

Christel, M y Kang, K. (1992). **Issues in Requirements Elicitation**, Software Engineering Institute.

DeMarco, T. (1979). **Structured Analysis and System Specification**. Editorial Prentice Hall.

Elgueta, M. , Palma, E. (2010), **La Investigación en Ciencias Sociales y Jurídicas**, Editorial ORION Colección Juristas Chilenos.

Eurostat, OCDE. (2005), **Oslo Manual: Guidelines for Collecting and Interpreting Innovation Data, Third edition**, OCDE/European Communities.

Finol, M y Camacho, H. (2008), **El proceso de investigación científica, Segunda edición**. Editorial McGraw-Hill.

Firesmith, D. (1993). **Object-Oriented Requirements Analysis and Logical Design**. Editorial Wiley.

Fitzpatrick, E. (2006). **Prevención de conflictos en la integración e implementación de contratos informáticos**, Revista Chilena de Derecho Informático.

Gallego, F., Villagrá, C., Satorre, R., Compañ, P., Molina, R., & Llorens Largo, F. (2014). **Panorámica: Serious games, gamification y mucho más**. *ReVisión*, *7*(2).

Gause, D. y Weinberg, G. (1989) **Exploring Requirements: Quality Before Design**, Editorial Dorset House.

Hernández, R., Fernández, C., Baptista, P. & Casas, M. (1998/9). **Metodología de la Investigación**, Editorial McGraw-Hill.

Naur, P. y Randall, B. (1969), **Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee**, Editorial NATO.

Silva, P. (2003). **Autonomía de la voluntad, contratación electrónica y protección del consumidor**. Revista Chilena de Derecho Informático.

Somerville, I. (2015). **Ingeniería de software**. Editorial Pearson.

Somerville, I. y Sawyer, P. (1997). **Requirements Engineering**, Editorial Wiley.

Standish Group (2016). **CHAOS Report 2016**. Editorial Standish Group.

Young R. (2001). **Effective Requirements Practices**. Editorial Addison-Wesley.

Zahniser, R. (1990). **Building Software in Groups**, Editorial American Programmer, vol. 3, núms. 7-8.

ANEXO A DOCUMENTACIÓN TÉCNICA

Las revisiones del software son un "filtro" para el proceso del software. Es decir, se aplican en varios puntos durante la ingeniería de software y sirven para descubrir errores y defectos a fin de poder eliminarlos. Las revisiones del software "purifican" los productos del trabajo de la ingeniería de software, incluso los modelos de requerimientos y diseño, código y datos de prueba. Freedman y Weinberg analizan del modo siguiente la necesidad de hacer revisiones:

El trabajo técnico necesita las revisiones por la misma razón que los lápices necesitan borradores: errar es humano. La segunda razón por la que son necesarias las revisiones técnicas es porque, si bien las personas son buenas para detectar algunos de sus propios errores, muchas clases de ellos pasan desapercibidos con más facilidad para quien los comete que para otras personas. Por tanto, este proceso de revisión es la respuesta a la oración de Robert Burns:

Oh, quiera algún Dios el regalo darnos de vernos a nosotros como los demás nos ven

Una revisión cualquiera es una forma de utilizar la diversidad de un grupo para lo siguiente:

1. Resaltar las mejoras necesarias en el producto que elaboró una sola persona o equipo;
2. Confirme aquellas partes de un producto en las que no se desea o no se necesita hacer una mejora;
3. Realice el trabajo técnico de calidad más uniforme, o al menos más predecible, que pueda lograrse sin hacer revisiones, a fin de que el trabajo técnico sea más manejable.

Como parte de la ingeniería de software, pueden realizarse muchos diferentes tipos de revisiones. Cada uno tiene su lugar. Una reunión informal alrededor de la máquina del café es una forma de revisión si se analizan problemas técnicos. La presentación formal de la arquitectura del software a un público de clientes, administradores y técnicos también es una forma de revisión. Sin embargo, en este libro nos centramos en las revisiones técnicas o por pares, ejemplificadas por las revisiones casuales, walkthroughs e inspecciones.

A.1 ANEXO TÉCNICO PARA TOMA DE REQUERIMIENTOS

<table>
<tr><td>Requerimiento funcional</td><td>< Nombre descriptivo del requerimiento funcional ></td></tr>
<tr><td>Versión</td><td>< Número de la versión actual del requerimiento indicando la fecha del registro ></td></tr>
<tr><td>Autor</td><td>< Nombre de los autores involucrados en la generación del requerimiento ></td></tr>
<tr><td>Fuente</td><td>< De donde se obtuvo referencias sobre el requerimiento (empresa, expertos, libros, etc) ></td></tr>
<tr><td>Propósito</td><td>< Propósito de la realización del requerimiento funcional dentro del sistema ></td></tr>
<tr><td>Descripción</td><td>< Detalle de la funcionalidad del requerimiento funcional ></td></tr>
<tr><td>Alcance y nivel</td><td>< Hasta qué nivel se desarrollará el requerimiento y cuál será su alcance dentro del proyecto ></td></tr>
<tr><td>Condiciones de aprobación</td><td>< Condiciones con las cuales se dará por aprobado el requerimiento funcional ></td></tr>
<tr><td>Condiciones de rechazo</td><td>< Condiciones con las cuales se rechazará el requerimiento funcional ></td></tr>
<tr><td>Actores</td><td>< Actores involucrados con este requerimiento y sus permisos correspondientes ></td></tr>
<tr><td>Evento disparador</td><td>< Evento o acción que da inicio este requerimiento funcional ></td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><th>Paso</th><th>Acción</th></tr>
<tr><td>1</td><td></td></tr>
<tr><td>2</td><td></td></tr>
<tr><td>3</td><td></td></tr>
<tr><td>...</td><td></td></tr>
<tr><td>1000</td><td></td></tr>
</table>
</td></tr>
<tr><td>Condición posterior</td><td>< Condición posterior del caso de uso del requerimiento funcional ></td></tr>
</table>

<table>
<tr><td>Excepciones y extensiones</td><td>
<table>
<tr><th>Paso</th><th>Acción</th></tr>
<tr><td>1</td><td></td></tr>
<tr><td>2</td><td></td></tr>
<tr><td>3</td><td></td></tr>
<tr><td>...</td><td></td></tr>
<tr><td>1000</td><td></td></tr>
</table>
</td></tr>
<tr><td colspan="2">INFORMACIÓN RELACIONADA</td></tr>
<tr><td>Prioridad</td><td>< Grado de prioridad del requerimiento funcional dentro del desarrollo del sistema > (Crítico, Alto Medio, Bajo)</td></tr>
<tr><td>Frecuencia</td><td>< Cantidad de veces que este requerimiento será utilizado en el sistema ></td></tr>
<tr><td>Desempeño</td><td>
<table>
<tr><th>Acción</th><th>Medida</th></tr>
<tr><td>1</td><td></td></tr>
<tr><td>2</td><td></td></tr>
<tr><td>3</td><td></td></tr>
<tr><td>...</td><td></td></tr>
<tr><td>1000</td><td></td></tr>
</table>
</td></tr>
<tr><td>Canales hacia los actores</td><td>< Archivos, interactivos, bases de datos, etc ></td></tr>
<tr><td>Características abiertas</td><td>< Listado de características que pueden afectar las decisiones sobre el caso de uso ></td></tr>
<tr><td>Requerimientos superiores</td><td>< Listado de requerimientos que ocupan este requerimiento funcional ></td></tr>
<tr><td>Requerimientos inferiores</td><td>< Listado de requerimientos que se ocupan en este requerimiento funcional ></td></tr>
<tr><td>Comentarios</td><td>< Observaciones adicionales sobre el requerimiento funcional ></td></tr>
</table>

A.2 PRUEBAS PARA APLICACIONES WEB

Existe una urgencia que siempre impregna un proyecto web. Los participantes (intranquilos por la competencia de otras webapps, presionados por las demandas del cliente y preocupados porque perderán la ventana de mercado) fuerzan para poner la webapp en línea. Como consecuencia, en ocasiones desechan por completo las actividades técnicas que frecuentemente ocurren tarde en el proceso, como las pruebas de la aplicación web. Esto puede ser un error catastrófico. Para evitarlo, los miembros del equipo deben asegurarse de que cada producto resultante muestre alta calidad.

Los modelos de requerimientos y de diseño no pueden probarse en el sentido clásico: por ello, el equipo debe realizar revisiones técnicas y pruebas ejecutables. La intención es descubrir y corregir errores antes de que la webapp esté disponible para sus usuarios finales.

A.2.1 Conceptos de pruebas para aplicaciones web

Probar es el proceso de ejecución del software con la intención de encontrar (y a final de cuentas corregir) errores. Los sistemas y las aplicaciones basadas en web residen en una red e interactúan con muchos sistemas operativos, navegadores (residentes en varios dispositivos), plataformas de hardware, protocolos de comunicaciones y aplicaciones "de cuarto trasero" diferentes, la búsqueda de errores representa un reto significativo. Para entender los objetivos de las pruebas dentro de un contexto de ingeniería web, debemos considerar las diversas dimensiones de calidad de la webapp. En el contexto de esta discusión, se consideran las dimensiones de calidad que son particularmente relevantes en cualquier análisis de las pruebas de la webapp. También se considera la naturaleza de los errores que se encuentran como consecuencia de las pruebas y la estrategia de prueba que se aplica para descubrir dichos errores.

Dimensiones de calidad

La calidad se incorpora en una aplicación web como consecuencia de un buen diseño. Se evalúa aplicando una serie de revisiones técnicas que valoran varios elementos del modelo de diseño y un proceso de prueba que se estudia a lo largo de este anexo. Tanto las revisiones como las pruebas examinan una o más de las siguientes dimensiones de calidad:

- El contenido se evalúa tanto en el nivel sintáctico como en el semántico. En el primero, se valora vocabulario, puntuación y gramática para documentos basados en texto. En el segundo, se valora la corrección (de la información presentada), la consistencia (a través de todo el objeto de contenido y de los objetos relacionados) y la falta de ambigüedad.
- La función se prueba para descubrir errores que indican falta de conformidad con los requerimientos del cliente. Cada función de la webapp se valora en su corrección, inestabilidad y conformidad general con estándares de implantación adecuados (por ejemplo, estándares de lenguaje Java o AJAX).
- La estructura se valora para garantizar que entrega adecuadamente el contenido y la función de la aplicación, que es extensible y que puede soportarse conforme se agregue nuevo contenido o funcionalidad.
- La usabilidad se prueba para asegurar que la interfaz soporta a cada categoría de usuario y que puede aprender y aplicar toda la sintaxis y semántica de navegación requerida.
- La navegabilidad se prueba para asegurar que toda la sintaxis y la semántica de navegación se ejecutan para descubrir cualquier error de navegación (por ejemplo, vínculos muertos, inadecuados y erróneos).
- El rendimiento se prueba bajo condiciones operativas, configuraciones y cargas diferentes a fin de asegurar que el sistema responde a la interacción con el usuario y que maneja la carga extrema sin degradación operativa inaceptable.
- La compatibilidad se prueba al ejecutar la webapp en varias configuraciones anfitrión, tanto en el cliente como en el servidor. La intención es encontrar errores que sean específicos de una configuración anfitrión única.
- La interoperabilidad se prueba para garantizar que la webapp tiene interfaz adecuada con otras aplicaciones y/o bases de datos.
- La seguridad se prueba al valorar las vulnerabilidades potenciales e intenta explotar cada una. Cualquier intento de penetración exitoso se estima como un fallo de seguridad.

Errores dentro de un entorno de webapp

Los errores que se encuentran como consecuencia de una prueba exitosa de una webapp tienen algunas características únicas:

1. Puesto que muchos tipos de pruebas de webapps descubren problemas que se evidencian primero en el lado del cliente (es decir, mediante una interfaz implementada en un navegador específico o en un dispositivo de comunicación personal), con frecuencia se ve un síntoma del error, no el error en sí.
2. Puesto que una webapp se implementa en algunas configuraciones distintas y dentro de diferentes entornos, puede ser difícil o imposible reproducir un error afuera del entorno en el que originalmente se encontró.
3. Aunque algunos errores son resultado de diseño incorrecto o codificación HTML (u otro lenguaje de programación) impropia, muchos errores pueden rastrearse en la configuración de la webapp.
4. Dado que las webapps residen dentro de una arquitectura cliente-servidor, los errores pueden ser difíciles de rastrear a través de tres capas arquitectónicas: el cliente, el servidor o la red en sí.
5. Algunos errores se deben al entorno operativo estático (es decir, a la configuración específica donde se realiza la prueba), mientras que otros son atribuibles al entorno operativo dinámico (es decir, a la carga de recurso instantánea o a errores relacionados con el tiempo).

Estos cinco atributos de error sugieren que el entorno juega un importante papel en el diagnóstico de todos los errores descubiertos durante la prueba de webapps. En algunas situaciones (por ejemplo, la prueba de contenido), el sitio del error es obvio, pero en muchos otros tipos de prueba de webapps (por ejemplo, prueba de navegación, prueba de rendimiento, prueba de seguridad), la causa subyacente del error puede ser considerablemente más difícil de determinar.

Estrategia de las pruebas

La estrategia para probar webapps adopta los principios básicos de todas las pruebas de software y aplica una estrategia y las tácticas que se recomendaron para los sistemas orientados a objetos. Los siguientes pasos resumen el enfoque:

1. El modelo de contenido para la webapp a se revisa a fin de descubrir errores.
2. El modelo de interfaz se examina para garantizar que todos los casos de uso pueden alojarse.
3. El modelo de diseño para la webapp se revisa para descubrir errores de navegación.

4. La interfaz de usuario se prueba para descubrir errores en la mecánica de presentación y/o navegación.
5. Los componentes funcionales se someten a prueba de unidad.
6. Se prueba la navegación a lo largo de toda la arquitectura.
7. La webapp se implanta en varias configuraciones de entorno diferentes y se prueba para asegurar la compatibilidad con cada configuración.
8. Las pruebas de seguridad se realizan con la intención de explotar las vulnerabilidades en la webapp o dentro de su entorno.
9. Se realizan pruebas de rendimiento.
10. La webapp se prueba con una población controlada y monitoreada de usuarios finales; los resultados de su interacción con el sistema se evalúan para detectar errores de contenido y de navegación, preocupaciones de usabilidad y compatibilidad, y seguridad, confiabilidad y rendimiento de la webapp.

Puesto que muchas webapps evolucionan continuamente, el proceso de prueba es una actividad siempre en marcha que realiza el personal de apoyo web, quien usa pruebas de regresión derivadas de las pruebas desarrolladas cuando comenzó la ingeniería de las webapps.

Planificación de pruebas

El uso de la palabra planificación (en cualquier contexto) es un anatema para algunos desarrolladores web que no planifican; sólo arrancan, con la esperanza de que surja una webapp asesina. Un enfoque más disciplinado reconoce que la planificación establece un mapa de ruta para todo el trabajo que va después. Vale la pena el esfuerzo.

Las preguntas que deben plantearse son: ¿cómo se "idean nuevas pruebas imaginativas" y sobre qué deben enfocarse dichas pruebas? Las respuestas a estas preguntas se integran en un plan de prueba que identifica: 1) el conjunto de tareas que se van a aplicar cuando comiencen las pruebas, 2) los productos de trabajo que se van a producir conforme se ejecuta cada tarea de prueba y 3) la forma en la que se evalúan, registran y reutilizan los resultados de la prueba cuando se realizan pruebas de regresión. En algunos casos, el plan de prueba se integra con el plan del proyecto. En otros, es un documento separado.

A.2.2 Pruebas de contenido

Los errores en el contenido de la webapp pueden ser tan triviales como errores tipográficos menores o tan significativos como información incorrecta, organización inadecuada o violación de leyes de la propiedad intelectual. La prueba de contenido intenta descubrir éstos y muchos otros problemas antes de que el usuario los encuentre. La prueba de contenido combina tanto revisiones como generación de casos de prueba ejecutables. Las revisiones se aplican para descubrir errores semánticos en el contenido. Las pruebas ejecutables se usan para descubrir errores de contenido que puedan rastrearse a fin de derivar dinámicamente contenido que se impulse por los datos adquiridos de una o más bases de datos.

Objetivos de la prueba de contenido

La prueba de contenido tiene tres objetivos importantes: 1) descubrir errores sintácticos (por ejemplo, errores tipográficos o gramaticales) en documentos de texto, representaciones gráficas y otros medios; 2) descubrir errores semánticos (es decir, errores en la precisión o completitud de la información) en cualquier objeto de contenido que se presente conforme ocurre la navegación y 3) encontrar errores en la organización o estructura del contenido que se presenta al usuario final.

Para lograr el primer objetivo, pueden usarse correctores automáticos de vocabulario y gramática. Sin embargo, muchos errores sintácticos evaden la detección de tales herramientas y los debe descubrir un revisor humano (examinador). De hecho, un sitio web grande debe considerar los servicios de un editor profesional para descubrir errores tipográficos, gazapos gramaticales, errores en la consistencia del contenido, errores en las representaciones gráficas y en referencias cruzadas.

La prueba semántica se enfoca en la información presentada dentro de cada objeto de contenido. El revisor (examinador) debe responder las siguientes preguntas:

- ¿La información realmente es precisa?
- ¿La información es concisa y puntual?
- ¿La plantilla del objeto de contenido es fácil de comprender para el usuario?
- ¿La información incrustada dentro de un objeto de contenido puede encontrarse con facilidad?

- ¿Se proporcionaron referencias adecuadas para toda la información derivada de otras fuentes?
- ¿La información presentada es consistente internamente y con la información presentada en otros objetos de contenido?
- ¿El contenido es ofensivo, confuso o abre la puerta a demandas?
- ¿El contenido infringe derechos de autor o nombres comerciales existentes?
- ¿El contenido incluye vínculos internos que complementan el contenido existente? ¿Los vínculos son correctos?
- ¿El estilo estético del contenido entra en conflicto con el estilo estético de la interfaz?

Obtener respuestas a cada una de estas preguntas para una gran webapp (que contiene cientos de objetos de contenido) puede ser una tarea atemorizante. Sin embargo, el fracaso para descubrir los errores semánticos sacudirá la fe del usuario en la webapp y puede conducir al fracaso de la aplicación basada en web.

Los objetos de contenido existen dentro de una arquitectura que tiene un estilo específico. Durante la prueba de contenido, la estructura y organización de la arquitectura de contenido se prueba para garantizar que el contenido requerido se presente al usuario final en el orden y relaciones adecuados. Por ejemplo, la webapp CasaSeguraAsegurada.com presenta información variada acerca de los sensores que se utilizan como parte de los productos de seguridad y vigilancia. Los objetos de contenido proporcionan información descriptiva, especificaciones técnicas, una representación fotográfica e información relacionada. Las pruebas de la arquitectura de contenido de CasaSeguraAsegurada.com luchan por descubrir errores en la presentación de esta información (por ejemplo, una descripción del sensor X se presenta con una fotografía del sensor Y).

Prueba de base de datos

Las webapps modernas hacen mucho más que presentar objetos de contenido estáticos. En muchos dominios de aplicación, la webapp tiene interfaz con sofisticados sistemas de gestión de base de datos y construyen objetos de contenido dinámico que se crean en tiempo real, usando los datos adquiridos desde una base de datos.

Por ejemplo, una webapp de servicios financieros puede producir información compleja basada en texto, tablas tabulares y gráficas acerca de un fondo específico (por ejemplo, una

acción o fondo mutualista). El objeto de contenido compuesto que presenta esta información se crea de manera dinámica después de que el usuario hace una solicitud de información acerca de un fondo específico. Para lograrlo, se requieren los siguientes pasos: 1) consulta a una gran base de datos de fondos, 2) extracción de datos relevantes de la base de datos, 3) organización de los datos extraídos como un objeto de contenido y 4) transmisión de este objeto de contenido (que representa información personalizada que requiere un usuario final) al entorno del cliente para su despliegue. Los errores pueden ocurrir, y ocurren, como consecuencia de cada uno de estos pasos. El objeto de la prueba de la base de datos es descubrir dichos errores, pero esta prueba es complicada por varios factores:

1. El lado cliente original solicita información que rara vez se presenta en la forma [por ejemplo, lenguaje de consulta estructurado (SQL)] en la que puede ingresarse a un sistema de gestión de base de datos (DBMS). Por tanto, las pruebas deben diseñarse para descubrir errores cometidos al traducir la solicitud del usuario de manera que pueda procesar el DBMS.
2. La base de datos puede ser remota en relación con el servidor que alberga la webapp. En consecuencia, deben desarrollarse pruebas que descubran errores en la comunicación entre la webapp y la base de datos remota[28].
3. Los datos brutos adquiridos de la base de datos deben transmitirse al servidor de la webapp y formatearse de manera adecuada para su posterior transmisión al cliente. Por tanto, deben desarrollarse pruebas que demuestren la validez de los datos brutos recibidos por el servidor de la webapp y también deben crearse pruebas adicionales que demuestren la validez de las transformaciones aplicadas a los datos brutos para crear objetos de contenido válidos.
4. El objeto de contenido dinámico debe transmitirse al cliente de forma que pueda desplegarse al usuario final. Por ende, debe diseñarse una serie de pruebas para 1) descubrir errores en el formato del objeto de contenido y 2) probar la compatibilidad con diferentes configuraciones del entorno del cliente.

Al considerar estos cuatro factores, los métodos de diseño de caso de prueba deben aplicarse a cada una de las “capas de interacción” que se mencionan en la figura A-01. Las pruebas deben garantizar que 1) información válida pasa entre el cliente y el servidor desde

[28] Estos datos pueden volverse complejos cuando se encuentran bases de datos distribuidas o cuando se requiere el acceso a un almacén de datos.

la capa interfaz, 2) la webapp procesa los guiones de manera correcta y extrae o formatea adecuadamente los datos del usuario, 3) los datos del usuario pasan correctamente a una función de transformación de datos del lado servidor que formatea consultas adecuadas (por ejemplo, SQL) y 4) las consultas pasan a una capa de gestión de datos que se comunica con las rutinas de acceso a la base de datos (potencialmente ubicadas en otra máquina).

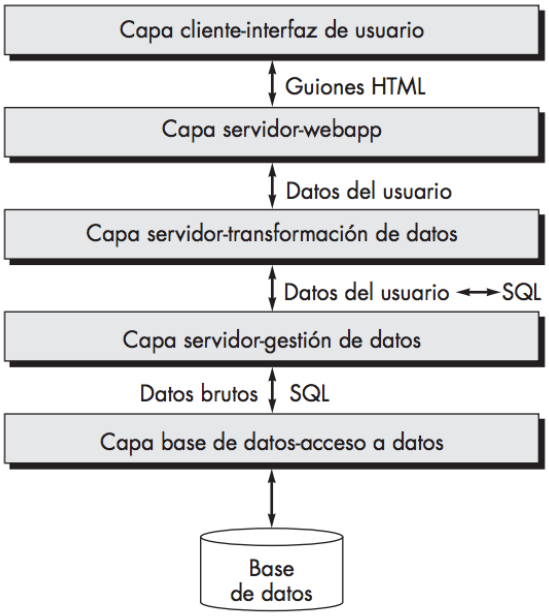


Figura N°A2-01 - Capas de interacción

Las capas de transformación de datos, de gestión de datos y de acceso a base de datos que se muestran en la figura A2-01, con frecuencia se construyen con componentes reutilizables que se validaron por separado y como paquete. Si éste es el caso, la prueba de webapps se enfoca en el diseño de casos de prueba para ejercitar las interacciones entre la capa cliente y las primeras dos capas servidor (webapp y transformación de datos) que se muestran en la figura.

La capa de interfaz de usuario se prueba para garantizar que los guiones se construyeron de manera adecuada para cada consulta de usuario y que transmiten adecuadamente al lado servidor. La capa webapp en el lado servidor se prueba para asegurar que los datos de usuario se extraen de manera adecuada de los guiones y que se transmite adecuadamente a la capa de transformación de datos en el lado servidor. Las funciones de transformación

de datos se prueban para asegurar que se creó el SQL correcto y que pasó a componentes de gestión de datos adecuados.

A.2.3 Pruebas de interfaces de usuario

La verificación y validación de una interfaz de usuario de webapp ocurre en tres puntos distintos. Durante el análisis de requerimientos, el modelo de interfaz se revisa para garantizar que se da conformidad a los requerimientos de los participantes y a otros elementos del modelo de requerimientos. Durante el diseño, se revisa el modelo de diseño de interfaz para garantizar que se logran los criterios de calidad genéricos establecidos para todas las interfaces de usuario y que los temas de diseño de interfaz específicos de la aplicación se abordaron de manera adecuada. Durante la prueba, la atención se centra en la ejecución de aspectos específicos de la aplicación de la interacción con el usuario, conforme se manifiesten por la sintaxis y la semántica de la interfaz. Además, la prueba proporciona una valoración final de la usabilidad.

Estrategia de prueba de interfaz

La prueba de interfaz ejercita los mecanismos de interacción y valida los aspectos estéticos de la interfaz de usuario. La estrategia global para la prueba de interfaz es 1) descubrir errores relacionados con mecanismos de interfaz específicos (por ejemplo, en la ejecución adecuada de un vínculo de menú o en la forma como entran los datos en un formulario) y 2) descubrir errores en la forma como la interfaz implementa la semántica de navegación, la funcionalidad de la webapp o el despliegue de contenido. Para lograr esta estrategia, se inician algunos pasos tácticos:

- Las características de la interfaz se prueban para garantizar que las reglas del diseño, estética y contenido visual relacionado estén disponibles sin error para el usuario. Las características incluyen tipo de fuente, uso de color, marcos, imágenes, bordes, tablas y características de interfaz relacionadas que se generan conforme avanza la ejecución de la webapp.
- Los mecanismos de interfaz individuales se prueban en forma análoga a la prueba de unidad. Por ejemplo, las pruebas se diseñan para ejercitar todas las formas, guiones del lado cliente, HTML dinámicos, guiones, contenido de streaming (transmisión continua) y mecanismos de interfaz específicos de la aplicación (por ejemplo, un carro de mandado para una aplicación de comercio electrónico). En muchos casos, la

prueba puede enfocarse exclusivamente en uno de estos mecanismos (la "unidad") y excluir otras características y funciones de interfaz.

- Cada mecanismo de interfaz se prueba dentro del contexto de un caso de uso o de una unidad semántica de navegación (USN) para una categoría de usuario específica. Este enfoque de pruebas es análogo a la prueba de integración porque las pruebas se realizan conforme los mecanismos de interfaz se integran para permitir la ejecución de un caso de uso o USN.
- La interfaz completa se prueba contra los casos de uso seleccionados y las USN a fin de descubrir errores en la semántica de la interfaz. Este enfoque de prueba es análogo a la prueba de validación porque el propósito es demostrar conformidad con la semántica de casos de uso o USN específicas. En esta etapa se lleva a cabo una serie de pruebas de usabilidad.
- La interfaz se prueba dentro de varios entornos (por ejemplo, navegadores) para garantizar que será compatible. En realidad, esta serie de pruebas también puede considerarse como parte de las pruebas de configuración.

Prueba de mecanismos de interfaz

Cuando un usuario interactúa con una webapp, la interacción ocurre a través de uno o más mecanismos de interfaz. En los párrafos que siguen se presenta un breve panorama de las consideraciones de prueba para cada mecanismo de interfaz.

- **Vínculos:** Cada vínculo de navegación se prueba para garantizar que se alcanza el objetivo de contenido o función apropiados. Se construye una lista de todos los vínculos asociados con la plantilla de interfaz (por ejemplo, barras de menú e ítems de índice) y luego se ejecuta cada uno individualmente. Además, deben ejercitarse los vínculos dentro de cada objeto de contenido para descubrir URL o vínculos defectuosos con objetos de contenido o funciones inadecuadas. Finalmente, los vínculos con webapps externas deben probarse en su precisión y también evaluarse para determinar el riesgo de que se vuelvan inválidos con el tiempo.
- **Formularios:** En un nivel macroscópico, las pruebas se realizan para asegurarse de que 1) las etiquetas identifican correctamente los campos dentro del formulario y los campos obligatorios se identifican visualmente para el usuario, 2) el servidor recibe toda la información contenida dentro del formulario y ningún dato se pierde en la transmisión entre cliente y servidor, 3) se usan valores por defecto adecuados

cuando el usuario no selecciona de un menú desplegable o conjunto de botones, 4) las funciones del navegador (por ejemplo, la flecha "retroceso") no corrompen la entrada de datos en un formulario y 5) los guiones que realizan la comprobación de errores en los datos ingresados funcionan de manera adecuada y proporcionan mensajes de error significativos. En un nivel más dirigido, las pruebas deben garantizar que 1) los campos del formulario tienen ancho y tipos de datos adecuados, 2) el formulario establece salvaguardas adecuadas que prohíben que el usuario ingrese cadenas de texto más largas que cierto máximo predefinido, 3) todas las opciones adecuadas para menús desplegables se especifican y ordenan en forma significativa para el usuario final, 4) las características de "autollenado" del navegador no conducen a errores en la entrada de datos y 5) la tecla de tabulación (o alguna otra) inicia el movimiento adecuado entre los campos del formulario.

- **Guión en el lado cliente:** Las pruebas de caja negra se realizan para descubrir cualquier error en el procesamiento conforme se ejecuta el guión. Estas pruebas con frecuencia se acoplan con pruebas de formularios porque la entrada del guión con frecuencia se deriva de los datos proporcionados como parte del procesamiento de formulario. Debe realizarse una prueba de compatibilidad para garantizar que el lenguaje del guión elegido funcionará adecuadamente en las configuraciones de entorno que soporten la webapp. Además de probar el guión en sí, se "debe asegurarse de que los estándares [de webapps] de la compañía enuncien el lenguaje y versión preferidos del lenguaje de guión que se va a usar para la escritura de guiones en el lado cliente (y en el lado servidor)".
- **HTML dinámico:** Cada página web que contenga HTML dinámico se ejecuta para asegurar que el despliegue dinámico es correcto. Además, debe llevarse a cabo una prueba de compatibilidad para asegurarse que el HTML dinámico funciona adecuadamente en las configuraciones de entorno que soportan la webapp.
- **Ventanas pop-up:** Una serie de pruebas garantiza que 1) la aparición instantánea tiene el tamaño y posición adecuadas, 2) la aparición no cubre la ventana de la webapp original, 3) el diseño estético de la aparición es consistente con el diseño estético de la interfaz y 4) las barras de desplazamiento y otros mecanismos de control anexados a la ventana de aparición se ubican y funcionan de manera adecuada, como se requiere.
- **Guiones CGI:** Las pruebas de caja negra se realizan con énfasis sobre la integridad de los datos (conforme los datos pasan al guión CGI) y del procesamiento del guión (una

vez recibidos los datos validados). Además, la prueba de rendimiento puede realizarse para garantizar que la configuración del lado servidor puede alojar las demandas de procesamiento de múltiples invocaciones de los guiones CGI.

- **Contenido de streaming:** Las pruebas deben demostrar que los datos de streaming están actualizados, que se despliegan de manera adecuada y que pueden suspenderse sin error y reanudarse sin dificultad.
- **Cookies:** Se requieren pruebas tanto del lado servidor como del lado cliente. En el primero, las pruebas deben garantizar que una cookie se construyó adecuadamente (que contiene datos correctos) y que se transmitió de manera adecuada al lado cliente cuando se solicitó contenido o funcionalidad específico. Además, la persistencia adecuada de la cookie se prueba para asegurar que su fecha de expiración es correcta. En el lado cliente, las pruebas determinan si la webapp liga adecuadamente las cookies existentes a una solicitud específica (enviada al servidor).
- **Mecanismos de interfaz específicos de aplicación:** Las pruebas se siguen conforme una lista de comprobación de funcionalidad y características que se definen mediante el mecanismo de interfaz. Por ejemplo, se menciona la siguiente lista de comprobación para la funcionalidad carro de compras definida para una aplicación de comercio electrónico:

  -> Prueba de frontera del número mínimo y máximo de artículos que pueden colocarse en el carro de compras.

  -> Prueba de una solicitud de "salida" para un carro de compras vacío.

  -> Prueba de borrado adecuado de un artículo del carro de compras.

  -> Prueba para determinar si una compra vacía el contenido del carro.

  -> Prueba para determinar la persistencia del contenido del carro de compras (esto debe especificarse como parte de los requerimientos del cliente).

  -> Prueba para determinar si la webapp puede recordar el contenido del carro de compras en alguna fecha futura (suponiendo que no se realizó compra alguna).

Pruebas de usabilidad

La prueba de usabilidad es similar a la de semántica de interfaz porque también evalúa el grado en el cual los usuarios pueden interactuar efectivamente con la webapp y el grado en el que la webapp guía las acciones del usuario, proporciona retroalimentación significativa y refuerza un enfoque de interacción consistente. En lugar de enfocarse atentamente en la

semántica de algún objetivo interactivo, las revisiones y pruebas de usabilidad se diseñan para determinar el grado en el cual la interfaz de la webapp facilita la vida del usuario[29].

Invariablemente, el ingeniero en software contribuirá con el diseño de las pruebas de usabilidad, pero las pruebas en sí las realizan los usuarios finales. La siguiente secuencia de pasos es aplicable para tal fin:

1. Definir un conjunto de categorías de prueba de usabilidad e identificar las metas de cada una.
2. Diseñar pruebas que permitirán la evaluación de cada meta.
3. Seleccionar a los participantes que realicen las pruebas.
4. Instrumentar la interacción de los participantes con la webapp mientras se lleva a cabo la prueba.
5. Desarrollar un mecanismo para valorar la usabilidad de la webapp.

La prueba de usabilidad puede ocurrir en varios niveles diferentes de abstracción: 1) puede valorarse la usabilidad de un mecanismo de interfaz específico (por ejemplo, un formulario), 2) puede evaluarse la usabilidad de una página web completa (que abarque mecanismos de interfaz, objetos de datos y funciones relacionadas) y 3) puede considerarse la usabilidad de la webapp completa.

El primer paso en la prueba de usabilidad es identificar un conjunto de categorías de usabilidad y establecer los objetivos de la prueba para cada categoría. Las siguientes categorías y objetivos de prueba (escritos en forma de pregunta) ilustran este enfoque:

- **Interactividad:** ¿Los mecanismos de interacción (por ejemplo, menús desplegables, botones, punteros) son fáciles de entender y usar?
- **Plantilla:** ¿Los mecanismos de navegación, contenido y funciones se colocan de forma que el usuario pueda encontrarlos rápidamente?
- **Legibilidad:** ¿El texto está bien escrito y es comprensible?8 ¿Las representaciones gráficas se entienden con facilidad?
- **Estética:** ¿La plantilla, color, fuente y características relacionadas facilitan el uso? ¿Los usuarios "se sienten cómodos" con la apariencia y el sentimiento de la webapp?

[29] En este contexto se ha usado el término amigable con el usuario. Desde luego, el problema es que la percepción de un usuario acerca de una interfaz "amigable" puede ser radicalmente diferente a la de otro.

- **Características de despliegue:** ¿La webapp usa de manera óptima el tamaño y la resolución de la pantalla?
- **Sensibilidad temporal:** ¿Las características, funciones y contenido importantes pueden usarse o adquirir en forma oportuna?
- **Personalización:** ¿La webapp se adapta a las necesidades específicas de diferentes categorías de usuario o de usuarios individuales?
- **Accesibilidad:** ¿La webapp es accesible a personas que tienen discapacidades?

Dentro de cada una de estas categorías se diseña una serie de pruebas. En algunos casos, la "prueba" puede ser una revisión visual de una página web. En otros, pueden ejecutarse de nuevo pruebas semánticas de la interfaz, pero en esta instancia las preocupaciones por la usabilidad son primordiales.

Como ejemplo, considere la valoración de usabilidad para los mecanismos de interacción e interfaz. Se sugiere que revise la siguiente lista de características de interfaz y probar la usabilidad: animación, botones, color, control, diálogo, campos, formularios, marcos, gráficos, etiquetas, vínculos, menús, mensajes, navegación, pá- ginas, selectores, texto y barras de herramientas. Conforme se valora cada característica, es calificada por los usuarios que realizan la prueba sobre una escala cualitativa. La figura A2-02 muestra un posible conjunto de "calificaciones" de valoración que pueden seleccionar los usuarios, mismas que se aplican a cada característica individualmente, a una página web completa o a la webapp como un todo.

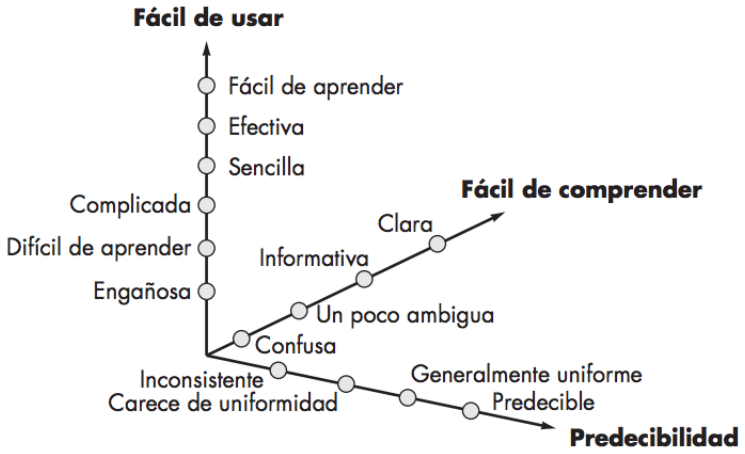


Figura N°A2-02 - Valoración cualitativa de la usabilidad

Pruebas de compatibilidad

Diferentes computadoras, dispositivos de despliegue, sistemas operativos, navegadores y velocidades de conexión de red pueden tener influencia significativa sobre la operación de una webapp. Cada configuración de cómputo puede dar como resultado diferencias en velocidades de procesamiento en el lado cliente, en resolución de despliegue y en velocidades de conexión. Los caprichos de los sistemas operativos en ocasiones pueden producir conflictos de procesamiento en la webapp. En ocasiones, diferentes navegadores producen resultados ligeramente distintos, sin importar el grado de estandarización HTML dentro de la webapp. Los plugins requeridos pueden o no conseguirse con facilidad para una configuración particular.

En algunos casos, pequeños conflictos de compatibilidad no representan problemas significativos, pero en otros pueden encontrarse serios errores. Por ejemplo, las velocidades de descarga pueden volverse inaceptables, carecer de un plug-in requerido puede hacer que el contenido no esté disponible, las diferencias de navegador pueden cambiar dramáticamente la plantilla de la página, los estilos de fuente pueden alterarse y volverse ilegibles o los formularios pueden organizarse de manera inadecuada. La prueba de compatibilidad busca descubrir dichos problemas antes de que la webapp esté en línea.

El primer paso en la prueba de compatibilidad es definir un conjunto de configuraciones de cómputo, y sus variantes, que "se encuentran comúnmente" en el lado cliente. En esencia, se crea una estructura de árbol, identificación de cada plataforma de cómputo, dispositivos de despliegue usuales, sistemas operativos aceptados en la plataforma, navegadores disponibles, probables velocidades de conexión a internet e información similar. A continuación se deriva una serie de pruebas de validación de compatibilidad, con frecuencia adaptadas de pruebas de interfaz existentes, de navegación, de rendimiento y de seguridad. La intención de estas pruebas es descubrir errores o problemas de ejecución que pueden rastrearse para identificar diferencias de configuración.

A.2.4 Pruebas de navegación

Un usuario viaja a través de una webapp en forma muy parecida a como un visitante camina a través de una tienda o de un museo. Existen muchas rutas que pueden tomarse, muchas paradas que pueden realizarse, muchas cosas que aprender y mirar, actividades por iniciar y decisiones por tomar. Este proceso de navegación es predecible porque cada visitante tiene

un conjunto de objetivos cuando llega. Al mismo tiempo, el proceso de navegación puede ser impredecible porque el visitante, influido por algo que ve o aprende, puede elegir una ruta o iniciar una acción que no es usual conforme el objetivo original. La labor de la prueba de navegación es 1) garantizar que son funcionales todos los mecanismos que permiten al usuario de la webapp recorrerla y 2) validar que cada unidad semántica de navegación (USN) pueda lograr la categoría de usuario apropiada.

Prueba de sintaxis de navegación

La primera fase de la prueba de navegación en realidad comienza durante la prueba de interfaz. Los mecanismos de navegación se prueban para asegurarse de que cada interfaz realiza la función que se le ha encargado. Se sugiere probar cada uno de los siguientes mecanismos de navegación:

- **Vínculos de navegación:** estos mecanismos incluyen vínculos internos dentro de la webapp, vínculos externos hacia otras webapps y anclas dentro de una página web específica. Cada vínculo debe ser probado para asegurarse de que se alcanza el contenido o funcionalidad adecuados cuando se elige el vínculo.
- **Redirecciones:** estos vínculos entran en juego cuando un usuario solicita una URL inexistente o cuando selecciona un vínculo cuyo contenido se removió o cuyo nombre cambió. Se despliega un mensaje para el usuario y la navegación se redirige hacia otra página (por ejemplo, la página de inicio). Los redireccionamientos deben probarse al solicitar vínculos internos incorrectos o URL externas y debe valorarse cómo maneja la webapp estas solicitudes.
- **Marcas de página (favoritos, bookmarks):** aunque las marcas de página son función del navegador, la webapp debe probarse para garantizar la extracción de un título de página significativo conforme se crea la marca.
- **Marcos y framesets:** cada marco incluye el contenido de una página web específica; un frameset contiene múltiples marcos y habilita el despliegue de múltiples páginas web al mismo tiempo. Puesto que es posible anidar marcos y framesets unos dentro de otros, estos mecanismos de navegación y despliegue deben probarse para que tengan el contenido correcto, la plantilla y tamaño adecuados, rendimiento de descargas y compatibilidad de navegador.

- **Mapas de sitio:** un mapa de sitio proporciona una tabla de contenido completa para todas las páginas web. Cada entrada del mapa de sitio debe probarse para garantizar que los vínculos llevan al usuario al contenido o funcionalidad adecuados.
- **Motores de búsqueda internos:** las webapps complejas con frecuencia contienen cientos o incluso miles de objetos de contenido. Un motor de búsqueda interno permite al usuario realizar una búsqueda de palabra clave dentro de la webapp para encontrar el contenido necesario. La prueba del motor de búsqueda valida la precisión y completitud de la búsqueda, las propiedades de manejo de error del motor de búsqueda y las características de búsqueda avanzadas (por ejemplo, el uso de operadores booleanos en el campo de búsqueda).

Algunas de las pruebas anotadas pueden realizarse mediante herramientas automatizadas (por ejemplo, comprobación de vínculos), mientras que otras se diseñan y ejecutan manualmente. La intención de principio a fin es garantizar que los errores en la mecánica de navegación se encuentran antes de que la webapp entre en línea.

Pruebas de la semántica de navegación

Una unidad semántica de navegación (USN) se define como "un conjunto de estructuras de información y navegación relacionada que colaboran en el cumplimiento de un subconjunto de requerimientos de usuario relacionados". Cada USN se define mediante un conjunto de trayectorias de navegación (llamadas "rutas de navegación") que conectan los nodos de navegación (por ejemplo, páginas web, objetos de contenido o funcionalidad). Considerado como un todo, cada USN permite a un usuario lograr requerimientos específicos definidos por uno o más casos de uso para una categoría de usuario. La prueba de navegación ejercita cada USN para asegurarse de que dichos requerimientos pueden lograrse. Es necesario responder las siguientes preguntas conforme se prueba cada USN:

- ¿La USN se logra en su totalidad sin error?
- ¿Todo nodo de navegación (definido por una USN) se alcanza dentro del contexto de las rutas de navegación definidas por la USN?
- Si la USN puede lograrse usando más de una ruta de navegación, ¿se probó cada ruta relevante?
- Si la interfaz de usuario proporciona una guía para auxiliar en la navegación, ¿las instrucciones son correctas y comprensibles conforme avanza la navegación?

- ¿Existe un mecanismo (distinto a la flecha "retroceso" del navegador) para regresar al nodo de navegación anterior y al comienzo de la ruta de navegación?
- ¿Los mecanismos de navegación dentro de un gran nodo de navegación (es decir, una página web grande) funcionan de manera adecuada?
- Si una función debe ejecutarse en un nodo y el usuario elige no proporcionar entrada, ¿el resto de la USN puede completarse?
- Si una función se ejecuta en un nodo y ocurre un error en el procesamiento de la función, ¿la USN puede completarse?
- ¿Existe alguna forma para descontinuar la navegación antes de que todos los nodos se hayan alcanzado, pero luego regresar a donde se descontinuó la navegación y avanzar desde ahí?
- ¿Todo nodo es alcanzable desde el mapa de sitio? ¿Los nombres de nodo son significativos para los usuarios finales?
- Si un nodo dentro de una USN se alcanza desde alguna fuente externa, ¿es posible avanzar hacia el nodo siguiente en la ruta de navegación? ¿Es posible regresar al nodo anterior en la ruta de navegación?
- ¿El usuario entiende su ubicación dentro de la arquitectura de contenido conforme se ejecuta la USN?

La prueba de navegación, como las pruebas de interfaz y usabilidad, debe realizarse por tantos departamentos como sea posible. Usted tiene la responsabilidad durante las primeras etapas de la prueba de navegación, pero las etapas posteriores las deben realizar otros participantes en el proyecto, un equipo de prueba independiente y, a final de cuentas, usuarios no técnicos. La intención es ejercitar la navegación de la webapp a profundidad.

A.2.5 Pruebas de configuración

La variabilidad y la inestabilidad de la configuración son factores importantes que hacen de la prueba de webapps un desafío. El hardware, los sistemas operativos, navegadores, capacidad de almacenamiento, velocidades de comunicación de red y varios otros factores en el lado cliente son difíciles de predecir para cada usuario. Además, la configuración para un usuario dado puede cambiar de manera regular [por ejemplo, actualizaciones del sistema operativo (OS), nuevos ISP y velocidades de conexión]. El resultado puede ser un entorno lado cliente que es proclive a errores sutiles y significativos. La impresión que un usuario tiene de la webapp y la forma en la que interactúa con ella pueden diferir

significativamente de la experiencia de otro usuario si ambos usuarios no trabajan dentro de la misma configuración en el lado cliente. La labor de la prueba de configuración no es ejercitar toda configuración posible en el lado cliente. En vez de ello, es probar un conjunto de probables configuraciones en los lados cliente y servidor para garantizar que la experiencia del usuario será la misma en todos ellos y que aislará los errores que puedan ser específicos de una configuración particular.

Conflictos en el lado servidor

En el lado servidor, los casos de prueba de configuración se diseñan para verificar que la configuración servidor proyectada [es decir, servidor webapp, servidor de base de datos, sistemas operativos, software de firewall (cortafuegos), aplicaciones concurrentes] pueden soportar la webapp sin error. En esencia, la webapp se instaló dentro del entorno del lado servidor y se probó para asegurar que opera sin error.

Conforme se diseñan las pruebas de configuración del lado servidor, debe considerarse cada componente de la configuración del servidor. Entre las preguntas que deben plantearse y responderse durante la prueba de configuración del lado servidor se encuentran:

- ¿La webapp es completamente compatible con el servidor OS?
- ¿Los archivos de sistema, directorios y datos de sistema relacionados se crean correctamente cuando la webapp es operativa?
- ¿Las medidas de seguridad del sistema (por ejemplo, firewalls o encriptado) permiten a la webapp ejecutarse y atender a los usuarios sin interferencia o degradación del rendimiento?
- ¿La webapp se probó con la configuración de servidor distribuido[30] (si existe alguno) que se eligió?
- ¿La webapp se integró adecuadamente con el software de base de datos? ¿La webapp es sensible a diferentes versiones del software de base de datos?
- ¿Los guiones de la webapp en el lado servidor se ejecutan adecuadamente?
- ¿Los errores del administrador del sistema se examinaron en sus efectos sobre las operaciones de la webapp?

[30] Por ejemplo, puede usarse un servidor de aplicación separado de un servidor de base de datos. La comunicación entre las dos máquinas ocurre a través de una conexión de red.

- Si se usan servidores proxy, ¿las diferencias en su configuración se abordaron con pruebas en sitio?

Conflictos en el lado cliente

En el lado cliente, las pruebas de configuración se enfocan con más peso en la compatibilidad de la webapp con las configuraciones que contienen una o más permutas de los siguientes componentes:

- **Hardware:** CPU, memoria, almacenamiento y dispositivos de impresión
- **Sistemas operativos:** Linux, Macintosh OS, Microsoft Windows, un OS móvil
- **Software navegador:** Firefox, Safari, Internet Explorer, Opera, Chrome y otros
- **Componentes de interfaz de usuario:** Active X, Java applets y otros
- **Plugins:** QuickTime, RealPlayer y muchos otros
- **Conectividad:** cable, DSL, módem regular, T1, WiFi

Además de estos componentes, otras variables incluyen software de redes, caprichos de la ISP y aplicaciones que corren de manera concurrente.

Para diseñar pruebas de configuración en el lado cliente, debe reducir el número de variables de configuración a un número manejable[31]. Para lograr esto, cada categoría de usuario se valora para determinar las probables configuraciones que pueden encontrarse dentro de la categoría. Además, pueden usarse datos de participación de mercado para predecir las combinaciones de componentes más probables. Entonces la webapp se prueba dentro de estos entornos.

A.2.6 Pruebas de seguridad

La seguridad de la webapp es un tema complejo que debe comprenderse por completo antes de que pueda lograrse una prueba de seguridad efectiva. Las webapps y los entornos en los lados cliente y servidor donde se albergan representan un blanco atractivo para hackers externos, empleados descontentos, competidores deshonestos y para quien quiera robar información sensible, modificar contenido maliciosamente, degradar el rendimiento, deshabilitar la funcionalidad o avergonzar a una persona, organización o negocio. Las pruebas de seguridad se diseñan para sondear las vulnerabilidades del entorno lado cliente,

[31] Aplicar pruebas en toda combinación posible de componentes de configuración consume demasiado tiempo.

las comunicaciones de red que ocurren conforme los datos pasan de cliente a servidor y viceversa, y el entorno del lado servidor. Cada uno de estos dominios puede atacarse, y es tarea del examinador de seguridad descubrir las debilidades que puedan explotar quienes tengan intención de hacerlo. En el lado cliente, las vulnerabilidades con frecuencia pueden rastrearse en errores preexistentes en navegadores, programas de correo electrónico o software de comunicación. Se describe como un hueco de seguridad común:

Uno de los errores comúnmente mencionados es el desbordamiento de buffer, que permite que código malicioso se ejecute en la máquina cliente. Por ejemplo, ingresar una URL en un navegador que es mucho más larga que el tamaño de buffer asignado para la URL provocará un error de sobrescritura de memoria (desbordamiento de buffer) si el navegador no tiene código de detección de error para validar la longitud de la URL ingresada. Un hacker experimentado puede explotar astutamente este error al escribir una URL larga con código que se va a ejecutar y que puede hacer que el navegador derribe o altere las configuraciones de seguridad (de alto a bajo) o, peor aún, corromper datos del usuario.

Otra vulnerabilidad potencial en el lado cliente es el acceso no autorizado a las cookies colocadas dentro del navegador. Los sitios web creados con intenciones maliciosas pueden adquirir información contenida dentro de cookies legítimas y usar esta información en formas que ponen en riesgo la privacidad del usuario o, peor aún, que montan el escenario para el robo de identidad.

Los datos comunicados entre el cliente y el servidor son vulnerables al spoofing (engaño). El spoofing ocurre cuando un extremo de la ruta de comunicación se trastorna por una entidad con intenciones maliciosas. Por ejemplo, un usuario puede ser engañado por un sitio malicioso que actúa como si fuese el servidor de la webapp legítimo (apariencia y sensación idénticas). La intención es robar contraseñas, información personal o datos de crédito.

En el lado servidor, las vulnerabilidades incluyen ataques de negación de servicio y guiones maliciosos que pueden pasar hacia el lado cliente o usarse para deshabilitar operaciones del servidor. Además, puede accederse sin autorización a las bases de datos en el lado servidor (robo de datos).

Para proteger contra éstas (y muchas otras) vulnerabilidades, se implanta uno o más de los siguientes elementos de seguridad:

- **Firewall:** mecanismo de filtrado, que es una combinación de hardware y software que examina cada paquete de información entrante para asegurarse de que proviene de una fuente legítima y que bloquea cualquier dato sospechoso.
- **Autenticación:** mecanismo de verificación que valida la identidad de todos los clientes y servidores, y permite que la comunicación ocurra solamente cuando ambos lados se verifican.
- **Encriptado:** mecanismo de codificación que protege los datos sensibles al modificarlos de forma que hace imposible leerlos por quienes tienen intenciones maliciosas. El encriptado se fortalece usando certificados digitales que permiten al cliente verificar el destino al que se transmiten los datos.
- **Autorización:** mecanismo de filtrado que permite el acceso al entorno cliente o servidor sólo a aquellos individuos con códigos de autorización apropiados (por ejemplo, ID de usuario y contraseña).

Las pruebas de seguridad deben diseñarse para sondear cada una de estas tecnologías de seguridad con la intención de descubrir huecos en la seguridad.

El diseño real de las pruebas de seguridad requiere conocimiento profundo del trabajo interno de cada elemento de seguridad y amplia comprensión de una gran gama de tecnologías de redes. En muchos casos, la prueba de seguridad se subcontrata con firmas que se especializan en dichas tecnologías.

A.2.7 Pruebas de rendimiento

Nada es más frustrante que una webapp que tarda minutos en cargar contenido cuando sitios de la competencia descargan contenido similar en segundos. Nada es más exasperante que intentar ingresar a una webapp y recibir un mensaje de “servidor ocupado”, con la sugerencia de que se intente de nuevo más tarde. Nada es más desconcertante que una webapp que responde instantáneamente en algunas situaciones y luego en otras parece caer en un estado de espera infinita. Estos eventos suceden en la web todos los días y todos ellos se relacionan con el rendimiento.

Las pruebas de rendimiento se usan para descubrir problemas de rendimiento que pueden ser resultado de: falta de recursos en el lado servidor, red con ancho de banda inadecuada, capacidades de base de datos inadecuadas, capacidades de sistema operativo deficientes o débiles, funcionalidad de webapp pobremente diseñada y otros conflictos de hardware o software que pueden conducir a rendimiento cliente-servidor degradado. La intención es doble: 1) comprender cómo responde el sistema conforme aumenta la carga (es decir, número de usuarios, número de transacciones o volumen de datos global) y 2) recopilar mediciones que conducirán a modificaciones de diseño para mejorar el rendimiento.

Objetivos de la prueba de rendimiento

Las pruebas de rendimiento se diseñan para simular situaciones de carga del mundo real. Conforme aumenta el número de usuarios simultáneos de la webapp o el número de transacciones en línea o la cantidad de datos (descargados o subidos), las pruebas de rendimiento ayudarán a responder las siguientes preguntas:

- ¿El tiempo de respuesta del servidor se degrada a un punto donde es apreciable e inaceptable?
- ¿En qué punto (en términos de usuarios, transacciones o carga de datos) el rendimiento se vuelve inaceptable?
- ¿Qué componentes del sistema son responsables de la degradación del rendimiento?
- ¿Cuál es el tiempo de respuesta promedio para los usuarios bajo diversas condiciones de carga?
- ¿La degradación del rendimiento tiene impacto sobre la seguridad del sistema?
- ¿La confiabilidad o precisión de la webapp resulta afectada conforme crece la carga sobre el sistema?
- ¿Qué sucede cuando se aplican cargas que son mayores que la capacidad máxima del servidor?
- ¿La degradación del rendimiento tiene impacto sobre los ingresos de la compañía?

Para desarrollar respuestas a estas preguntas, se realizan dos tipos diferentes de pruebas de rendimiento: 1) la prueba de carga examina la carga del mundo real en varios niveles de carga y en varias combinaciones, y 2) la prueba de esfuerzo fuerza a aumentar la carga hasta

el punto de rompimiento para determinar cuánta capacidad puede manejar el entorno de la webapp.

Prueba de carga

La intención de la prueba de carga es determinar cómo responderán las webapps y su entorno del lado servidor a varias condiciones de carga. Conforme avanzan las pruebas, las permutas de las siguientes variables definen un conjunto de condiciones de prueba:

- N, número de usuarios concurrentes
- T, número de transacciones en línea por unidad de tiempo
- D, carga de datos procesados por el servidor en cada transacción

En todo caso, dichas variables se definen dentro de fronteras operativas normales del sistema. Conforme se aplica cada condición de prueba, se recopila una o más de las siguientes medidas: respuesta de usuario promedio, tiempo promedio para descargar una unidad estandarizada de datos y tiempo promedio para procesar una transacción. Estas medidas deben examinarse para determinar si una disminución abrupta en el rendimiento puede rastrearse en una combinación específica de N, T y D.

La prueba de carga también puede usarse para valorar las velocidades de conexión recomendadas para los usuarios de la webapp. El rendimiento global, P, se calcula de la forma siguiente:

$$P = N \times T \times D$$

Tome como ejemplo un sitio popular de noticias deportivas. En un momento dado, 20.000 usuarios concurrentes envían una solicitud (una transacción, T) una vez cada 2 minutos en promedio. Cada transacción requiere que la webapp descargue un nuevo artículo que promedia 3 Kb de longitud. Por tanto, el rendimiento global puede calcularse como:

$$P = [20\,000 \times 0.5 \times 3Kb] / 60 = 500 \text{ Kbytes/s}$$

$$P = 4 \text{ megabits por segundo}$$

Por ende, la conexión de la red para el servidor tendría que soportar esta tasa de datos y debería ponerse a prueba para asegurarse de que lo hace.

Prueba de esfuerzo

La prueba de esfuerzo es una continuación de la prueba de carga, pero en esta instancia las variables N, T y D se fuerzan a satisfacerse y luego se superan los límites operativos. La intención de estas pruebas es responder a cada una de las siguientes preguntas:

- ¿El sistema se degrada "suavemente" o el servidor se apaga conforme la capacidad se supera?
- ¿El software servidor genera mensajes "servidor no disponible"? De manera más general, ¿los usuarios están conscientes de que no pueden llegar al servidor?
- ¿El servidor pone en cola los recursos solicitados y vacía la cola una vez que disminuye la demanda de capacidad?
- ¿Las transacciones se pierden conforme la capacidad se excede?
- ¿La integridad de los datos resulta afectada conforme la capacidad se excede?
- ¿Qué valores de N, T y D fuerzan el fallo del entorno servidor? ¿Cómo se manifiesta la falla? ¿Se envían notificaciones automáticas al personal de apoyo técnico en el sitio servidor?
- Si el sistema falla, ¿cuánto tiempo tardará en regresar en línea?
- ¿Ciertas funciones de la webapp (por ejemplo, funcionalidad de cálculo intenso, capacidades de transmisión de datos) quedan descontinuadas conforme la capacidad alcanza el nivel de 80 o 90 por ciento?

A una variación de las pruebas de esfuerzo en ocasiones se le conoce como prueba pico/rebote (spike/bounce). En este régimen de pruebas, la carga alcanza un pico de capacidad, luego se baja rápidamente a condiciones operativas normales y después alcanza de nuevo un pico. Al rebotar la carga del sistema, es posible determinar cuán bien el servidor puede ordenar los recursos para satisfacer una demanda muy alta y entonces liberarlos cuando reaparecen condiciones normales (de modo que esté listo para el siguiente pico).

A.2.8 Resumen

La meta de la prueba de las webapps es ejercitar cada una de las muchas dimensiones de calidad de la webapp con la intención de encontrar errores o descubrir conflictos que puedan conducir a fallas en la calidad. Las pruebas se enfocan en contenido, función, estructura, usabilidad, navegabilidad, rendimiento, compatibilidad, interacción, capacidad y seguridad. Estas pruebas incorporan revisiones que ocurren conforme se diseña la webapp y pruebas que se llevan a cabo una vez que se implanta la misma.

La estrategia de prueba de webapps ejercita cada dimensión de calidad al examinar inicialmente "unidades" de contenido, funcionalidad o navegación. Una vez validadas las unidades individuales, la atención se cambia hacia pruebas que ejercitan la webapp como un todo. Para lograr esto, muchas pruebas se derivan desde la perspectiva del usuario y se activan mediante la información contenida en casos de uso. Se desarrolla un plan de prueba de webapps y se identifican los pasos de la prueba, productos de trabajo (por ejemplo, casos de prueba) y mecanismos para la evaluación de los resultados de la prueba. El proceso de prueba abarca siete tipos diferentes de pruebas.

La prueba de contenido (y las revisiones) se enfocan en varias categorías de contenido. La intención es descubrir errores semánticos y sintácticos que afectan la precisión del contenido o la forma en la que se presenta al usuario final. La prueba de interfaz ejercita los mecanismos de interacción que permiten al usuario comunicarse con la webapp y valida los aspectos estéticos de la interfaz. La intención es descubrir errores que resultan a partir de mecanismos de interacción pobremente implantados o de omisiones, inconsistencias o ambigüedades en la semántica de la interfaz.

Las pruebas de navegación aplican casos de uso, derivados de la actividad de modelado, en el diseño de casos de prueba que ejercitan cada escenario de uso contra el diseño de navegación. Los mecanismos de navegación se ponen a prueba para garantizar que cualquier error que impida completar un caso de uso se identifique y corrija. La prueba de componente ejercita las unidades de contenido y funcionales dentro de la webapp.

La prueba de configuración intenta descubrir errores y/o problemas de compatibilidad que son específicos de un entorno cliente o servidor particular. Entonces se realizan pruebas para descubrir errores asociados con cada posible configuración. Las pruebas de seguridad

incorporan una serie de pruebas diseñadas para explotar las vulnerabilidades en la webapp y su entorno. La intención es encontrar huecos de seguridad. La prueba de rendimiento abarca una serie de pruebas que se diseñan para valorar el tiempo de respuesta y la confiabilidad de la webapp conforme aumenta la demanda en la capacidad de recursos en el lado servidor.