UNIVERSIDAD VERACRUZANA

Facultad de Estadística e Informática
Licenciatura en Informática

Ingeniería de Software II
Manual de prácticas

Elaboraron:

Dra. María de los Ángeles Sumano López
Dr. Juan Manuel Fernández Peña
Dra. María Karen Cortés Verdín

Este material fue utilizado en los periodos: febrero – julio 2010, agosto 2010 – enero 2011, enero – julio 2011 en nueve cursos del plan 2002 de la Licenciatura en Informática

Iniciada en: Febrero de 2010
Última actualización: mayo de 2011

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

Introducción

Dentro del curso de Ingeniería de Software II, por acuerdo de entre los maestros que conforman la Academia de Ingeniería de Software, se incluyeron los siguientes los grupos prácticas:

- Prácticas sobre Ingeniería de Requerimientos de Software para la obtención de u documento de Especificación de Requerimientos de Software.
- Prácticas para la aplicación de principios y métodos de Ingeniería de Usabilidad para la obtención de una Interfaz Gráfica de Usuario acorde al contexto donde se utilizará el software.
- Prácticas de aplicación de la metodología OO escogida para la obtención de los modelos de análisis y diseño de un sistema de software.
- Prácticas sobre métricas de análisis que ayuden tanto a establecer detalles de éste como al cálculo del esfuerzo para el desarrollo de software
- Prácticas sobre la planeación de pruebas de sistema e integración y planeación y aplicación de pruebas de unidad.

Cada práctica de este manual ocupa uno o varios conceptos teóricos y con ellas se adquieren varias competencias.

Las prácticas que se incluyen en este manual son de suma importancia para la práctica profesional de un ingeniero de software.

Prácticas de Ingeniería de Requerimientos

Saber que es lo que se quiere de un nuevo software es primordial para empezar a desarrollarlo. Aunque muchos clientes, usuarios y hasta los mismos Ingenieros de Software ven a la Especificación de Requerimientos de Software (ERS) como una tarea chocante. Sin embargo, sin una ERS correcta, completa y consistente el futuro del software a desarrollar puede preverse como fracasado.

Prácticas de métricas de Análisis

Una vez que el Ingeniero de Software y su cliente han llegado a un acuerdo sobre los servicios que brindará el software y las restricciones de ambiente que se incluirán en la Línea Base, es conveniente: 1) refinar el análisis siguiendo algún método de de cálñculo de métricas y 2) estimar costo y tiempo de desarrollo basado en la complejidad del nuevo software. Así, es necesario saber medirla. Por ello, es necesario que el alumno aplique una o dos técnicas de cálculo de esfuerzo, en este manual se proponen: cálculo de puntos de función[1] y cálculo de puntos de casos de uso.

Prácticas de Ingeniería de Usabilidad

Realizar una GUI (por sus siglas en inglés: Graphic User Interface) pareciera tarea fácil ya que existen muchas herramientas de software que brindan todo tipo de elementos que la conforman, desde ventanas hasta botones y elementos pictóricos. Sin embargo, hacer una GUI usable para el usuario de un software es lo que se busca lograr y para ello hay que conocer: los gustos del usuario, su forma de trabajar, las restricciones de la empresa

[1] Tanto el contenido teórico sobre puntos de función como su práctica se vio en el curso de Ingeniería de Software I, previo a éste. Sin embargo, se vuelve a incluir pues ayuda a revisar los modelos de requerimientos.

y su ambiente físico de trabajo. Los elementos anteriores se deben establecer y para ello se proponen las prácticas de perfilado de usuarios y contexto de uso.

Prácticas de Análisis y Diseño de Software

Antes de empezar a codificar un sistema de software es necesario plantear modelos que permitan entender la brecha entre lo que el usuario plantea en los modelos de requerimientos y lo que realmente se programará. Los modelos necesarios van evolucionando de un alto grado de abstracción, como en el análisis, hasta un grado fino de abstracción, como las clases que conformarán el diseño pasando por estados intermedios de modelos tanto estáticos como dinámicos. Así este grupo de prácticas van paulatinamente acercándose a la codificación final del software.

Prácticas de Prueba de Software

La prueba de software resulta ser una de las tareas más amplias y minuciosas dentro del desarrollo. Sin embargo, sin ella no se podría asegurar, ni remotamente, que el software terminado tiene buena calidad. Así resulta sumamente importante para el Ingeniero de Software tener práctica en esta tarea. En este manual se parten las prácticas por nivel: sistema, integración y unidad.

Estructura del documento

El documento aquí presentado no sigue la secuencia exigida en el documento rector de la Universidad Veracruzana, pero sí contiene todos los elementos que se piden. Así, este documento está constituido en dos partes:

I. Elementos teóricos que sustentan a las prácticas y que por servir a varias de ellas se presentan por separado de la práctica misma.
II. Lista de prácticas en donde se hace alusión a:
    a. Objetivos de aprendizaje.
    b. Descripción de la práctica..
    c. Materiales a emplear en la(s) práctica(s) que son:
        i. Los elementos teóricos, tanto los presentados aquí como a los textos utilizados en el curso
        ii. Las herramientas de software de apoyo

NOTA sobre las Referencias Bibliográficas. Cabe la pena mencionar que muchas referencias no se incluyen por ser anteriores al año 2000, pero que siguen siendo valiosas y que en la redacción de los materiales teóricos aquí presentados sí se emplearon. Los autores no vemos la razón de excluir bibliografía útil, sin embargo no se incluye explícitamente.

Parte I. Bases teóricas

En esta parte se reúnen los elementos teóricos que sustentan las prácticas correspondientes, el material incluido en ocasiones fue tomado de los libros de texto y en otras es un resumen original. En todos los casos los autores han tomado parte y la teoría ha sido revisada en clase.

1 Ingeniería de Requerimientos

La Ingeniería de Requerimientos de Software es la disciplina encargada de “establecer los servicios que un sistema debe suministrar así como las restricciones bajo las cuales debe operar” [Sommerville, I (2005)]. La meta final de la Ingeniería de Requerimientos de Software (IRS) es construir un documento con la Especificación de Requerimientos de Software (ERS). El papel de la ERS es esencial para el logro de un producto de software correcto, cualquiera que sea su área de aplicación. Una lista de actividades de la IRS es: 1) Estudio de factibilidad, 2) Obtención y análisis de requerimientos, 3) Validación de Requerimientos y 4) Especificación de Requerimientos. Las actividades de Obtención y Análisis son las que se revisan y practican en este manual actividades que se realizan utilizando la metodología Áncora [Sumano (2006)].

1.1 Obtención y análisis de requerimientos con Áncora

La metodología Áncora [Sumano-López, M. A. (2006)] -acrónimo de ANálisis de requerimientos de software COnducente al Reuso de Artefactos y sinónimo de ancla -, fue concebida como una metodología que sirviera a los Ingenieros de Software mexicanos para tratar con stakeholders mexicanos y establecer los requerimientos de un nuevo software. Sin embargo, puede ser, y de hecho lo es, de utilidad en otros países.

La idea principal de Áncora es que el Ingeniero de Requerimientos se apropie del conocimiento sobre el dominio de la aplicación y modelo del negocio de forma rápida y expedita y que exprese los requerimientos en diversos formatos para ser entendidos tanto por usuarios como por desarrolladores.

El ciclo de vida de Áncora (ver Figura 1) incluye básicamente cinco fases inmersas en la Ingeniería de Requerimientos, dentro de ellas se realizan una serie de actividades que producen diversos artefactos interrelacionados. La lista de actividades y artefactos se irá desglosando en el apartado correspondiente.

Aunque Áncora surgió sólo como una metodología para el análisis de requerimientos, ha ido creciendo al agregársele actividades de: usabilidad, administración, especificación y prueba.

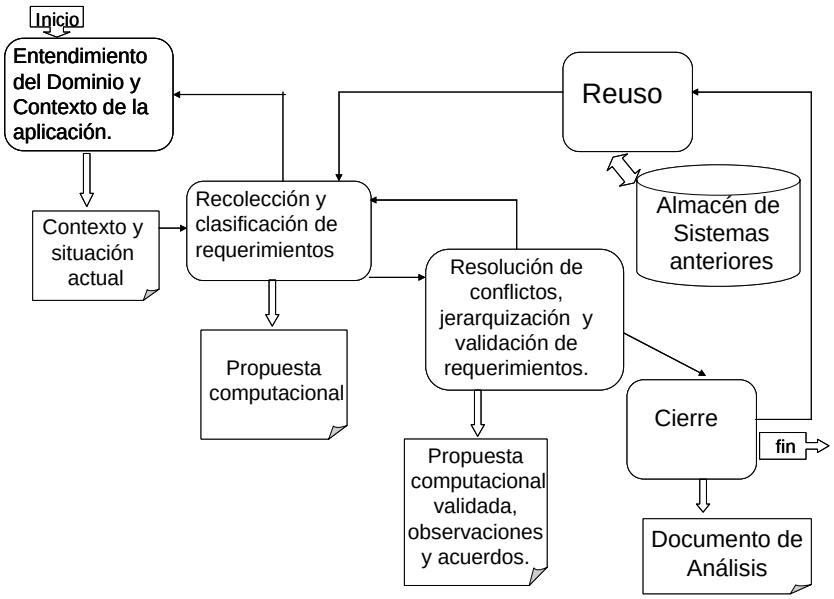


Figura 1. Ciclo de Vida de Áncora

1.2 Procesos Mezclados

El desarrollo de software de buena calidad involucra cientos de actividades que van desde las actividades de protección pasando por las de aseguramiento de la calidad hasta las propias de construcción del software [Pressman, R. S. (2006)], todas ellas entrelazadas.

Un conjunto de estas actividades, que aquí interesan, son las que tienen que ver con el contacto entre humanos y computadoras, a saber: entender qué se quiere del software, cómo se va a usar y cómo se va a mostrar que se hizo lo que se pidió. Es decir, las actividades que marcan Ingeniería de Requerimientos, Ingeniería de Usabilidad y Prueba de Sistemas de Software.

1.3 Conociendo al Usuario y el Contexto de la Aplicación

Conocer el sistema, la situación problemática y su contexto es la meta inicial de Áncora para ello se crean sólo los siguientes artefactos:

i. Red Semántica Natural (RSN). Artefacto que se obtiene aplicando una técnica psicosocial que busca obtener de personas elementos que conforman un concepto en sólo tres minutos. En Áncora se construyen cuatro RSN durante doce minutos en contacto con el usuario, donde se establecen: fronteras, elementos, actividades y sensaciones acerca del sistema en estudio.
ii. Encuesta de Actitud. Se aplica a los usuarios basándose en los resultados de la RSN y es calificada por los usuarios utilizando una escala de Likert de cinco puntos. La encuesta de actitud ayuda a establecer tanto la actitud de los usuarios ante el nuevo software como la identificación de los problemas más graves.
iii. Guión de la situación Actual. Representación gráfica, basada en los Script del área de tratamiento de lenguaje natural, que muestra las actividades que realizan los usuarios del sistema en estudio. En la Figura 2 se muestra un ejemplo de representación utilizando los guiones de Áncora de un sistema de Creación de currículo. Un guión de Áncora puede estar constituido por otros guiones que se identifican con la etiqueta Pista.

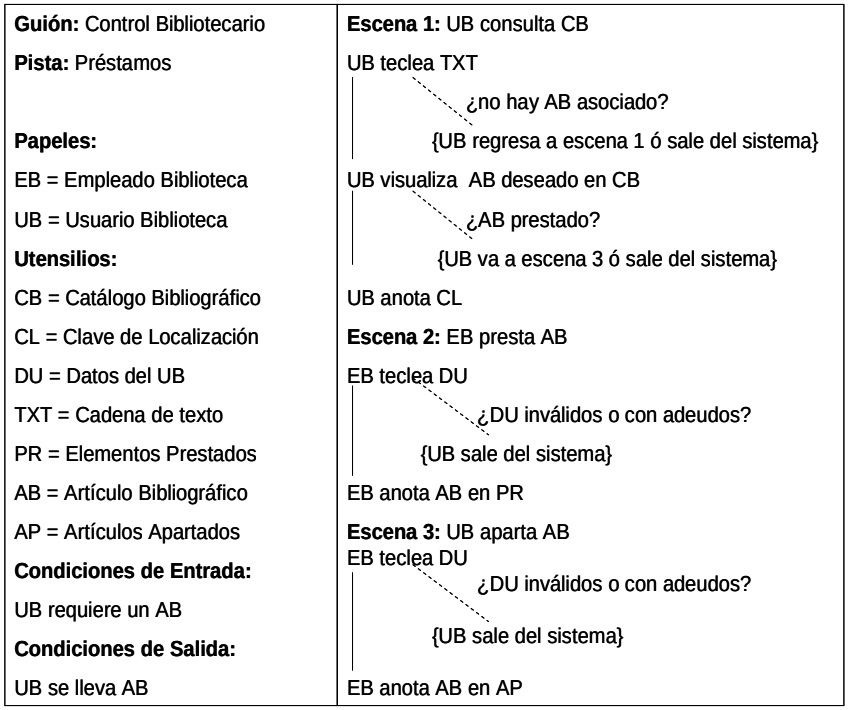


Figura 2. Guión de Áncora que representa un subsistema de préstamos bibliotecarios

Posteriormente, se decidió que no bastaba con saber cómo se trabaja con el sistema en estudio sino que también era necesario saber qué características tenían las personas que realizaban el trabajo, por ello, se decidió perfilar a los usuarios utilizando los artefactos de la siguiente lista [Galindo-Monfil, A. R., (2007)]:

iv. Documento Base. Donde se expresan: motivos para la creación del nuevo software, metas principales del nuevo software y usuarios potenciales.
v. Formulación de Hipótesis de Personajes. Contiene una lista de los diversos usuarios del nuevo software y su clasificación (primaria, secundaria o terciaria).
vi. Contexto de Uso. Se describe el ambiente en donde correrá el nuevo software: factores organizacionales y sociales, medio ambiente físico y elementos tecnológicos.

Los artefactos anteriores se obtienen primero de los tres artefactos originales de Áncora y, posteriormente, se refinan con cuestionarios específicos (de contexto y perfilado de usuario) y la retroalimentación de los usuarios.

1.4 Estableciendo la Propuesta Computacional

¿Qué puede hacer la computadora para mejorar la situación actual y cómo se muestra al usuario la propuesta computacional?, estos son los objetivos de esta etapa, para ello en Áncora se crean los siguientes artefactos [Sumano-López, M. A. (2006)]:

vii. Guión de la Propuesta Computacional. Se sigue la misma notación ejemplificada en la Figura 2 pero se consideran sólo las actividades que se podrían hacer con el uso de la computadora. Del guión se derivan todos los siguientes artefactos, mismos que retroalimentan al guión y viceversa.
viii. Prototipo Rápido. Se conforma como un Manual Preliminar del Sistema desde el punto de vista del usuario. La funcionalidad se muestra por pista (o subsistema) y el orden dentro de cada una va de acuerdo a las escenas que conforman el guión, las actividades dentro de éstas (denominadas quintetas) y los utensilios

empleados. En el prototipo se incluyen las tanto formatos empleados como ventanas propuestas, se detallan los campos y la forma de trabajo, todo basado en las características de los usuarios.

ix. Modelo de datos. De los utensilios y resultados que aparecen en el guión de la propuesta computacional se derivan ya sean los objetos semánticos o entidades y relaciones que conformarán el modelo del dominio.

x. Bitácora de Desarrollo. Este artefacto es una tabla plana de cuatro entradas: función que realizará el sistema (quinteta), forma de comprobación desde el punto de vista del usuario de que la función se realiza, tiempo propuesto para su desarrollo y tiempo real de desarrollo (se llena al entregar el software). Se crea una Bitácora por escena.

xi. Puntos de Función (PF). Más que un artefacto es un número obtenido por el método del mismo nombre (Bradley, M., 1999) que ayuda a estimar la complejidad del sistema y con base en él aproximar tiempos, personal y costos. Además de la estimación, PF brinda una guía para el establecimiento de requerimientos no funcionales, ésto al tratar de establecer los catorce modificadores que incluye el método de PF.

Adicionalmente, si el software es interactivo, se crean los siguientes artefactos de usabilidad (Galindo, A.R. 2007).

xii. Documento Fundacional. Contiene elementos que deben considerarse en la interfaz del usuario.

xiii. Personajes. Por cada tipo de personaje que fue definido en v se relata una historia "inventada" que definirá arquetipos de usuarios, que representan patrones de conducta, objetivos y necesidades.

1.5 Resolviendo Conflictos y Validando los Requerimientos

En este punto ya se tienen elementos para discutir con los stakeholders (involucrados en el desarrollo del software, como: cliente, usuarios, ingenieros) si se entendió el problema y si la propuesta computacional está planteada en términos adecuados, organizados y sin conflicto. Para ello en Áncora se recurre al la Reunión de Reflexión y Diseño (RRD), herramienta nacida de la planeación institucional utilizada en México con éxito (Rodríguez-Contreras, C. ,1998).

La RRD se implementa después de haber entregado a los stakeholders todos los modelos desarrollados en papel. Se debe llevar a cabo en un sitio en el que no interrumpan a los stakeholders y con un facilitador (usualmente un ingeniero de requerimientos) y su rotafolio (o herramienta equivalente). Al final de la RRD se obtienen una serie de correcciones, observaciones y acuerdos sobre la propuesta computacional.

1.6 Cierre

Una vez corregidos los errores, convenidas las observaciones y reflejados los acuerdos tomados en la RRD los modelos de requerimientos pueden utilizarse para:

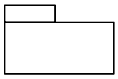
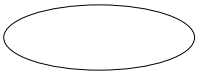
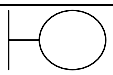
A. Trazar los modelos de requerimientos a modelos de análisis.
B. Planificar las pruebas de sistema a partir de la Bitácora de Desarrollo de Áncora.
C. Crear el documento de Especificación de Requerimientos.

1.6.1 Trazado de modelos

El trazado de modelos se refiere al paso de los artefactos de Áncora a modelos de alguna metodología escogida, puede ser: funcional, de eventos u objetos. Un ejemplo de trazado

sería, para una metodología basada en UML, como se muestra en la Tabla 1 (tomada de (Sumano-López, M. A., Hernández-González, L.A, Fernández-Peña, J. M. 2006)):

Tabla 1. Trazado de artefactos de requerimientos a artefactos de análisis

Artefactos Requerimientos	Cardinalidad del trazado	Artefactos de Análisis	Notación UML
Pista	1:1	Paquete	
Escena	1:1	Caso de uso	
Papel	1:1	Actor	
Relación n:m	1:1	Clases <<entity>>	
Entidad	1:1	Clases <<entity>>	
Quinteta	N:1	Clase <<control>>	
Pantalla de prototipo	M:1	Clase <<boundary>>	

1.6.2 Planificación de Pruebas de Sistema

El establecimiento de requerimientos y la preparación de las pruebas de sistema son actividades simbióticas: si una de ellas se hace mal se perjudica a la otra e, inversamente, al realizar cada una con cuidado hace más sencilla la otra y más útiles sus resultados.

Uno de los defectos más comunes de los requerimientos es que se expresan de forma vaga, haciendo difícil, aún imposible, comprobar si se cumplen en el producto final. Un caso común es el requerimiento “el software debe ser amigable”. ¿Cómo podría comprobarse que el software lo cumple?

Tratar de definir un conjunto de pruebas que aseguren que el software es correcto sin contar con una lista bien definida de requerimientos sería casi imposible. Los desarrolladores propondrían criterios basados en sus propias especificaciones mientras el cliente podría tener una visión completamente diferente a partir de sus necesidades, que probablemente cambiaron desde que encargó el software.

Por otro lado, si se tiene un grupo de requerimientos establecidos, aunque no sean muy buenos y se preparan casos de prueba a partir de ellos, el cliente puede verificarlos para asegurarse que representan lo que espera del sistema, señalando problemas de mala interpretación y resolviendo dudas del probador. Así, después de algunas iteraciones se contaría con requerimientos y casos de prueba adecuados.

La bitácora de desarrollo de Áncora contiene un campo llamado *forma de comprobación*, cuya descripción indica una manera de asegurar que se cumple la quinteta correspondiente, es decir una acción iniciada por una persona.

El campo *forma de comprobación* de la bitácora ha pasado por tres etapas conforme se ha refinado su contenido:

1. Inicial (antes del 2000): originalmente el campo se llenaba de modo totalmente libre. Algunas veces era simplemente una paráfrasis de la propia quinteta o una explicación muy vaga.
2. Intermedia (2000 - 2007) [Fernández-Peña, J.M. y Sumano-López, M.A. (2000)]: Un primer intento de homogeneización incluyó pedir que se incluyeran casos de éxito y de fracaso. Así mejoró un poco, pero algunas personas comenzaban por el caso más común y otros preferían los casos de fracaso más extraños y poco probables. En esta etapa se notó que se incluían requerimientos no funcionales de tipo organizacional, como "se permiten tres intentos".
3. Actual (2007-2009): se organiza un formato homogéneo que incluye casos de éxito, casos indeseables y fallas debidas al medio ambiente.

Antes de llegar a la tercera versión, surgió la tentación de formalizar la descripción de cómo probar, usando predicados o bien OCL, por su liga con UML y el Proceso Unificado [Jacobson, I.,Booch, G., Rumbaugh, J., 2000]. Recordando que ésta etapa, aún cuando busca precisar elementos para uso de los desarrolladores, está enfocada al usuario final y al cliente, se determinó que conviene expresarlo en lenguaje natural, aunque con una cierta estructura.

Al momento se está utilizando con la estructura que se muestra en la Tabla 2; los casos se aplican en el orden que se presentan. En su contenido se consideran los siguientes casos:

1) típico: es el uso más común, en el cual se introducen datos típicos esperando una respuesta aceptable, positiva
2) alterno: es un uso menos frecuente, pero válido, en que se introducen datos no tan comunes y se espera una acción positiva; muchas quintetas carecen de éste tipo de caso
3) indeseable: por algún error o intencionalmente, se introducen datos inaceptables, por lo que se espera una respuesta rechazando la acción, pero sin dañar el sistema
4) fallidos: sin importar la validez de los datos o acciones iniciadas, el software resulta incapaz de responder adecuadamente, es decir, ocurre un problema, generalmente externo, que causa una excepción; idealmente debería contarse con mecanismos de defensa frente a estas situaciones (por ejemplo la base de datos está saturada, la red no funciona, falta un componente)

Tabla 2. Estructura del campo "Forma de Probar"

Tipo de caso	Contenido
Típico (obligatorio)	<acción iniciada por el papel> ; <valores típicos de datos> ; [<acción de respuesta visible al papel> | <acción interna verificable posteriormente>]
Alterno (opcional)	Igual que el anterior
Indeseable (al menos uno)	Igual que los anteriores
Fallido (opcionales)	<acción iniciada por papel> ; <situación causante> ; <cómo se manifiesta>

Una forma de asegurar que una regla se cumpla es incluirla en las herramientas de desarrollo, de modo que se reduzcan las omisiones por descuido o desconocimiento. Así pues, se tiene en desarrollo una herramienta que ayuda, de manera interactiva, a

completar éste campo, asegurando la homogeneidad del contenido. Tal herramienta ayudará, además, a preparar los casos de prueba a nivel de sistema.

Generación de casos de prueba. A partir del guión y la bitácora de desarrollo se tienen elementos para avanzar en la preparación de pruebas, aunque no los suficientes para automatizar totalmente la preparación de casos de prueba. Actualmente se realiza como sigue:

1) La estructura del guión consta de una serie de escenas, cada una de las cuales es relativamente independiente de las otras y corresponde a una cierta funcionalidad. A cada escena se asocia un procedimiento de prueba que incluye las condiciones necesarias para poder aplicar los casos de prueba asociados a ella
2) Los casos de prueba se generan para cada quinteta incluida en la escena, usando la información del campo *forma de comprobación*

Nótese que una quinteta puede formar parte de varias escenas diferentes, por lo cual los casos de prueba se reutilizarán. Se deben aplicar en cada caso, pues su contexto es diferente.

Los casos de prueba se generan de modo que correspondan a los casos incluidos en el campo *forma de comprobación*, empleando los métodos de particiones en clases de equivalencia y de valores a la frontera (Jorgensen, P.C., 1995) para casos: típicos, alternos e indeseables. Al menos se incluirá un caso de prueba por cada caso descrito textualmente. Para los casos fallidos importa más una condición de entrada que valores específicos de entrada, pero su estructura es semejante.

2 Métricas de Análisis

Las métricas juegan un papel importante en la Ingeniería de Software. Dependiendo de la etapa del proceso de desarrollo y de los artefactos involucrados pueden obtenerse diferentes métricas. Las métricas se emplean para comprender mejor los atributos de los modelos que se crean y evaluar la calidad de los artefactos de los sistemas que se construyen. De esta manera, una métrica permite tomar decisiones con respecto a los artefactos y al proceso de desarrollo. Una métrica se define como “medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo determinado” de acuerdo al Glosario de Términos de Ingeniería de Software de la IEEE (1993).

Dado que el interés de la experiencia educativa se centra, en un principio en los requerimientos, específicamente en el análisis de los mismos, en esta sección se presenta la métrica de Puntos de Casos de Uso (PCU o UCP por sus siglas en inglés). Existe también la métrica de Puntos de Función (PF) pero ésta fue tema de Ingeniería de Software I por lo que no se cubre aquí. La métrica de UCP sirve para estimar el esfuerzo requerido para el desarrollo de un sistema basado en el modelo de casos de uso del sistema. Los PCU toman en cuenta la complejidad del sistema, aspectos ambientales y características técnicas como rendimiento, concurrencia y seguridad. Ya que los UCP determinan el esfuerzo de desarrollo de un sistema pueden emplearse para determinar el tiempo requerido para su desarrollo así como su costo.

La fórmula para los casos de uso es:

UCP=UUCP*TCF*ECF*PF

Donde:

1. UUCP = Puntos de Caso de Uso sin ajustar
2. TCF = Factor de complejidad técnica
3. ECF = Factor de Complejidad del Medio Ambiente
4. PF = Factor de productividad

En las siguientes secciones se explican los pasos para obtener los UCP.

2.1 Cálculo de los Puntos de Casos de Uso sin ajustar (UUCP)

Los UUCP dan una idea más precisa de la dificultad de los casos de uso y de las interfaces. La fórmula para calcularlos es:

UUCP = UAW + UUCW

Donde:

1. UAW son los pesos de los Actores sin ajustar, que se calculan basado en la la combinación de la complejidad de todos los actores en todos los casos de uso.
2. UUCW son los pesos de los Casos de Uso sin ajustar y se calculan de acuerdo al número total de actividades (o pasos) contenidos en todos los escenarios del caso de uso.
3.

Para calcular los pesos de los Actores sin ajustar (UAW), se emplea la Tabla 3.

Tabla 3. Asignación de pesos para Actores

Tipo de Actor	Descripción	Peso (factor)
Simple	Otro sistema que interactúa con el sistema a desarrollar mediante una interfaz de programación (API).	1
Medio	Otro sistema interactuando mediante un protocolo (ej. TCP/IP) o una persona interactuando a través de una interfaz en modo texto.	2
Complejo	Una persona que interactúa con el sistema mediante una interfaz gráfica (GUI).	3

Para calcular los pesos de los Casos de Uso sin ajustar (UUCW), se emplea la Tabla 4.

Tabla 4. Asignación de Pesos para Casos de Uso (ver Nota al final de esta sección)

Categoría de Caso de Uso	Descripción	Peso (factor)
Simple	Transacciones= 3 ó menos Clases= Menos de 5	5
Medio	Transacciones= 4 a 7 Clases= 5 a 10	10
Complejo	Transacciones= Más de 7 transacciones Clases= Más de 10 clases	15

2.2 Factor de Complejidad Técnica (TCF)

La fórmula para el TCF es:

TCF=0.6 + (0.1*Factor Total Técnico)

Para el cálculo del Factor Total Técnico se consideran 13 puntos que evalúan la complejidad de los módulos que integran el sistema en desarrollo. Los factores ya tienen un peso definido como puede verse en la Tabla 5. Cada factor se evalúa de acuerdo a su importancia en el proyecto: Irrelevante se califica entre 0 a 2, Medio se le asigna de 3 a 4 y Esencial obtiene el valor de 5. La asignación o evaluación de la importancia en el proyecto la realiza el equipo de desarrollo de manera subjetiva para cada uno de los 13 puntos. El peso del factor se multiplica por la importancia percibida. En la misma Tabla 6 se muestra un ejemplo de esta asignación y la suma de las multiplicaciones.

Tabla 5. Asignación de Pesos para Factores Técnicos

Factor técnico	Descripción	Peso
T1	Sistema distribuido	2
T2	Rendimiento o tiempo de respuesta	1
T3	Eficiencia del usuario final	1
T4	Procesamiento interno complejo	1
T5	El código debe ser reutilizable	1
T6	Facilidad de instalación	0.5
T7	Facilidad de uso	0.5
T8	Portabilidad	2
T9	Facilidad de cambio	1
T10	Concurrencia	1
T11	Características especiales de seguridad	1
T12	Provee acceso directo a terceras partes	1
T13	Se requiere facilidades especiales de entrenamiento a usuarios	1

Tabla 6. Asignación de Pesos para Factores Técnicos

FT	Descripción	Peso	Impacto	Factor
T1	Sistema distribuido	2	1	2
T2	Rendimiento o tiempo de respuesta	1	3	3
T3	Eficiencia del usuario final	1	3	3
T4	Procesamiento interno complejo	1	3	3
T5	El código debe ser reutilizable	1	1	1
T6	Facilidad de instalación	0.5	1	0.5
T7	Facilidad de uso	0.5	3	1.5
T8	Portabilidad	2	1	2
T9	Facilidad de cambio	1	3	3
T10	Concurrencia	1	0	0
T11	Características especiales de seguridad	1	0	0
T12	Provee acceso directo a terceras partes	1	3	3
T13	Facilidades entrenamiento a usuarios	1	1	1
			Factor Total Técnico=	23

El resultado de esta evaluación es el Factor Total Técnico con el cual puede calcularse el TCF.

2.3 Factor de Complejidad Ambiental (ECF)

Permite determinar la experiencia del equipo de desarrollo ya que evalúa las habilidades y experiencia del equipo de desarrollo. La fórmula es:

$$ECF = 1.4 + (-0.03 * \text{Factor Ambiental Total})$$

Cada factor ambiental cuenta ya con un peso asignado, de acuerdo con la Tabla 7. Cada factor ambiental se debe calificar con un valor de 0 a 5. Un valor de 1 significa que el factor tiene un fuerte impacto negativo para el proyecto, 3 es medio y 5 significa que tiene un fuerte impacto positivo. El peso del factor se multiplica por la importancia o impacto percibido. En la Tabla 8 se muestra un ejemplo de esta asignación y la suma de las multiplicaciones.

Tabla 7. Asignación de pesos para Factores Ambientales

Factor Ambiental	Descripción	Peso
E1	Familiaridad con el modelo de proyecto utilizado (Ej. UML)	1.5
E2	Personal tiempo parcial	-1
E3	Capacidad del analista líder	0.5
E4	Experiencia en la aplicación	0.5
E5	Experiencia en orientación a objetos	1
E6	Motivación	1
E7	Dificultad del lenguaje de programación	-1
E8	Estabilidad de los requerimientos	2

Tabla 7. Asignación de pesos para Factores Ambientales

Factor Ambiental	Descripción	Peso	Impacto	Factor
E1	Familiaridad con el modelo de proyecto utilizado Familiaridad con UML	1.5	3	4.5
E2	Personal tiempo parcial	-1	0	0
E3	Capacidad del analista líder	0.5	5	2.5
E4	Experiencia en la aplicación	0.5	0	0
E5	Experiencia en orientación a objetos	1	5	5
E6	Motivación	1	3	3
E7	Dificultad del lenguaje de programación	-1	0	0
E8	Estabilidad de los requerimientos	2	3	6
	Factor Ambiental Total=			21

El resultado de esta evaluación es el Factor Ambiental Total con el cual puede calcularse el ECF.

2.4 Productividad

Una vez calculados los UCP, el Factor de Productividad (PF) se emplea para estimar el número de horas. El PF es la relación de horas-hombre necesitadas por cada punto de caso de uso. Para ello se emplean datos históricos de productividad. Cuando éstos no se tienen, se pueden seguir las siguientes recomendaciones:

1. Establecer una base para cálculos de los UPC de proyectos completados anteriormente.
2. Utilizar un valor entre 15 y 30 dependiendo de la experiencia y logros pasados del equipo de desarrollo. Si se trata de un nuevo equipo, emplear un valor de 20 para el primer proyecto.

NOTA Aclaratoria sobre Asignación de Pesos para Casos de Uso

La Tabla de Asignación de Pesos para Casos de Uso, según el autor del artículo original, queda un poco incompleta por lo que se propone utilizar la Tabla 5.

Tabla 5. Asignación de Pesos para Casos de Uso (AMPLIADA)

Transacciones Clases	1 a 3	4 a 7	Más de 7
1 a 4	Simple	Simple	Mediano
5 a 10	Simple	Mediano	Complejo
Más de 10	Mediano	Complejo	Complejo

3 Ingeniería de Usabilidad

Un sistema de buena calidad no sólo debe procurar la funcionalidad correcta, sino que la tiene que proporcionar de manera que sus usuarios puedan utilizarla de forma efectiva. De ahí que la usabilidad es parte fundamental para el éxito de un software.

Por otro lado, Áncora es una metodología de Ingeniería de Requerimientos con orientación al usuario que incluye aspectos lingüísticos, psico-sociales y de planeación, por lo que se decidió aprovechar sus ventajas y orientación, para la introducción de actividades y la creación de artefactos para el establecimiento de características de usabilidad de un Sistema de Software, que puedan ser generados en forma paralela con el establecimiento de requerimientos del software.

3.1 Ingeniería de Usabilidad y Métodos a Incorporar en Áncora

Usabilidad es una construcción multidimensional que puede ser examinada desde varias perspectivas. Las definiciones de usabilidad proporcionadas por Jackob Nielsen y la Organización Internacional para la Estandarización (ISO), son las más extensamente citadas y utilizadas, por lo que se presentan a continuación.

Nielsen, precisa que la usabilidad está asociada tradicionalmente con cinco atributos, a saber: 1) Fácil de aprender.2) Eficiente. 3) Fácil de Recordar. 4) Baja Incidencia de Errores o Fácil Recuperación de Errores y 5) Satisfacción subjetiva.

Por su lado, ISO dispone de dos definiciones de usabilidad:

i. "La usabilidad se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso".
ii. “La medida en que un producto puede ser utilizado por usuarios específicos, para alcanzar metas específicas, de forma efectiva, eficiente y satisfactoria, dentro de un contexto de uso”.

Basándose en las definiciones anteriores se nota que para poder conseguir usabilidad en el diseño de interfaces de usuario durante el desarrollo de un producto interactivo, la Ingeniería de Usabilidad debe proporcionar la manera de proceder organizadamente. Ésta se trata de una disciplina que tiene sus raíces en otras disciplinas básicas como: Psicología Cognitiva, Psicología Experimental, Etnografía, Ergonomía e Ingeniería de Software.

Es importante tener en cuenta que un diseño óptimo, desde el punto de vista de usabilidad, no puede conseguirse basándose solamente en principios generalistas: *cada producto y sus usuarios son únicos. Por lo tanto, aplicar métodos sin seguir unas líneas de trabajo perfectamente definidas y bien organizadas suele llevar al fracaso*.

3.1.1 Algunos Métodos de Usabilidad

Después de hacer una amplia revisión de métodos para conseguir la usabilidad, en este apartado se muestra un breve resumen de los tres más prometedores para incorporar en el análisis de requerimientos de software mediante Áncora y son: Investigación Contextual, Personajes, Ordenamiento de Tarjetas.

- **Investigación Contextual.** Esta técnica fue desarrollada por Beyer y Holtzblatt en el marco de una metodología más completa denominada Diseño Contextual, tiene sus raíces en la Antropología y consiste en desarrollar entrevistas en profundidad y

observación de usuarios en sus lugares habituales de trabajo, como una forma de incorporar tanto el análisis del ambiente en el que se desenvuelven los futuros usuarios como la forma en la que interactúan con éste, además de provocarles mayor confianza al estar en su entorno.

- **Personajes.** Esta técnica, que el autor nombra *persona* aquí se tradujo como *personajes* para continuar con el símil de Áncora de una obra de teatro, fue desarrollada por Alan Cooper se basa en la definición de arquetipos de usuarios, que representan patrones de conducta, objetivos y necesidades. Deriva de una técnica de mercadeo en la cual se crea un personaje ficticio que refleje el estilo de vida y modo de pensar de una parte importante de la audiencia. La razón detrás de estos personajes arquetípicos es que el otorgarle una identidad al usuario, para quien se está desarrollando el software, ayuda a orientar las discusiones manteniendo el foco sobre qué aspectos serán más relevantes. Se pueden utilizar como guía para ayudar a tomar decisiones sobre características del producto de software, navegación, interacciones y diseño visual.

- **Ordenamiento de Tarjetas.** Técnica nombrada en inglés Card Sorting, ha sido utilizada con éxito durante años en otras áreas, como la psicología clínica y la adquisición de conocimiento en Sistemas Expertos. En el ámbito de la usabilidad ha sido utilizada por varios autores entre los que se encuentran Nielsen y Constantine. Ésta técnica se basa en la observación de cómo los usuarios agrupan y asocian entre sí un número predeterminado de tarjetas etiquetadas con diferentes categorías temáticas del Sistema. De esta forma, partiendo del comportamiento de los propios usuarios, es posible organizar y clasificar la información conforme a su modelo mental.

3.2 Métodos y Artefactos de Ingeniería de Usabilidad Incorporadas en Áncora

Las técnicas de Interacción Humano-Computadora (HCI) tienen como objetivo incrementar el nivel de usabilidad de un producto de software. En esta sección se muestra, de forma resumida, como fueron incorporadas técnicas HCI en el ciclo de vida de Áncora y los artefactos derivados de éstas, así como una mejora realizada al prototipo propuesto por Áncora.

3.2.1 Técnicas de IU aplicadas en Ancora

Se proporciona el nombre de la técnica, sus propósitos, la etapa o etapas en el Ciclo de Vida de Áncora que son afectadas en sus actividades, las condiciones necesarias para la aplicación de la técnica, los instrumentos que complementan al método, los participantes en su aplicación, las actividades extra que son necesarias llevar a cabo, marcadas con asterisco (*****), y por último los productos resultantes.

Análisis Contextual para la Usabilidad con Áncora. El objetivo es observar el comportamiento cotidiano de los usuarios, los artefactos con los que complementan su interacción con la computadora y otros aspectos de su contexto diario.

Etapa del Ciclo de Vida de Áncora: Entendimiento del Dominio y Contexto de la Aplicación

Condiciones para Aplicar*:* Debe iniciarse después que se haya aplicado la encuesta de actitud.

Instrumentos Complementarios: Como herramienta de apoyo se utiliza un cuestionario para perfilar a los usuarios en cuanto a edad, sexo, escolaridad,

experiencia en el uso de computadoras, motivación, experiencia en cuanto a software y aspectos de usabilidad.
Participantes: Clientes, Usuarios y Desarrolladores.
Actividades: Entrevistas informales con el cliente, lectura de material, aplicar de cuestionario*, entrevistas informales con los usuarios y observar usuarios*.
Productos: Artefacto1 (Documento Base), Artefacto 2 (Formulación de Hipótesis de Personajes), Artefacto 3 (Contexto de Uso) y Artefacto 4 (Cuestionario Perfilado de Usuario).

Definición de Personajes para la Usabilidad con Áncora. Tiene como fin identificar los tipos de usuario y darles forma mediante características cualitativas.

Etapas en el Ciclo de Vida de Áncora: Entendimiento del Dominio y Contexto de la Aplicación, Recolección y Clasificación de Requerimientos, Resolución de conflictos, jerarquización y validación de requerimientos
Condiciones para su Aplicación: Se diseñan después de realizar las actividades de la Técnica de Investigación Contextual.
Instrumentos Complementarios: Se utilizan los artefactos obtenidos mediante la técnica de Investigación Contextual para diseñar a los personajes, pero con la información obtenida en las otras etapas del ciclo de vida de Áncora, se irán refinando.
Participantes: Desarrolladores.
Actividades: Revisar la hipótesis de personajes*, sintetizar características y metas relevantes*, verificar la completitud del elenco*, desarrollar narrativas*, definir tipos de personajes*.
Productos: Artefacto 5 (Documento Fundacional), Artefacto 6 (Personajes).

Ordenamiento de Tarjetas para la Usabilidad con Áncora. Su propósito es organizar y clasificar la información conforme al modelo mental de los usuarios.

Etapas en el Ciclo de Vida de Áncora: Recolección y Clasificación de Requerimientos, Resolución de conflictos, jerarquización y validación de requerimientos
Condiciones para su Aplicación: Tener la primera aproximación del Guión de la Propuesta Computacional.
Instrumentos Complementarios: Se utilizan las Actividades del usuario, obtenidas mediante la técnica Redes Semánticas Naturales y expresadas en escenas y quintetas.
Participantes: Usuarios y Desarrolladores.
Actividades: Crear tarjetas*, prueba piloto*, elegir a los participantes*, aplicar técnica*, analizar resultados*.
Productos: Artefacto 7 (Guión de la Propuesta Computacional Corregido).

Prototipado para la Usabilidad con Áncora. Su objetivo es mejorar el prototipo establecido por Áncora, primordialmente en relación a las pantallas presentadas en el Manual de Operación del Usuario.

Etapas en el Ciclo de Vida de Áncora: Resolución de conflictos, jerarquización y validación de requerimientos y Cierre.
Condiciones para su Aplicación: Haber realizado todas las actividades de las técnicas HCI propuestas al menos una vez.
Instrumentos Complementarios: Se utilizan todos los Artefactos generados.
Participantes: Desarrolladores, Diseñadores Gráficos y Usuarios.

Actividades: Crear un diseño base*, Crear un prototipo en papel*, Evaluar el prototipo*, Refinar el prototipo*.
Productos: Artefacto 8 (Diseño Base).

3.2.2 Artefactos Resultantes de la aplicación de Técnicas de IU

Por cuestión de espacio se presentan sólo algunos de los formatos correspondientes a los artefactos resultantes de la aplicación de las técnicas, proporcionándose una breve descripción de lo que contienen.

Artefacto 1. (Figura 1). Documento Base. Contiene los motivos del cliente para la creación del nuevo software, sus metas principales y los usuarios potenciales.

Artefacto 2. Formulación de Hipótesis de Personajes (Figura 2). Contiene enunciados que describen a los posibles usuarios del producto de software, lo cuál permitirá dirigir la investigación en campo (observación y entrevista de usuarios).

Artefacto 3. Contexto de Uso (Figura 3). Describe parte del contexto en el que será usado el producto de software, para que el equipo de desarrollo pueda adaptar mejor el software, contiene datos cualitativos y cuantitativos sobre factores: organizacionales, sociales, medio ambiente físico, horario de uso y aspectos técnicos sobre hardware y software.

RESULTADOS DE LA ENTREVISTA INFORMAL REALIZADA AL CLIENTE:
Proyecto: SISTEMA DE FARMACIA
Nombre del entrevistador: Luis E. Méndez M.
Fecha de entrevista:18/10/2006

1. Motivos para la creación del nuevo software:
Descontrol en los inventarios de la farmacia del hospital

2. Metas Principales del nuevo software:
- Llevar el control de las existencias (entradas y salidas) de Farmacia.
- Registrar las ventas realizadas al público en general y las ventas a los pacientes que se encuentran internados.
- Que el sistema interactúe con los sistemas de Hospitalización y de Contabilidad.

3. Usuarios potenciales:
El personal de la farmacia.

Figura 3. Artefacto 1 (Documento Base)

FORMULACIÓN DE HIPÓTESIS DE PERSONAJES
Proyecto: SISTEMA DE FARMACIA
Elaboró: Ana Lilia Izquierdo C.
Fecha de elaboración: 20/10/2006

Utilizarán el sistema:
Personas que laboran en el área de farmacia, con nivel educativo de preparatoria, en su mayoría mujeres entre 25 y 50 años de edad, con poca experiencia en el uso de computadoras y orientadas a la atención al público.

Figura 4. Artefacto 2 (Formulación de Hipótesis de Personajes)

Artefacto 4. Cuestionario Perfilado de Usuario (Figura 4). Tiene datos que describen a los usuarios en cuanto a: puesto, tiempo en el puesto, experiencia en puestos similares, tiempo en la empresa, edad, sexo, escolaridad, horas diarias dedicadas a trabajos con la computadora, actividades para las que utiliza la computadora y experiencia en el uso de software. Este artefacto ayuda a obtener el perfil de los usuarios y a corregir, en caso necesario, el artefacto 2.

Artefacto 5. Documento Fundacional (Figura 5). Se utiliza para archivar las características principales de cada personaje y sirve de respaldo a la creación del Artefacto 6.

Artefacto 6. Personajes (Figura 6). Este artefacto contiene la narrativa sobre los personajes ficticios, la cual se obtiene de sintetizar la información recabada en los artefactos: 2, 3, 4, 5 y llevar a cabo las actividades: Desarrollar narrativas y Definir tipos de Personajes (primarios y secundarios). Ésta narrativa debe contener de manera explícita las metas, necesidades, patrones de trabajo, comportamiento, ambiente y actitudes del personaje, basadas en la información real obtenida mediante la Investigación Contextual, así como algunos detalles personales supuestos que deben estar registrados en el artefacto 5 y que también pueden estar sustentados en dicha información. Su objetivo es tener una idea clara de para quién se diseñará el producto interactivo y servir como medio de comunicación entre los integrantes del equipo de diseño, contribuyendo a estimular la capacidad inventiva y de análisis, así como también realizar inferencias acerca del futuro uso y utilidad del Sistema de Software.

Artefacto 7. Guión de la Propuesta Computacional Corregido. Éste artefacto consiste en reordenar la Propuesta Computacional acorde a los resultados obtenidos en el ordenamiento de tarjetas y por lo tanto de acuerdo al modelo mental de los usuarios.

Uno de los beneficios percibidos de la aplicación de los artefactos de usabilidad, es que ayuda a los estudiantes de la Licenciatura en Informática a comprender mejor las técnicas HCI y su aplicación, al estar completamente integradas en el ciclo de vida de los requerimientos. Además los usuarios han expresado que la técnica Ordenamiento de Tarjetas, resulta útil pues permite reconsiderar los requerimientos, su agrupación y el orden de aplicación.

Documento Fundacional
Proyecto:
Elaboró:
Fecha:
Personaje:
Un día en la vida
Narrar cómo es un día típico en la vida del personaje.
Trabajo
Describir el puesto de trabajo y las actividades que realiza.
Vida privada
Relatar las actividades del personaje fuera del trabajo.
Metas
Especificar las metas finales del personaje.
Miedos y aspiraciones
En la vida profesional y personal del personaje.
Experiencia con computadoras
Explicar el uso que da el personaje a las computadoras.
Atributos demográficos
Proporcionar información demográfica acerca del personaje.
Uso de tecnología
Describir lo que hace el personaje con los artefactos tecnológicos.
Actitud
Señalar su actitud con respecto a la tecnología.
Comunicación
Señalar cómo y a través de qué medios el personaje se comunica con los demás.

Figura 5. Artefacto 5 (Documento Fundacional)

CONTEXTO DE USO DEL PROYECTO:

Fecha. _____

Elaboró: _____

Factores Organizacionales

	Inexistente	Admisible	Bueno
Estructura Organizacional			
Procesos de Trabajo			

	Alta	Moderada	Inexistente
Presión Organizacional			

Factores Sociales

	Pésimo	Admisible	Bueno
Ambiente laboral			

	Inexistente	Moderado	Alto
Conflictos entre empleados			

Medio ambiente (Áreas donde se usará el software)

Nombre del Área	
Horario de uso	
Espacio Físico	

	Incipiente	Moderada	Alta
Iluminación			

	Incipiente	Moderado	Alto
Ruido			

	Invierno	Primavera	Verano	Otoño
	mala, regular, buena	mala, regular, buena	mala, regular, buena	mala, regular, buena
Temperatura				

Factores Técnicos:

1. Software que utilizan

Sistemas Operativos	
Software de aplicación	

2. Computadoras

Descripción	RAM	DD	Cantidad

3. Red

	SI	NO	Velocidad
Conectividad en red			

Figura 6. Artefacto 3 (Contexto de Uso)

Edad _____ Sexo: F ☐ M ☐

1. ¿Cuánto tiempo tiene trabajando para esta empresa?
2. ¿Qué puesto desempeña?
3. ¿Cuánto tiempo lleva desempeñando ese puesto?
4. ¿Tiene experiencia en puestos similares? ¿Cuánto tiempo?

5. Escolaridad:

Sin Estudios ☐ Secundaria ☐ Preparatoria ☐ Técnico ☐ Universitario ☐ Maestría ☐
Doctorado ☐ Diplomado ☐ Otro ☐ Especifique: _____

6. ¿Cuántas horas diarias dedica a trabajos con la computadora?

Menos de 1 ☐ Entre 1 y 2 ☐ Entre 2 y 4 ☐
Entre 4 y 8 ☐ Más de 8 ☐ No la utilizo ☐

7. Utiliza la computadora para:

Trabajar ☐ Estudiar ☐ Entretenimiento ☐ No la utilizo ☐
Otro motivo ☐ Especifique: _____

8. En los últimos 6 meses ¿Qué tipo de software ha utilizado? (Seleccione las casillas que estén de acuerdo a su experiencia.

Mi grado de experiencia como usuario es:	Nula	Incipiente	Medio	Mucha	Experto
Sistema operativo Unix (Linux)					
Sistema operativo Windows					
Sistema operativo Macintosh					
Lenguajes de programación					
Procesadores de texto					
Hojas de cálculo					
Sistemas de bases de datos					
Sistemas cooperativos					
Sistemas de información geográfica					
Edición gráfica					
CAD					
Aplicaciones Multimedia					
Navegadores y Buscadores para Internet					
Correo electrónico					
Chats					
Juegos					
Películas en DVD					
Música					
Otras aplicaciones					

1. En su opinión, ¿qué aspectos considera más importantes en un software? Valórelos:

Un software debe ser:

Fácil de usar	☐ ☐ ☐ ☐ ☐	Difícil de usar
Productivo	☐ ☐ ☐ ☐ ☐	Improductivo
Rápida respuesta	☐ ☐ ☐ ☐ ☐	Lenta
Fácil de Recordar	☐ ☐ ☐ ☐ ☐	Difícil de Recordar
Atractivo	☐ ☐ ☐ ☐ ☐	Nada atractivo
Fácil de Entender	☐ ☐ ☐ ☐ ☐	Incomprensible
Bien organizado	☐ ☐ ☐ ☐ ☐	Caótico
Entretenido	☐ ☐ ☐ ☐ ☐	Aburrido
Flexible	☐ ☐ ☐ ☐ ☐	Rígido
Confiable	☐ ☐ ☐ ☐ ☐	Inseguro
Fácil de Aprender	☐ ☐ ☐ ☐ ☐	Difícil de Aprender

2. ¿Cuál es su color favorito?
3. ¿Cuáles son los colores de la empresa (fondo, letras, logo)?

¡Muchas Gracias por su colaboración!

Figura 7. Artefacto 4 (Cuestionario Perfilado de Usuario)

Ana Iñarritu
(Personaje Primario)

Ana tiene 30 años de edad, es originaria de Xalapa, Veracruz donde vive actualmente. Se desempeña como empleada de un hospital en el área de farmacia, en la cual se brinda servicio tanto a pacientes internos del hospital como al público en general. Ella vive en una casa pequeña, de forma modesta que está ubicada cerca del hospital, tiene esposo y dos niños. Ella estudió la preparatoria y ha trabajado en el hospital desde hace 10 años, por lo que conoce perfectamente sus funciones. Ella se considera una persona activa, eficiente y tolerante.

Ella ha recibo algunos cursos de capacitación sobre el uso de computadoras por parte del hospital, regularmente la utiliza para llenar formatos en Excel y elaborar documentos en Word, no cuenta con una computadora en casa.

Ana se encarga actualmente de registrar la información de los movimientos de inventarios en la farmacia en formatos de Excel, pero constantemente existen inconsistencias con los inventarios físicos y eso le ocasiona retraso, extender su jornada de trabajo, presiones y estrés, lo que afecta de forma significativa su vida personal. Además a finales de año el trabajo es mucho mayor debido a los movimientos que debe realizar para cerrar un inventario y empezar el siguiente.

Las metas de Ana son:

Realizar su trabajo de manera fácil
Cometer pocos errores y poder corregirlos
Terminar su trabajo en el horario establecido

Figura 8. Artefacto 6 (Personajes)

4 Análisis y Diseño Orientado a Objetos

La metodología que se emplea en este curso, para el análisis y diseño, es ICONIX, en esta sección se presenta un resumen de la metodología, que sirve de recordatorio, pero no sustituye la lectura del libro donde el autor de la metodología la explica [Rosenberg, D. (1999)].

ICONIX, utiliza UML como notación y genera artefactos tanto dinámicos como estáticos (ver Figura 9).

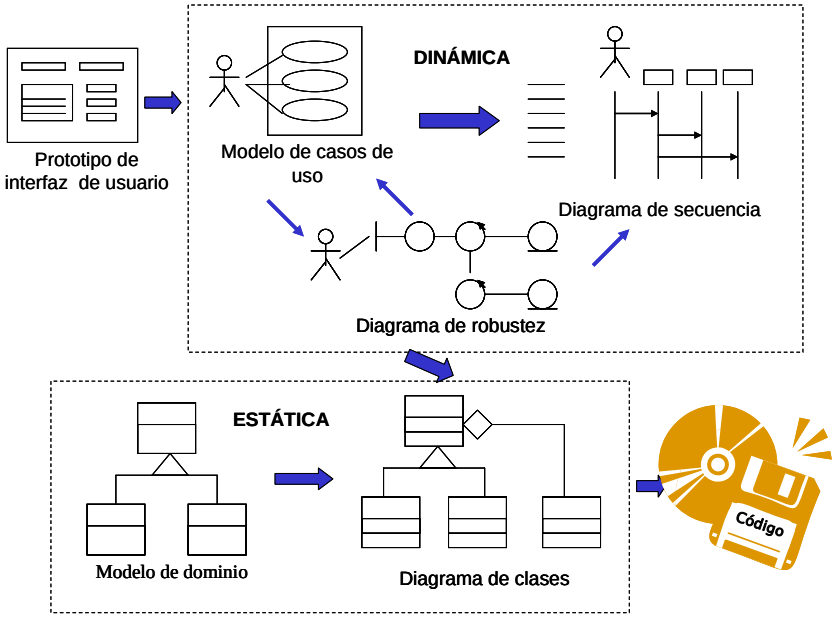


Figura 9. Artefactos generados por ICONIX

La metodología ICONIX para el desarrollo de sistemas se comprende cuatro disciplinas: Análisis de los requerimientos, Análisis y Diseño Preeliminar, Diseño e Implementación.

4.1 Análisis de los requerimientos con ICONIX

El análisis de requerimientos es la etapa de desarrollo de sistemas que se aboca a la definición de qué se quiere que haga el software y bajo qué restricciones éste correrá. Las actividades básicas a realizar, en el análisis de los requerimientos, son las siguientes, mismas que se explican en las subsecciones que siguen:

I. Identificar los objetos del dominio del mundo real y las relaciones de generalización y/o agregación entre estos objetos. Hacer el diagrama de clase de nivel alto.
II. Si es factible, elaborar un prototipo rápido del propósito del sistema. O recopilar la información importante que le dé legalidad a la reingeniería.
III. Identificar los casos de uso, usando los diagramas.
IV. Organizar los casos de uso en grupos. Capturar esta información en paquetes de diagramas.
V. Asignar los requerimientos funcionales a los casos de uso y objetos de dominio.

4.1.1 Identificar el dominio del mundo real.

El término del dominio del problema se refiere, al área que compatibiliza las cosas del mundo real y los conceptos relacionados al problema que el sistema a diseñar debe resolver. De esta forma, modelar el dominio, es la tarea de descubrir los objetos (clases) que representan esas cosas y esos conceptos. A saber:

- Sustantivos y frases nominales que se convierten a **objetos** y **atributos**.
- Verbos y enunciados que se convierten en **operaciones** y **asociaciones**.
- Frases posesivas que indican que el nombre deberá ser **atributo** en lugar de **objeto**.

4.1.2 Descubriendo las clases.

Una clase se define como una descripción de un grupo de objetos con propiedades similares. Para lograr la lista de clases, ICONIX propone lo siguiente:

- Elaborar una lista de requerimientos relevantes del dominio del problema.
- Subrayar en negritas, los sustantivos y las frases nominales.
- Eliminar la duplicidad de términos, cambiar los términos del plural a singular y ordenarlos en forma alfabética.
- Examinar de la lista, los candidatos a clases y eliminar los términos irrelevantes o incorrectos.

4.1.3 Construcción de Generalización.

Una generalización es una relación en la cual una clase es un refinamiento de otra clase. Otro término de una generalización es: **categoría de relación** donde, una clase es categoría de otra clase. Dentro de esta relación, la clase anterior se llama el superclase, o padre, mientras la última clase es la subclase o hija.

La subclase hereda los atributos y métodos definidos para la superclase, y también las asociaciones en que la superclase está envuelta. Las subclases pueden usar estos rasgos de su superclase tal cual o puede modificarlos y los métodos de la subclase no están disponibles a la superclase.

4.1.4 Construcción de Asociaciones entre clases.

Una asociación es una relación estática entre dos clases. Una asociación muestra dependencia entre dos clases pero no acciones. Las acciones involucradas en dos clases son capturas por las operaciones y sus respectivos mensajes.

- Construir la lista de candidatos a asociaciones desde la lista de verbos y enunciados, así como, también, desde el conocimiento del dominio del problema.
- Refinar la lista, eliminando cualquier cosa que claramente sea una acción de otra dependencia o no relevante en este proceso.
- Identificar la multiplicidad de las asociaciones (de uno a uno y de uno a muchos).

4.1.5 Prototipo del Sistema.

Este punto se refiere a trabajar en una Interfaz gráfica del usuario para identificar los casos de uso. Esto refuerza la noción fundamental que se está formando del sistema y conocer los puntos de vista de los usuarios. También proporciona un resumen del manejo de casos de uso, significa: escribir el manual del usuario y después escribir el código. Se puede construir un prototipo en papel, diseñando algunas pantallas que estén ligadas a los casos de uso y explicando la navegación de las mismas.

4.1.6 Modelar casos de uso.

La esencia de este modelo, es capturar los requerimientos del usuario de un nuevo sistema o de un sistema existente, para detallar todos los escenarios que los usuarios deben realizar. En este sentido, debemos entenderlo como una secuencia de acciones que un actor realiza para obtener un objetivo particular o meta deseada.

- Un caso de uso se detecta como una frase verbal, en tiempo presente y voz activa.
- Como resultado del modelado de casos de uso, se describirá toda la funcionalidad del sistema. Si uno no adopta este principio básico, se corre el riesgo de tener un sistema no deseado.
- Describir por escrito los casos de uso.

4.1.7 Organizar los casos de uso en grupos.

En el UML, un paquete es una agrupación de elementos relacionados, como las clases. Se agrupan los casos de uso en paquetes, principalmente porque estos paquetes forman los límites lógicos de división del trabajo en subtemas. Una buena regla es: cada paquete debe corresponder con un capítulo, o por lo menos una sección mayor, en su manual del usuario**.**

4.1.8 Asignar requerimientos funcionales a casos de uso y objetos de dominio

Esta actividad relaciona los requisitos y los casos del uso. El punto de vista de esta metodología se resume como sigue:

- Un caso de uso describe una unidad del comportamiento.
- Un requerimiento describe una ley que gobierna el comportamiento.
- Un caso de uso puede satisfacer uno o más requerimientos funcionales.
- Un requerimiento funcional puede satisfacerse por uno o más casos de uso.

4.2 Análisis y Diseño Preeliminar.

En esta fase las actividades a realizar son las siguientes, mismas que se detallan en las subsecciones de esta fase:

I. Describir los casos de uso. La dirección de las acciones principales, las alternativas y las rutas menos frecuentes así como condiciones de errores.
II. Elaborar un análisis de robustez. Para cada caso de uso:
    - identificar un primer corte de los objetos que llevan a cabo los estados del escenario.
    - Actualizar el diagrama de clases del modelo de dominio con nuevos objetos y atributos descubiertos.
III. Finalmente actualizar el diagrama de clases para reflejar la conclusión de la fase de análisis del proyecto.

4.2.1 Describir los casos de uso.

Un caso de uso describe uno o más cursos mediante una operación del usuario. El curso básico siempre debe estar presente; los cursos alternos son optativos (pero no, se deben verse como insignificantes).

El curso básico de acción de un caso de uso es el principal camino que el usuario seguirá, bajo circunstancias normales. Un curso alternativo de acción puede presentar un camino no frecuente en el escenario, una excepción, o una condición de error.
Recomendaciones para identificar los cursos básicos y alternos de acción:

- Crear una lista de caso de uso que tenga dos áreas: una para el curso básico y otra para el alterno. No ponga nada más allí que lo distraiga.

- Pregunte, ¿qué pasa? Ésto conseguirá el curso básico de acción inicial.
- Pregunte, “y entonces, ¿qué pasa?” Siga haciendo esa pregunta hasta tenga todos los detalles de su curso básico en papel.
- Sea implacable. ¿Qué más puede pasar? ¿Hay cualquier otra cosa que pueda pasar? ¿Usted está seguro? Siga haciendo esas preguntas hasta que usted tenga un conjunto diverso de cursos alternativos apuntado.

4.2.2 Elaborar un análisis de robustez

Involucra analizar el texto narrativo de cada caso de uso e identificar la primera suposición de objetos que participarán en el caso del uso, entonces clasificar estos objetos en tres tipos:

- Los objetos fronterizos (boundary), que actores están en comunicación con el sistema
- Los objetos entidad (entity), los objetos mas utilizados en el modelo del dominio.
- Los objetos de control (control), que sirven como la "comunicación" entre el objeto fronterizo y el objeto entidad.

El análisis de robustez juega un papel esencial dentro del modelado de Objeto e incluye las siguientes revisiones:

- Revisión de sanidad. Consiste en revisar que el texto de los caso de uso sea correcto y que no se tienen especificadas conductas irrazonables o imposibles del sistema, dado el conjunto de objetos con los que se cuenta.
- Revisión de integridad. Asegura todas las direcciones de acción de los casos de uso, incluyendo los alternos.
- Descubrimiento continuo de objetos, el cual integra objetos olvidados durante el modelado del dominio y el diseño preliminar que propone que los diagramas de secuencia se sustituyan por los diagramas de robustez por ser más simples y más fáciles de leer.

4.2.3 Actualizar el diagrama de clases

En el análisis de robustez, al momento de refinar su modelo y descubrir nuevos objetos, además de actualizar el diagrama de robustez, es conveniente agregarlos a los diagramas de clases.

En este momento se identifican los atributos llaves de las clases.

4.3 Diseño.

Este proceso se basa en los productos del análisis para generar una especificación destinada a la solución lógica del problema. Su esencia es la elaboración de los diagramas de interacción, que muestran gráficamente como los objetos se comunican entre ellos a fin de cumplir los requerimientos.
Las actividades a realizar en esta fase son las siguientes:

I. Asignar comportamiento. Para cada caso de uso:
   - Identificar los mensajes que se necesitan pasar entre los objetos, los objetos y las asociaciones. Dibujar un diagrama de secuencias con el texto de los casos de uso hacia abajo e izquierda y el diseño de la información hacia el lado derecho. Continuar actualizando el diagrama de clases con los atributos y operaciones que se encuentren.
   - Si se desea, usar un diagrama de colaboración para mostrar las transacciones clave entre los objetos.

- Si se desea, usar un diagrama de estados para mostrar el comportamiento en tiempo real.

II. Finalizar el modelo estático de la información.
III. Verificar con su equipo si el diseño satisface todos los requerimientos que se han identificado.

Cada uno de estos pasos se debe desarrollar como se explica en las secciones siguientes.

4.3.1 Asignar comportamiento:

Elaborar el diagrama de secuencias considerando tres pasos:

- Copiar el texto de los casos de uso y ponerlo al margen izquierdo de la página.
- Adicionar las entidades de objetos
- Adicionar el limite de los objetos.
- Poner métodos a las clases. En este punto se debe decidir que métodos van en que clases, esto es la esencia de la interacción del modelo

4.3.2 Finalizar el modelo estático de la información.

Esta actividad, consiste en actualizar el modelo estático conforme se avanza en el diseño, adicionando los detalles del diseño de la información (por ejemplo, valores visibles y patrones).

4.3.3 Verificar diseño

Por último la actividad de verificación se resume como:

- Repase las asignaciones que se hizo de los requerimientos para los casos de uso, usando los diagramas de los casos de uso originales y los elaborados en la fase de robustez.
- Verifique que cada requerimiento se dirige por lo menos a una clase en su modelo estático
- Siga los niveles de requerimiento y las descripciones de los casos de uso en el diseño actual de los diagramas de secuencia. Verifique que los requerimientos se satisfagan en cada caso de uso.
- Si usted tiene diagramas de colaboración o estado, asegúrese que pueda rastrear el comportamiento de cada requerimiento especifico.

4.4 Implementación

Esta fase es la última, si se ha trabajado bien durante el análisis y el diseño esta fase resultará más sencilla y consta de las siguientes actividades:

I. Elaborar los diagramas de componentes. Representar los componentes de software que integrará el sistema.
II. Escribir el código. Programar en el lenguaje de programación seleccionado.
III. Ejecutar pruebas unitarias y de integración**.** Las pruebas unitarias se refieren a probar las estructuras y métodos de cada clase existente, probando todos los caminos posibles; así como pruebas de cambios de estado en los atributos; por otro lado las pruebas de integración consisten en probar los grupos de casos de uso.
IV. Hacer pruebas con el usuario. Probar la arquitectura del sistema con el usuario para validar que estén satisfechos sus requerimientos.

5 Pruebas de sistema

Cualquier pieza de software completo, desarrollado o adquirido, puede verse como un sistema que debe probarse, ya sea para decidir acerca de su aceptación, para analizar defectos globales o para estudiar aspectos específicos de su comportamiento, tales como seguridad o rendimiento. A éste tipo de pruebas donde se estudia el producto completo se les llama ***Pruebas de Sistema***.

La prueba de sistemas usualmente es de caja negra, especialmente si quien prueba no tiene acceso al código fuente del producto a probar, que es lo más frecuente.

En este capítulo se trata el proceso de prueba de sistema, incluyendo su relación con el proceso de obtención de requerimientos y el resto del proceso de desarrollo de software.

5.1 Visión sistémica de la prueba

Cuando se deben realizar pruebas, debe mantenerse un enfoque sistémico, es decir integral, que está detrás de todo desarrollo de software. Al hablar de enfoque sistémico se indica que:

a) Todo sistema tiene una serie de objetivos que le dan sentido. Esos objetivos están asociados con indicadores de éxito que permiten determinar si los objetivos se cumplen y en qué medida.

b) Todo sistema tiene una serie de elementos que lo forman y la interacción de tales elementos se orienta a satisfacer los objetivos.

c) Todo sistema tiene una frontera que lo separa de un medio ambiente. Los elementos de ese medio ambiente influyen sobre el sistema proporcionándoles una serie de entradas y obteniendo del mismo un conjunto de salidas.

d) Ningún sistema existe en aislamiento; siempre interaccionan con otros sistemas constituyendo un sistema mayor.

Al aplicar esos conceptos a la prueba de software, se obtienen una serie de principios que servirán de base para la prueba:

1. Debe asegurarse de conocer con precisión los objetivos del software a probar, así como sus indicadores de éxito. Éstos elementos se localizan en los documentos obtenidos en la etapa de recolección de requerimientos, así como en las especificaciones del software. Esta información será indispensable para preparar el plan de pruebas y será la base para iniciar el desarrollo de los casos de prueba.
2. Deben determinarse las entradas y salidas del sistema aprobar. Éste aspecto es necesario en la preparación de los casos de prueba y también en el establecimiento de procedimientos de prueba, orientados especialmente a los casos de prueba que muestran el cumplimiento de los objetivos.
3. Considerar el sistema mayor donde opera el software a probar. Generalmente es un ambiente organizacional, formado por elementos de hardware, de software y personas (usuarios). Todos estos elementos influyen mucho sobre el sistema y ayudan especialmente en la preparación de casos de prueba de situaciones no deseadas, relacionadas con datos

inadecuados, ausencia de elementos necesarios y ocurrencia de excepciones.

5.2 Vista general de la prueba de sistemas

El proceso de prueba de un sistema tiene dos etapas que pueden estar muy separadas en el tiempo: la preparación de las pruebas y la aplicación de las mismas. La primera está muy ligada a la obtención de requerimientos, por lo que ocurre en las primeras etapas del proyecto, mientras que la segunda requiere del sistema completo o al menos una ***integración***, como se denomina a un producto parcial, aún no liberado, para poder aplicar las pruebas, por lo que ocurre en etapas avanzadas del proyecto. La situación exacta de estas partes depende del modelo de ciclo de vida que se haya elegido.

La etapa de preparación de pruebas incluye al menos tres actividades:

a) preparar un plan de pruebas,

b) preparar una lista de verificaciones de los requerimientos y

c) preparar casos de prueba.

Para ejecutar la segunda y la tercera actividades se requiere contar con el documento de requerimientos.

La primera etapa de pruebas provee retroalimentación para el análisis de requerimientos, identificando huecos, ambigüedades y otros problemas. También provee valiosas sugerencias para el diseño y la implementación del sistema, si apenas está desarrollando.

La etapa de aplicación de pruebas requiere del plan de pruebas y de una versión del sistema que sea ejecutable (una integración). Sobre ésta se aplicarán los casos de prueba que se prepararon, se analizarán los resultados y se identificarán posibles defectos.

Esta segunda etapa provee retroalimentación a la implementación y al diseño, mostrando posibles defectos que deben ser corregidos. También provee información que será de utilidad en la liberación del sistema, su aceptación, la estimación de su confiabilidad y para su mantenimiento.

En la Figura 10 se muestra el proceso de prueba de sistema y su relación con otros procesos.

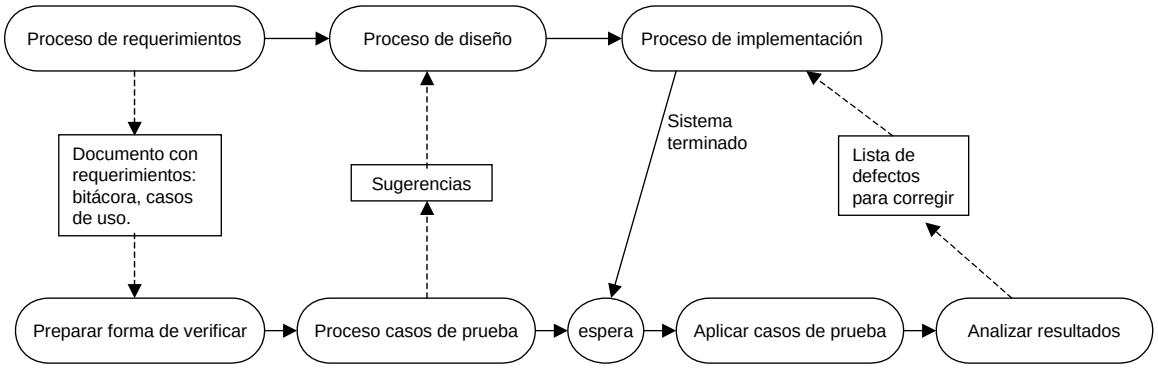


Figura 10. Proceso de Prueba de Sistema

La prueba de sistemas tiene varias suposiciones importantes:

a) Cada unidad que forma el sistema ha sido probada por separado y se han eliminado sus defectos.

b) Las interfaces humano-computadoras (de texto o gráficas) han sido probadas también por separado.

c) Se han realizado pruebas de integración para analizar la interacción entre partes del sistema y se han eliminado los defectos identificados.

El segundo punto es importante, ya que algunas veces se confunde la prueba de sistema con la prueba de la interfaz. La primera verifica la interacción de todas las partes, mientras que la segunda únicamente analiza los elementos de la interfaz y posiblemente el manejo de eventos asociados. Sin embargo, las herramientas que ayudan a la prueba de interfaz pueden utilizarse para iniciar las pruebas de sistema.

5.2.1 Plan de prueba

En la sección anterior se dio una idea general de lo que es un caso de prueba y cómo se utilizan en la práctica informalmente. Surgen varias preguntas: ¿cuántos casos serán suficientes? ¿cómo generar los menos posibles? ¿qué valores son adecuados?

5.2.2 Documento de Plan de pruebas

El plan de pruebas es un documento muy importante dentro del proceso de prueba del software. En él se explican los propósitos y enfoques de las pruebas, el plan de trabajo, los procedimientos operacionales, las herramientas necesarias y las responsabilidades. La extensión y detalle del plan debe adecuarse al proyecto y a las necesidades de la empresa, pudiendo usarse como guía el estándar IEEE 829.

A continuación se muestra una propuesta mínima de contenido, para proyectos pequeños y medianos.

a) Identificación del plan de pruebas y el sistema al que se aplica
b) Elementos a probar: qué módulos, clases, casos de uso se van a probar; cuando se emplea desarrollo iterativo, deben especificarse las prestaciones (funcionalidades) a probar y cuáles no se probarán (ya sea que se probaron antes o que se implementarán después).
c) Enfoque: vista general de la estrategia de prueba.
d) Criterio de aceptación o rechazo de un caso de prueba: criterio para dar por bueno o malo un caso de prueba al ser ejecutado.
e) Criterio de suspensión: ya sea por tiempo o por cobertura.
f) Productos a entregar: desde el propio plan, los casos y procedimientos de prueba, los resultados.
g) Tareas a realizar para satisfacer el proceso.
h) Necesidades ambientales: hardware, software y espacio de trabajo necesarios.
i) Responsabilidades: quién es responsable de cada cosa.
j) Personal necesario y si requieren entrenamiento.
k) Calendario: tiempos e hitos en el proceso.
l) Riesgos y contingencias que pueden ocurrir en el proceso de prueba.

5.3 Lista de verificaciones

Para comenzar el proceso de pruebas del sistema se parte del documento de requerimientos. En caso que no exista y se deba evaluar un sistema para aceptarlo o seleccionarlo de entre un conjunto, habrá que preparar dicho documento, aún cuando sea extemporáneo.

Un documento de requerimientos debe contener la lista de las funciones que se desea realice el software, describiéndolas y priorizándolas; también debe incluir los requerimientos no funcionales, que pueden incluir aspectos organizacionales, de rendimiento y otros.

Un documento de requerimientos bien preparado debe proveer, para cada requerimiento, una forma de verificar que se satisface. En el caso de las funciones, será una descripción y en caso de requerimientos no funcionales pueden ser especificaciones muy precisas, como puede ser el tiempo de respuesta.

Por el momento concentraremos la atención en los requerimientos funcionales, dejando los otros para una sección posterior.

La actividad de preparar lista de verificación incluye los pasos siguientes:

a) asegurarse que para cada requerimiento exista una descripción de la manera en que se verificará; si no existe, debe desarrollarse. Una buena descripción debe contener al menos el funcionamiento típico de la función a que corresponde y los principales comportamientos alternos: variaciones menos frecuentes, respuesta ante datos incompletos y fallas del ambiente.
b) revisión de las descripciones: cada descripción debe revisarse para asegurarse que se entiende claramente, que efectivamente es realizable.

5.3.1 Ejemplo 1

Una función muy común en los sistemas de información es la de identificar al usuario pidiendo un nombre y una contraseña. Una manera de describir la forma de verificar sería la siguiente:

> “Se introduce el nombre y la contraseña de un usuario registrado y entonces se activa el sistema habilitando las opciones a que tenga derecho”.

Una forma más completa podría agregar como casos alternos:

> “Se introduce un nombre y contraseña que no coinciden con un usuario registrado y entonces se activa una ventana donde se muestra un mensaje especificando el error cometido”.

5.3.2 Ejemplo 2

Suponga una función donde un cliente de una papelería desea saber que tipos de cuadernos existen y su precio de mayoreo, eligiendo el concepto “cuaderno” en un catálogo. La forma de verificar podría ser como ésta:

> “El cliente selecciona “Cuaderno” en el catálogo de productos y oprime el botón “Buscar”; el sistema mostrará una página con los diferentes tipos y marcas, con su precio al menudeo y su precio de mayoreo. Si no se oprime ningún botón en 30 segundos, aparecerá un mensaje que solicite elegir un producto y que indique que se oprima un botón”.

Debe observarse que estas frases, que indican cómo verificar, no constituyen casos de prueba, aunque sí ofrecen una guía para prepararlos. Algunas recomendaciones para estas listas son las siguientes:

a) Comenzar siempre por el funcionamiento típico deseado, el que se considerará un éxito del sistema.

b) Si existen casos alternos aceptables, describirlos después.

c) Finalmente, describir casos que se consideran fracasos, pero que están (o deben estar) considerados en la programación, generalmente en forma de validaciones.

d) Escríbalos de manera clara y concreta, en tiempo presente y evite frases condicionales como: "podría oprimir" o "debería escribir un identificador"; en su lugar, escriba: "oprima ..." o "escriba identif".

e) Si existe algún requerimiento no funcional asociado con la verificación, asegúrese de anotarlo en el documento de requerimientos. Por ejemplo, en el caso de la identificación es común considerar que tiene un máximo de tres intentos.

Si utiliza una metodología que ayude a la recolección y manejo de requerimientos, es posible que esta forma de verificar ya esté escrita. Por ejemplo, la metodología Áncora (Sumano, 2006) utiliza una bitácora donde se exige anotar la manera de verificar cada quinteta que representa una acción iniciada por un papel (usuario). Algunos enfoques de Casos de Uso sugieren describir los pasos más importantes que se siguen en cada caso de uso, incluyendo caminos alternos. Esta información es bastante similar a la que se indica como verificación. Estos dos casos se tratarán en las secciones siguientes.

Si la metodología que utiliza no incluye este elemento o si el software ya existe y no se cuenta con un documento de requerimientos adecuado, usted puede preparar esa lista de verificaciones.

5.3.3 Detalle a partir de Áncora

La metodología Áncora permite identificar de manera precisa la funcionalidad requerida por el usuario y ofrece una serie de artefactos útiles en la preparación de pruebas. El principal de éstos es la Bitácora de Desarrollo.

La Bitácora de Desarrollo (ver Figura 11) contiene la lista de quintetas[2] diferentes que se identificaron en los Guiones de la propuesta computacional del sistema; para cada quinteta incluye un campo que indica cómo verificar y otro con el tiempo estimado de desarrollo. Para fines de las pruebas interesan los dos primeros.

Quinteta	Cómo verificar	Tiempo estimado

Figura 11. Bitácora de Áncora

[2] Una quinteta incluye un papel, una acción (verbo), un utensilio principal, un utensilio auxiliar opcional y una periodicidad opcional.

Usualmente el campo “cómo verificar” contiene únicamente una vaga descripción de lo que se hace y posiblemente incluye aspectos que corresponden más bien a requerimientos no funcionales, que no fueron identificados por separado. Entre éstos pueden incluirse número máximo de intentos para una operación o alguna costumbre de la organización donde operará el sistema. Un ejemplo de cómo verificar es el siguiente:

Quinteta	Cómo verificar
Se anota compromiso en libreta	La secretaria selecciona una fecha y hora en la ventana de libreta electrónica y anota el compromiso; si no existen conflictos, el sistema confirma que quedó anotado. Si existe conflicto para esa fecha y hora o se omite algún dato, el sistema avisa y no lo deja registrado.

Como se indicó antes, esta forma de verificar debe revisarse y completarse, de modo que se asegure contar con:

a) descripción de la acción iniciada por el papel (primer elemento de la quinteta) y la respuesta del sistema (lo que será visible y quizá parte no visible, como actualización de una base de datos).

b) descripción de variantes del casó típico, si los hubiera.

c) descripción de al menos un caso fracasado.

Los casos fracasados pueden incluir algunos de los siguientes:

1. Equivocación en los datos proporcionados por el papel
2. Falta de disponibilidad de algún componente del sistema (pieza de software, base de datos, etc)
3. Falla del hardware
4. Otro problema del ambiente del sistema

5.3.4 Detalle a partir de Casos de Uso

Una alternativa muy popular para definir la funcionalidad de un sistema es el empleo de casos de uso. Para éstos existe una notación semiformal dentro del estándar UML, orientada a su representación gráfica, pero existen muchas variantes acerca de los detalles que acompañan a dicha notación. En esta sección usaremos algunas ideas tomadas del método ICONIX [Rosemberg, 1999], del Proceso Unificado [Jacobson et al, 2000] y del enfoque de Select Perspective [Select, 2005], que se describen brevemente.

En todos los métodos que utilizan el Lenguaje Unificado de Modelado (UML), la descripción general de lo que hace el sistema se representa en un **Modelo de Casos de Uso**, el cual contiene actores, casos de uso y relaciones entre ellos.

Un Caso de Uso corresponde a una forma en que un actor utilizará el sistema, es decir, a una funcionalidad o prestación. Cada caso de uso se representa gráficamente como un óvalo con un texto que indica su función, usualmente un verbo o una frase verbal. Usualmente la forma gráfica no es suficiente para documentar las prestaciones de un sistema y se acostumbra agregar una descripción textual, que no es parte del estándar de UML. Algunas formas de realización son las siguientes:

a) **Proceso Unificado**: la forma gráfica del caso de uso se complementa con el **Flujo de Sucesos**, que es una descripción textual de la secuencia de acciones que forman el caso de uso. Puede haber varios niveles de detalle.

b) **ICONIX:** se recomienda describir en forma de texto la serie de acciones que indican la función que realiza el caso de uso. El autor recomienda iniciar con una sola frase eneral y luego detallarla.

c) **Select Perspective**: en este método (y su herramienta correspondiente) se describe cada caso de uso por medio de un diálogo donde se indican las acciones del actor y las respuestas del sistema, de manera alternada

Como puede verse, en realidad los tres constituyen una misma forma de texto, con diferentes detalles pero con el mismo concepto.

Ejemplos.

El siguiente es un ejemplo tomado de [Jacobson et al, 2000]

> "Caso de uso **Pagar Factura**:
> El comprador estudia la factura a pagar y verifica que se corresponde con el pedido original.
> El comprador planifica el pago de la factura por banco.
> Cuando vence el día de pago, el sistema revisa si hay suficiente dinero en la cuenta del Comprador. Si tiene suficiente dinero disponible, se realiza la transacción."

La descripción textual de los casos de uso contiene, en cierta medida, la información necesaria para verificar su cumplimiento. Si no fuera suficiente, deberán detallarse más.

5.4 Casos de prueba

La forma de verificar de las diversas funcionalidades de un producto de software, descritas en el formato de Áncora o en Casos de Uso, son el punto de partida para la preparación de casos de prueba y, en ocasiones, de procedimientos de prueba.

Las funcionalidades o prestaciones de un sistema pueden separarse en dos grupos:

> a) aquellas que reciben un conjunto de entradas más o menos simultáneas y a partir de ellas generan un resultado y
>
> b) las que requieren una serie de interacciones con el actor(usuario), en cada una de las cuales éste introduce una serie de datos.

El primer caso ocurre cuando las entradas deben completarse antes de que el sistema se lance a realizar una función, básicamente sin retroalimentación que pueda influir en el usuario. Es el caso de una sola variable, una sola acción a través de un botón o de un *Enter*, pero también incluye la lectura de listas de datos cuyo proceso se realiza cuando la lista ha terminado. Un ejemplo típico, en interfaces gráficas, donde resulta muy común que deban llenarse varios campos (por ejemplo nombre, dirección, teléfono y rfc) y al final se oprime un botón el cual indica al sistema que la entrada está completa y puede aplicar la función seleccionada.

El segundo caso ocurre cuando la entrada ocurre en varias etapas donde una de ellas genera una respuesta parcial del sistema, la cual influye en la siguiente, ya que el usuario no puede indicar las entradas sin tomar en cuanta la salida intermedia. Sin embargo la función completa requiere de terminar todas las etapas. Un ejemplo de éste tipo es la siguiente entrada de una función de venta:

- "El empleado selecciona la opción "Consultar" y el sistema le muestra una lista de productos;
- El empleado marca los productos que le interesan y al terminar oprime el botón "Cotizar"; El sistema le muestra la lista de productos seleccionados y los precios de cada uno, así como la disponibilidad; también deja un espacio a la derecha de cada producto para anotar la cantidad deseada;
- El empleado marca la cantidad deseada de cada uno; los no deseados los marca con un cero. Al concluir oprime el botón "Listo"; el sistema genera una orden y le solicita que indique la forma de pago;
- El empleado marca "Tarjeta" y escribe el número y la compañía y oprime "Termina"; el sistema verifica los datos con el banco y, si está de acuerdo, envía mensaje y un muestra un botón "Imprimir factura";
- El empleado oprime el botón "Imprimir factura" y el sistema lo hace."

En el ejemplo es claro que existen cinco etapas de entrada de datos para completar una sola función. Lo que el empleado puede ingresar en cada una depende de la anterior.

En las tres subsecciones siguientes se describen los aspectos generales de la preparación de casos de prueba y los detalles cuando ocurre cada uno de los casos mencionados.

5.4.1 Forma general de los casos de prueba

La idea básica para la preparación de casos de prueba a partir de la forma de verificar que se trató en la sección anterior, es como sigue:

1) Se toma la parte típica de la forma de verificar, es decir la parte exitosa de la función. Para esa parte se genera un grupo de casos de prueba, empleando algunos de los métodos descritos en el Capítulo 2 (dominios y valores a la frontera) o bien por otros que se tratarán en capítulos posteriores, como el de grafo causa-efecto o el de máquinas de estados.
2) Si existen caminos alternos exitosos, se hace lo mismo que en el apartado anterior.
3) Si existen caminos alternos que terminen en fracaso, se aplica el mismo procedimiento del primer apartado.
4) Considerando el enfoque de sistemas para prueba (ver Sección 6.1), se agregan casos de prueba para situaciones no previstas en los apartados anteriores, que incluirán:
    - Ausencia de algún elemento del sistema (biblioteca, clase, archivo)
    - Falta de disponibilidad de recursos necesarios (acceso a sitio remoto, acceso a base de datos, etc.) o errores en su direccionamiento *(path*)
    - Equivocaciones del usuario (datos críticos vacíos, llaves duplicadas, tipos de datos erróneos, etc.)
    - Situaciones del medio ambiente que afecten negativamente la operación del sistema (sobrecarga del equipo o la red, dispositivos en problemas (no listos, sin papel, etc.)

Muchas de las situaciones incluidas en los apartados 3 y 4 corresponden a problemas de validación y manejo de excepciones que deben tomarse en cuenta y deben producir salidas en forma de mensajes de aviso o toma de valores por omisión.

5.4.2 Entrada simultánea

Cuando las entradas son simultáneas, usualmente la generación de casos de prueba no tiene mayores problemas y no es necesario hacer más. Recuerde que los casos de prueba utilizan valores específicos de las variables de entrada o salida, lo que los distingue totalmente de las formas de verificar, que pueden ser más genéricas.

Como se trata de prueba de todo un sistema que rara vez trabajará en aislamiento, los casos de prueba que toman en cuenta el medio ambiente son muy importantes y a veces no basta representar los casos de prueba como se hizo en ejemplos de los capítulos anteriores. En muchos casos se requiere considerar las **condiciones** (mencionadas en el Capítulo 1) en que se da la prueba y en ocasiones debe considerarse la manera de lograr dichas condiciones, a través de un **procedimiento de prueba**.

Las condiciones describen el contexto deseado al realizar una prueba, como puede ser la presencia de ciertos datos en la base de datos, la ausencia de otras aplicaciones que compitan por los recursos o el estado específico de un objeto o una estructura de datos (por ejemplo una pila vacía o llena al tope). Las condiciones deben expresarse en forma de predicados específicos, sin indefiniciones.

Ejemplos de condiciones:

El usuario “Gonzalo” está registrado en la base de datos.
La base de datos está en el path “c:\unaaplicacion\bd”
La red está desconectada.
El producto “Jabón” no está registrado en la base de datos.

Las condiciones pueden ser de entrada y de salida, siendo más comunes las primeras. Puede haber varias condiciones para un solo caso de prueba. Los casos de prueba con condiciones quedan como el ejemplo de la Tabla 7.

Tabla 6. Casos de prueba con condiciones

Condiciones entrada	Entradas	Salidas esperadas	Condiciones salida
El usuario “Gonzalo” está registrado en la BD con contraseña “W3fY74”	usuario=”Gonzalo” contra=”W3fY74”	“Bienvenido, Gonzalo”	El menú principal se activa
El usuario “Gonzalo” está registrado en la BD con contraseña “W3fY47”	usuario=”Gonzalo” contra=”W3fY74”	“Error en contraseña”	El sistema termina ejecución.
La red está desconectada	url=”http:www/Elsitio .org”	“Sitio inaccesible”	
La red está desconectada	url=”http:www.Elsitio .org”	(se muestra la página www.Elsitio.org)	

En algunos casos las condiciones resultan complejas de expresar o no resulta fácil verificar su cumplimiento. También puede suceder que las pruebas propuestas no sean inmediatamente identificables para el probador o para una persona que revise los casos de prueba. Esto puede suceder cuando no hay una manera

directa de probar un resultado y deben aplicarse varios pasos. En estas dos situaciones se hace necesario un **procedimiento de prueba**. Éste consiste en una serie de pasos que deben realizarse para aplicar un grupo de casos de prueba.

Un procedimiento de prueba puede tener tres secciones:

a) preparativos para la prueba

b) instrucciones especiales para aplicar los casos de prueba

c) instrucciones para verificar los resultados y restaurar el estado original.

Los preparativos para la prueba incluyen la creación de elementos auxiliares (como archivos de datos y objetos), carga de registros de prueba a una base de datos y aplicación de otras funciones del sistema como antecedente para la función que se desea probar.

Las instrucciones para la prueba pueden incluir aviso de reiniciar el estado antes de cada caso de prueba o dejar que trabajen de manera sucesiva, sin reiniciar el estado.

La parte final puede incluir la verificación de resultados insertados en la base de datos o revisión de documentos impresos, así como retirar de la base de datos los registros usados para las pruebas.

Ejemplo.

1. Salve la base de datos.
2. Utilice una copia vacía de la base de datos.
3. Inserte los registros que se indican en el archivo “datosPrueba.txt”
4. Asegúrese de tener deshabilitadas las conexiones de red.
5. Aplique todos los casos de prueba, uno a uno, sin reiniciar la base de datos.
6. Verifique que en la base de datos haya sido eliminado el registro del producto “Café en polvo 250 gr.”
7. Verifique que en la base de datos la cantidad del producto “Azúcar cubos” indique 47.
8. Elimine la base de datos de prueba y restaure la original.

Estos procedimientos de prueba pueden realizarse manualmente, programarse en forma de script o dentro de un programa auxiliar para pruebas. El uso de scripts es muy común cuando se realizan pruebas automáticas.

5.4.3 Entradas en varias etapas

La sección anterior trató el caso en que se tiene un conjunto de entradas más o menos simultáneas. En ésta se tratará el segundo caso, cuando la entrada a un sistema se da en etapas que no resultan independientes.

Este caso puede probarse con el mismo procedimiento del caso sencillo, siempre y cuando se separe cada etapa en un caso de prueba y todos los pasos anteriores se incluyan en un procedimiento de prueba. Sin embargo éste enfoque resulta poco práctico, ya que se repetirán muchos pasos innecesariamente.

Ahora bien, la prueba de varias etapas de manera encadenada obliga a realizar un procedimiento de prueba donde se van detallando los datos de entrada en cada etapa y las salidas intermedias esperadas. Si cualquier paso de una salida intermedia difiere de lo esperado, se suspende la ejecución y se marca una falla.

Esta manera de probar usando un procedimiento que cubra las etapas requiere cuidados especiales en la selección de valores. A diferencia de un caso sencillo, donde se conocen los diversos dominios (o su equivalente, según el método), en este caso deben considerarse valores que obliguen a seguir un camino de ejecución determinado, que permita cubrir las diferentes etapas. Existirán valores que pasan la primera etapa, pero que no sigan adelante con la segunda y otros que pasen tres etapas y fracasen en la cuarta.

Por ejemplo, considere un sistema de inscripciones a diversos cursos, con diferentes precios, de acuerdo al siguiente procedimiento de prueba:

1. Se registra el nombre del alumno y otros datos del mismo, terminando al oprimir "Registra"; el sistema lo registra y muestra un número de control en un campo de la ventana, habilitando el botón "Arancel". Si ya existe un alumno con el mismo nombre, rechaza el registro enviando un mensaje.

2. Cuando aparece el número de control se anota el tipo de curso y se oprime "Arancel"; el sistema muestra la cantidad a pagar y habilita el botón "Pago".

3. El alumno indica si el pago es en efectivo o con tarjeta (y da el número) y oprime "Pago"; el sistema verifica y genera un recibo.

Si se desea probar la primera etapa por separado, se pueden dar nombres de alumnos no registrados y otros ya registrados. Sin embargo, si se desea probar todo el procedimiento hasta generar el recibo, sólo se pueden elegir nombres no repetidos; los otros producen salidas antes de generar el recibo. Igualmente, si no se elige un curso válido o ninguno le interesa al alumno, el procedimiento se quedará en la segunda etapa, sin concluir.

Así pues, cuando se prueba en este tipo de secuencias, debe tenerse cuidado con la selección de valores que lleguen hasta el final.

Recuerde que cada unidad ya fue probada por separado, así que aquí no se trata de volver a probar cada parte, sino el sistema como un todo.

5.4.4 Beneficios a partir de la preparación de casos de prueba

Hasta aquí se ha descrito como preparar listas de verificación de los requerimientos y la preparación de casos de prueba a partir de ellos, lo que llevará a asegurar el cumplimiento de todos los requerimientos.
Como se dijo al principio, la preparación de casos de prueba ocurre en las etapas tempranas del desarrollo del software, cuando todavía no se ha diseñado ni escrito el mismo, mientras que su ejecución ocurrirá en etapas más avanzadas.

Sin embargo, desde la preparación de los casos de prueba y los procedimientos asociados ya se obtienen ganancias para la calidad del software. Entre otras se tienen:

a. Las listas de verificación del cumplimiento de los requerimientos ayudan a realizar revisiones técnicas al software antes de iniciar su diseño,

identificando los aspectos más débiles y los que pueden generar problemas; como deben estar completos, ayuda a asegurarse que no existan requerimientos que no se pueden comprobar. Por ejemplo, si hubiese un requerimiento "El sistema debe ser totalmente amigable", sin una forma de verificar tal cosa, debe revisarse qué se entiende por "amigable" y, más aún, qué debe entenderse por "totalmente amigable".

b. La preparación de los casos de prueba obliga a reflexionar acerca de los valores aceptables y los no aceptables, así como las situaciones no deseadas que podrían ocurrir, como los errores del usuario o la falla de la base de datos. El tener presentes estas situaciones permite diseñar una buena validación de las entradas y aseguramiento del ambiente, en vez de darlo por hecho. Por ejemplo, en Delphi es común hacer programas que abren automáticamente la base de datos, desde el inicio; si por alguna razón la base de datos tiene problemas, tales programas se cierran sin más trámite, dejando al usuario frustrado. De haberlo enido en cuenta, se habría dejado la base cerrada para abrirla más adelante, asegurándose de su buen funcionamiento.

c. Al preparar los casos de prueba se pueden identificar la funciones más complejas, con más casos posibles, lo que hará que se ponga cuidado especial en su desarrollo, ya que muy probablemente serán las funciones que más defectos tendrán.

5.4.5 Ejecución de casos de prueba

Cuando se tiene lista una **integración** de un sistema, es decir, el sistema completo o una parte del mismo que ya satisface un cierto número de requerimientos y ha sido probado a nivel de unidades y de integración de partes, se puede pasar a ejecutar los casos de prueba que se prepararon con anterioridad. Esta prueba puede realizarse en varias etapas o maneras:

a) prueba de humo
b) pruebas de regresión
c) pruebas alfa realizadas por el equipo desarrollador en sus instalaciones
d) pruebas beta realizadas por el cliente en sus instalaciones, bajo supervisión del equipo desarrollador

Las pruebas de humo son de tipo muy tosco y preliminar; su nombre viene del campo de la electrónica antigua, donde se probaba un circuito nuevo conectándolo y observando si no se quemaba, es decir, si no lanzaba humo. El software no puede producir humo, pero si puede fallar estrepitosamente. Un sistema recién integrado se pone en ejecución sin preocuparse mucho de los casos de prueba, sólo para observar si al menos comienza a trabajar. Como se mencionó en alguna sección, existen programas que al abrirse de inmediato terminan sin avisar qué sucede (echan "humo")[3].

A cada iteración de un proyecto se logra una integración del software, la cual debe probarse. Los defectos que se identifique se corrigen y se agregan nuevas funcionalidades antes de llegar a otra integración, repitiéndose el proceso.

En cada iteración habrá muchas pruebas que ya se cumplían en la iteración anterior, pero que deben volver a probarse, debido a los cambios, correcciones y modificaciones. Estas

[3] Ver descripción en la Wikipedia: http://wikipedia.org

pruebas que se repiten se denominan pruebas de **regresión**, y definitivamente deben automatizarse, para lo cual se escriben los procedimientos de prueba a manera de scripts.

Cuando el sistema está completo, se realizan pruebas intensivas para ganar confianza en su funcionamiento. Inicialmente las pruebas se realizan en el ambiente del equipo desarrollador, aunque buscando su parecido con el ambiente del cliente. La duración de ésta etapa es uno de los asuntos más delicados que debe resolver el responsable del proyecto: si libera el software muy tarde, quizá pierda sus ganancias y aún al cliente; si libera antes de tiempo puede dejar demasiados defectos. Para auxiliar a quien debe tomar la decisión existen técnicas estadísticas que permiten estimar los defectos remanentes, aún no descubiertos, y decidir cuando ese número es inferior a un umbral dado (ver trabajo de [Lambuth, 2002]). Por ejemplo, ciertas compañías japonesas liberan el software cuando se estiman 0.2 defectos por millar de líneas de código (ver trabajo de [Yamaura, 1996]).

Una vez concluidas las pruebas alfa, se pasa a pruebas en el ambiente del cliente, pero bajo supervisión del desarrollador, para validar el software contra las necesidades reales. Es probable que aparezcan nuevos defectos, ya que los usuarios reales no se comportan igual a los probadores de software.

5.5 Otras pruebas de sistema

Hasta ahora se han tratado pruebas de tipo funcional, es decir, que verifican los requerimientos funcionales o prestaciones deseadas del sistema. Sin embargo en muchos casos no son suficientes; apenas son una condición necesaria pero no suficiente.
Otras pruebas que pueden resultar necesarias son:

a) pruebas de rendimiento: donde se evalúa el número de transacciones por unidad de tiempo o el tiempo de respuesta a determinadas funcionalidades
b) pruebas de estrés: donde se somete el sistema a casos extremos, no esperados en situación normal, exagerando la cantidad de datos o la frecuencia de transacciones.

6 Pruebas de Integración orientadas a objetos

Las pruebas de integración han sido, hasta ahora, las menos estudiadas y comprendidas y las más evitadas. Aún en empresas que dan poco valor a las pruebas, los desarrolladores realizan pruebas de unidad, aún cuando sean informales. También se efectúan algunas pruebas de sistema, al menos poco antes de entregar el software. Sin embargo, las pruebas de integración no se ven como necesarias.

Las pruebas de integración orientadas a objetos se enfocan a la **interacción** entre unidades, suponiendo que cada una fue probada a nivel de unidad. A este nivel se mezclan aspectos estructurales que relacionan las maneras de interactuar de las unidades y también los aspectos típicamente funcionales.

Las pruebas de integración se ven dificultadas por el polimorfismo y la liga tardía al tiempo de ejecución. También, en sistemas distribuidos, el uso de objetos remotos resulta problemático.

Según Binder, las pruebas de integración pueden determinar problemas de los tipos siguientes:

a) Problemas de configuración: se producen fallas debido a que las versiones de los diferentes elementos pueden resultar incompatibles, por mal control de versiones, o por usar una versión equivocada.
b) Funciones faltantes, traslapadas o que tienen conflictos: Una clase puede invocar un método de otra que aún no está implementado o que fue olvidado; también puede suceder que la función invocada no realice lo que se deseaba.
c) Uso incorrecto o inconsistente de archivos y bases de datos: Los archivos y bases de datos son elementos importantes al integrar un sistema, pero pueden estar ausentes o tener claves imprevistas o restricción en el número de usuarios concurrentes o formatos diferentes al previsto.
d) Violaciones a la integridad de los datos: Al manejar bases de datos, si no se respetan las restricciones de integridad, se producirán errores que quizá no se anticiparon al crear las clases.
e) Llamadas a método equivocado, debido a errores de codificación o a liga inesperada al tiempo de ejecución: Como los objetos usan liga dinámica y a veces los nombres de los métodos no dicen su función, es posible invocar métodos equivocados; el polimorfismo lo agrava, ya que no se sabe con exactitud qué objeto será el que interactúe en un momento dado.
f) Una unidad cliente envía un mensaje que viola las precondiciones del servidor: por ignorancia o descuido, es posible que una clase solicite un servicio pero no cumpla las reglas debidas en los parámetros que se envía y eso provoca el rechazo de la clase que debe proporcionar el servicio; por ejemplo esperaba un número positivo y recibe uno negativo.
g) Objeto equivocado como destinatario en caso de polimorfismo: se esperaba un objeto (por ejemplo un cuadrado) y llegó otro (por ejemplo un triángulo o una elipse) y no se sabe qué hacer con él.
h) Parámetros erróneos o valores equivocados: los parámetros esperados no corresponden a los enviados; por ejemplo esperaban un entero y llegó un real; otro caso sería que falten parámetros; también puede suceder que el valor no

corresponda al rango permitido. Algunos de estos problemas no se notan al compilar debido a la liga dinámica.

i) Problema de precisión en parámetros: similar al anterior, pero con tipos parecidos; puede suceder que esperaba un entero de 16 bits y recibe uno de 32 o viceversa.
j) Fallas causadas por mal manejo de memoria: en ocasiones una clase crea y destruye objetos de otra clase y puede originar problemas si no lo hace correctamente (en lenguajes como Java esto es poco frecuente).
k) Uso incorrecto de servicios del S.O., ORB[4] o de máquina virtual (como la de Java): similares a los anteriores, pero referidos a invocaciones de servicios del sistema operativo o software intermedio, en vez de clases del usuario.
l) Conflictos entre componentes: puede ocurrir una falla en una clase cuando otra está corriendo, debido a mal manejo de concurrencia (ya sea a nivel de hilo o de proceso).
m) Recursos insuficientes: el destinatario no asigna recursos necesarios la creación de objetos va disminuyendo los recursos del sistema y puede suceder que se excedan las capacidades asignadas a una aplicación. Por ejemplo, Delphi no destruye automáticamente las ventanas en desuso, sólo las oculta; el usuario debe cuidar de no dejar demasiadas.

Existen muchos enfoques a las pruebas de integración, destacando los de pares, los de vecindad y los progresivos (bottom-up y top-down). En el caso de software de objetos, cuando se usa algún método derivado de casos de uso, éstos brindan una manera práctica de realizar pruebas de integración, guiándose por los diagramas de colaboración o los diagramas de secuencia asociados con cada caso de uso. En estas notas seguiremos la guía de los casos de uso.

6.1 Caso de Uso como unidad de pruebas de integración

Cuando se desarrolla software orientado a objetos guiado por casos de uso, éstos se convierten en la unidad mínima de funcionalidad. Para el diseño del software, se desarrolla un conjunto de diagramas de colaboración o de secuencia (pueden ser ambos) que representan varios escenarios de la misma situación. Al menos se acostumbra incluir un diagrama para el caso típico exitoso y otro para un caso alterno, generalmente una falla típica. Como los diagramas de colaboración y los de secuencia son equivalentes en muchos aspectos, nos referiremos únicamente a los diagramas de secuencia.

Un diagrama de secuencia presenta varios elementos importantes para las pruebas de integración:

a) El conjunto de clases (objetos) que interactúan.
b) Las interacciones entre los objetos, mostradas como secuencias de mensajes que activan métodos.

Cada caso de uso ameritará varios casos de prueba; al menos uno por cada diagrama de secuencia que se haya preparado, pero usualmente más. El mayor número de pruebas se origina por dos elementos:

a) el estado de los diversos componentes del diagrama y
b) los posibles valores que pueden tomar los diversos parámetros de los métodos invocados.

[4] ORB: Object Request Broker: tipo de software intermedio que permite crear aplicaciones distribuidas. Un ejemplo es CORBA.

6.1.1 Estado de los elementos

En un diagrama de secuencia se muestran, como ya se dijo, una serie de objetos, que pueden representar clases internas del sistema o elementos almacenados en disco, como archivos o bases de datos. Además, de manera implícita, pueden existir otros elementos que afectan la ejecución del software, como pueden ser: la existencia de una red, la presencia o ausencia de otros elementos de software con el que se interactúa (sistema operativo, procesos externos) o incluso hardware externo que tiene influencia sobre el software (lector de tarjetas de crédito, sensores en equipo industrial, etc.). Algunos elementos podrían estar anotados al márgen del diagrama de secuencia o en una nota.

Todos los elementos mencionados influyen dentro de la ejecución del software, afectando la interacción entre los objetos, algo que no importaba al nivel de pruebas de unidad, pero que sí interesa al integrar las partes. De éstos elementos interesará básicamente su estado, expresado en forma de proposiciones lógicas (recordar que sólo pueden ser ciertas o falsas). Así podemos tener, a manera de ejemplos:

El objeto X está presente
El archivo está presente y abierto
La tabla está presente pero no es accesible
La red está desconectada
El objeto Y existe, pero es de una versión anterior a la esperada
Existen al menos diez procesos corriendo al mismo tiempo

En general, se puede decir que los elementos tienen al menos los estados que se muestran en la Tabla 1.

Tabla 1 Posibles estados según el tipo de elemento

Tipo de elemento	Posibles estados
objeto	Existe, no existe, existe pero es de versión inadecuada
Recurso compartido (archivo, BD)	Existe, No existe, No es accesible (falta autorización, exceso de usuarios, bloqueado)
Dispositivo (red, impresora)	Listo, en problemas, desconectado

Otro aspecto importante que corresponde al estado es la situación interna de los objetos involucrados. De éste se habló en el caso de pruebas de unidad, correspondiendo a los valores de los atributos propios del objeto.

6.1.2 Valores de parámetros

El otro aspecto importante que afecta al número de casos de prueba corresponde a los parámetros de los métodos invocados. Si un método carece de parámetros, bastará un caso de prueba que lo invoque. En caso de existir parámetros, existirán valores aceptables y no aceptables y además los valores aceptables pueden dividirse en subconjuntos con comportamientos similares. Esta separación en subconjutos se trata en el método de dominios (particiones).

Respecto de las interacciones, existe otro tipo de partición que resulta relevante y que se tratará en la subsección que sigue.

6.1.3 Conjuntos de datos adecuados

Cuando existen dos clases que interactúan, una de ellas llama algún método de la otra. Llamaremos ClaseA a la primera y ClaseB a la segunda, que es la que será invocada. Cuando un método en la ClaseA, que llamaremos MetA, es invocado, ya sea por un actor o por otra clase que por ahora no importa, existe un conjunto de datos de entrada aceptables como parámetros. Algunos valores de entrada generarán llamadas a la ClaseB, digamos al método MetB, y otros no; estos últimos producirán acciones locales. Igualmente, al invocar a MetB existen todo un conjunto de valores que pueden ser atendidos por éste, pero algunos nunca podrían ser producidos por MetA. Estas combinaciones corresponden a otras funciones de MetB que no corresponden a interacciones entre ClaseA y ClaseB.

Para aclarar las cosas, llamaremos ValoresA al conjunto de valores que pueden recibir los parámetros del método MetA y este conjunto se descompone en dos: AtraviesaB y NoAtraviesaB. El primero es el conjunto de valores que están en ValoresA y que producen una llamada a MetB. El segundo incluye valores que están en ValoresA pero nunca generan llamadas a MetB. En la Figura 1 se ilustra esta situación.

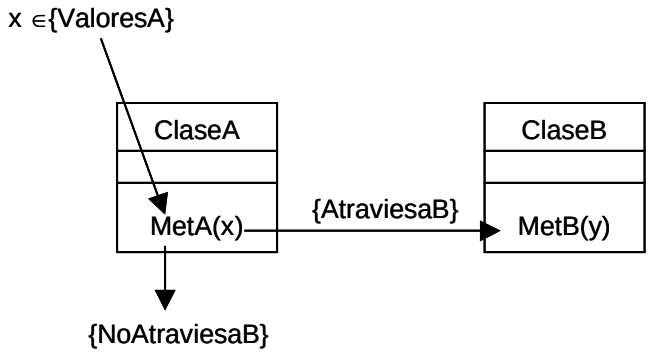


Figura 1 Valores que llegan a ClaseB y otros que no lo hacen

Ahora bien, para probar una interacción entre las clases ClaseA y ClaseB se necesitan datos que pertenezcan al conjunto AtraviesaB. Ésto tiene que determinarse a partir de las especificaciones de las clases o por análisis del problema.

Hasta aquí se consideran dos clases, pero un camino puede cruzar cuatro o cinco clases, por lo cual deberán escogerse datos que atraviesen todas esas clases, lo cual no siempre es fácil. Para facilitar las pruebas de integración podrían limitarse las pruebas a pares de clases, aunque eso aumentará el número de pruebas.

6.2 Generación de casos de prueba

A partir de los aspectos descritos en la sección anterior, se está es situación de poder generar los casos de prueba, para lo cual se siguen los pasos mostrados a continuación:

1. Para cada diagrama de secuencia, generar los estados posibles de los elementos involucrados, de acuerdo a 7.1.1 y la Tabla 1.
2. Para cada método en las interacciones incluidas en el diagrama de secuencia determinar los conjuntos de valores adecuados, según las particiones internas y según si permiten continuar la cadena de llamadas a otras clases involucradas y seleccionar valores representativos. Ésto se hace de acuerdo con las secciones 7.1.2 y 7.1.3.

3. Con cada estado y cada representante de conjunto de datos se forma un caso de prueba, combinando los diferentes valores de cada categoría.

6.3 Ejemplo

Para entender mejor el procedimiento descrito en la sección anterior se muestran los pasos con un caso de uso correspondiente a una aplicación que realiza la identificación de un usuario empleando una clase Tclave (ver Figura 2).

El ejemplo corresponde a un pequeño programa que puede ser parte de muchos sistemas mayores, dedicado a la identificación de usuarios. El programa recibe un nombre y una contraseña y regresa un mensaje de bienvenida o uno de error, según esté registrado y con la contraseña correcta o falle la contraseña o el registro. Para la identificación se emplea una tabla con las claves y contraseñas.

En la Figura 12 se muestran las clases que participan en un caso de uso de identificación del usuario. Una es una clase fronteriza, otra sirve como clase de control y se encarga de la conexión a base de datos y la otra es la entidad que contiene las claves. Además se emplea una clase auxiliar que presenta mensajes en una ventana de diálogo.

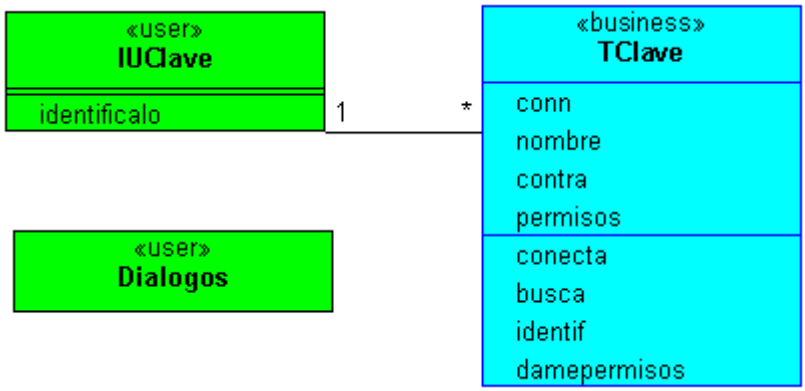


Figura 12. Diagrama de clases del ejemplo

Pasando directamente al diseño, la Figura 13 muestra un diagrama de secuencia para el escenario típico, donde el usuario se identifica satisfactoriamente.

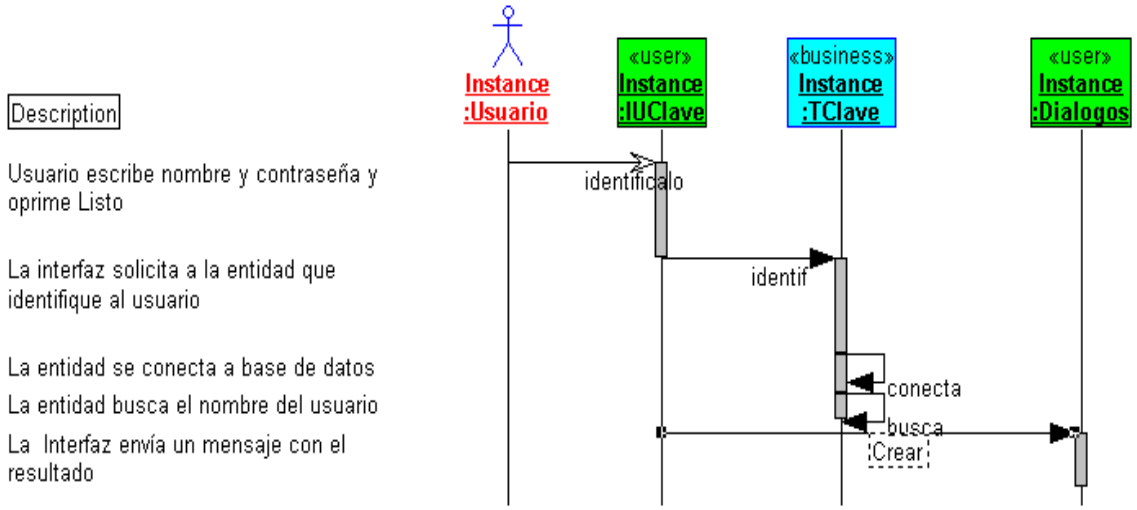


Figura 13. Diagrama de secuencia exitoso de la identificación de usuario

En las Figuras 14 y 15 se muestra el código correspondiente a este ejemplo, desarrollado en Java, usando una base de datos de Mysql, conectada a través de JDBC.

```
import java.awt.event.KeyEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Event;
import java.awt.BorderLayout;
import javax.swing.KeyStroke;
import javax.swing.JPanel;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JPasswordField;
import javax.swing.JButton;

public class IUClave extends JFrame {

	private JPanel jContentPane = null;
	private JLabel titulo = null;
	private JLabel nnom = null;
	private JTextField elnom = null;
	private JLabel ccon = null;
	private JPasswordField lacon = null;
	private JButton llis = null;
	private JTextField rres = null;
	private TClave laclave;

	private JTextField getElnom() {
		if (elnom == null) {
			elnom = new JTextField();
			elnom.setBounds(new java.awt.Rectangle(120,45,127,20));
```

```
		}
		return elnom;
	}

	private JPasswordField getLacon() {
		if (lacon == null) {
			lacon = new JPasswordField();
			lacon.setBounds(new java.awt.Rectangle(137,75,108,20));
		}
		return lacon;
	}

	private JButton getLlis() {
		if (llis == null) {
			llis = new JButton();
			llis.setBounds(new java.awt.Rectangle(90,105,81,17));
			llis.setText("LISTO");
			llis.addActionListener(new java.awt.event.ActionListener() {
				public void actionPerformed(java.awt.event.ActionEvent e) {
					System.out.println("actionPerformed()");	// TODO Auto-
generated Event stub actionPerformed()
					identificalo();
				}
			});
		}
		return llis;
	}

	private JTextField getRres() {
		if (rres == null) {
			rres = new JTextField();
			rres.setBounds(new java.awt.Rectangle(22,137,238,20));
		}
		return rres;
	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		IUClave application = new IUClave();
		application.show();
	}

	public IUClave() {
		super();
		initialize();
	}

	private void initialize() {
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.setSize(300, 200);
		this.setContentPane(getJContentPane());
		this.setTitle("Application");
		laclave= new TClave();
	}

	private JPanel getJContentPane() {
```

```
		if (jContentPane == null) {
			ccon = new JLabel();
			ccon.setBounds(new java.awt.Rectangle(35,75,91,16));
			ccon.setText("Contraseña:");
			nnom = new JLabel();
			nnom.setBounds(new java.awt.Rectangle(35,44,67,16));
			nnom.setText("Nombre:");
			titulo = new JLabel();
			titulo.setBounds(new java.awt.Rectangle(15,9,263,16));
			titulo.setText("Por favor escriba su nombre y su contraseña");
			jContentPane = new JPanel();
			jContentPane.setLayout(null);
			jContentPane.add(titulo, null);
			jContentPane.add(nnom, null);
			jContentPane.add(getElnom(), null);
			jContentPane.add(ccon, null);
			jContentPane.add(getLacon(), null);
			jContentPane.add(getLlis(), null);
			jContentPane.add(getRres(), null);
		}
		return jContentPane;
	}

	private void identificalo()
	{
		//aqui llamamos a laclave para identificar al usuario
		int qres;
		String n=elnom.getText();
		String c=lacon.getText();
		System.out.println("leido:"+n+", y:"+c+"!");
		qres=laclave.identif(n,c);
		if (qres==1)
		{
			rres.setText("El usuario "+n+" ha sido reconocido.");
			System.out.println("Permisos: "+laclave.damePermisos());
		}
		else
		{
			switch (qres)
			{
			case -1: rres.setText("Usuario "+n+", Contraseña errónea"); break;
			case -2: rres.setText("Usuario "+n+" no existe");break;
			case -3: rres.setText("Problema en conexión");break;
			}
		}
	}
}
```

Figura 14. Código de IUClave, del ejemplo

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;
```

```
public class TClave {
        private String nombre=null;
        private String contra=null;
        private String permisos=null;
        Connection conn=null;
   Statement stat=null;
   ResultSet rs=null;

        private boolean conecta()
        {
                boolean resp=false;
      System.out.println("reconocer driver");
      try
      {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
      } catch(Exception ex) {System.out.println("Fallo driver: "+ex);
        }


      try
      {
        conn=DriverManager.getConnection("jdbc:mysql://localhost/","root","antares05");
        resp=true;
      } catch (SQLException se)
      {
        System.out.println("Mensaje: "+se.getMessage());
        System.out.println("Estado: "+se.getSQLState());
        System.out.println("Error: "+se.getErrorCode());
      }
      return resp;
        }

        private int busca(String nn)
        {
                int resp=0;
      try
      {
        stat=conn.createStatement();
        rs=stat.executeQuery("use claves;");
        rs=stat.executeQuery("select * from clavebas where nombre='"+nn+"';");
        String datos;
        while (rs.next())
        {
                datos="nombre= "+rs.getString("nombre");
                datos=datos +" contrasenia= "+rs.getString("contra");
                System.out.println(datos);
                nombre=rs.getString("nombre");
                contra=rs.getString("contra");
                permisos=rs.getString("permiso");
                resp=1;
        }
```

```
      } catch (SQLException se)
        {
         System.out.println("LMensaje: "+se.getMessage());
         System.out.println("LEstado: "+se.getSQLState());
         System.out.println("LError: "+se.getErrorCode());
        }

      System.out.println("Terminamos por ahora");
          return resp;
        }

   public int identif(String nn, String cc)
   {
         int resp=0;
         int bres;
         if (conecta())
         {
                  if ((bres=busca(nn))==1)
                  {
                           if ((nombre.equals(nn))&& (contra.equals(cc)))
                                    resp=1; else resp = -1; //contraseña errónea
                  }
                  else resp =-2;  // usuario no existe
         }
         else resp=-3; //problema de conexión
         return resp;
   }

   public String damePermisos()
   {
         return permisos;
   }
}
```

Figura 15. Código de Tclave, del ejemplo

6.3.1 Estados identificables

Para el problema de la clave se tienen tres clases explícitamente definidas y un componente externo que es la base de datos de Mysql. En principio el sistema opera en un solo equipo, por lo que no interviene una red. Con ellos se identificanvarios estados posibles. En el caso de la base de datos puede suceder que no esté instalado Mysql, que la base de datos no exista, que la clave no sea correcta y otros estados indeseables, pero los agruparemos en uno, que llamaremos “con problemas”. Los estados identificados se muestran en la Tabla 8.

Tabla 7. Estados de los elementos que intervienen en ejemplo

Elemento	Estados
IUClave	Presente, ausente
Tclave	Presente, ausente
Dialogos	Presente, ausente
Base de datos	Activa, con problemas

6.3.2 Valores adecuados

La llamada inicial del actor es cuando oprime el botón Listo y se lanza el método asociado al evento, identificalo, con dos parámetros: nombre y contraseña, que toma de campos de la interfaz. Los parámetros pasan sin cambio a la clase Tclave y de ésta, el primero pasa tal cual a la base de datos. Así pues, sólo deben considerarse dos variables y eso nos deja los valores posibles que se muestran en la Tabla 9. Se incluyen valores aceptables y no aceptables.

Tabla 8. Valores de entrada a considerar

juan, orion7	nombre y contraseñas registradas
juan, qwerty	nombre registrado, contraseña mal
juan	falta contraseña
, orion7	falta nombre
ulises, df7th	nombre no registrado

6.3.3 Casos de prueba detallados

Combinando los estados y los valores a considerar, se pueden formar los casos de prueba que se incluyen en la Tabla 10. Se eliminaron algunos casos que no tendrían sentido, como por ejemplo combinar el estado de base de datos con problemas con cada valor, que sería innecesario, pues ninguno de ellos se podría comparar. Observe que los estados se reflejan en las condiciones de entrada.

Tabla 9. Casos de prueba

Entrada	Condiciones de entrada	Salida esperada	Condiciones de salida
juan, orion7	IUClave ausente	Falla de ejecución	
juan, orion7	Tclave ausente	Falla de ejecución	
juan, orion7	Base de datos con problema	Falla de ejecución	
juan, orion7	Dialogos ausente	Falla de ejecución	
juan, orion7	IUClave presente Tclave presente Base de datos activa	Mensaje de bienvenida	
juan, qwerty	IUClave presente Tclave presente Base de datos activa	Mensaje de error en contraseña	
juan	IUClave presente Tclave presente Base de datos activa	Mensaje de error en contraseña	
, orion7	IUClave presente Tclave presente Base de datos activa	Mensaje de error en nombre	
ulises, df7th	IUClave presente Tclave presente Base de datos activa	Mensaje de error en nombre	

Para automatizar las pruebas de integración orientadas a objetos se pueden emplear las mismas herramientas de la prueba de unidad, pero los casos de prueba serán un poco más largas en alguna ocasión y la verificación de resultados puede requerir más de una comprobación.

6.4 Pruebas de sistema

Las pruebas del sistema son importantes cuando se ha concluido con la implementación de un producto. Existen varias versiones, algunas realizadas por el desarrollador y otras por el cliente o un representante de éste. Las pruebas pueden ser funcionales, que son las que nos interesan aquí, así como pruebas de rendimiento, de estrés, de seguridad, de instalación y otras.

Las pruebas funcionales siempre tienen como guía los requerimientos establecidos por el cliente y se busca analizar si los satisfacen, tanto en los requerimientos positivos (qué se deseaba que haga) como en las limitaciones (qué se deseaba que no hiciera).

Las pruebas realizadas por el desarrollador a veces se dividen en Alfa y Beta. Las primeras se realizan en las instalaciones del desarrollador, mientras que las segundas se efectúan en las instalaciones del cliente, pero bajo control del desarrollador. Las pruebas que realiza el cliente por su cuenta se conocen a veces como pruebas de aceptación.

Sin importar el nombre, las pruebas funcionales se realizan de manera similar. Como se dijo, corresponden a los requerimientos y por ello se recomienda plantear los casos de prueba al tiempo de establecerlos. De hecho, el preparar los casos de prueba al mismo tiempo que se definen los requerimientos ayuda a precisarlos mejor y ayudan a comenzar mejor el proyecto. En efecto, es muy común que los requerimientos queden expresados de manera vaga; al tratar de definir casos de prueba se notará esa vaguedad y será necesario precisarlos.

Para cada requerimiento que se establezca, habrán casos que darán resultados deseados y también otros que corresponden a errores previstos y que deben tomarse en cuenta para evitar fallas. Aquí nuevamente las pruebas preparadas desde el inicio ayudan a definir mejor el software. Como regla, habrá al menos un caso de prueba típico y otro correspondiente a una excepción por cada requerimiento, aunque generalmente serán más.

La preparación de casos de prueba a nivel de sistema es independiente de la manera en que se implementó, ya sea tradicional o por objetos, ya que sólo interesa el cumplimiento de la funcionalidad deseada.

Para preparar los casos de prueba de sistema se recomiendan los métodos de dominios (particiones), de valores a la frontera y de grafo causa-efecto (también llamados de tabla de decisión).

Para realizar las pruebas de sistema pueden prepararse los casos de prueba de forma similar a las pruebas de integración, guiándose por los casos de uso, pero es más común que las pruebas las inicie un operador humano.

7 Pruebas de unidad usando el Método de rebanadas

En el software orientado a objetos la unidad mínima que tiene sentido es la clase, ya que los métodos individuales no son independientes, sino que interactúan entre sí a través de los atributos. Así pues, las pruebas de unidad deberán aplicarse a clases completas. La prueba puede dirigirse con varios métodos, como el de rebanadas y el de máquinas de estados. Para clases de complejidad simple o media resulta más sencillo emplear el método de rebanadas, que se describe en este capítulo.

El método de rebanadas ha sido adaptado de la obra de Bashir y Goel, buscando adecuarlo a Java y eliminando ciertos aspectos que lo obscurecen. Sin embargo, se recomienda revisar el material de dichos autores.

7.1 Concepto de rebanadas

Cuando se habla de objetos se considera una abstracción definida por un nombre, un estado y un conjunto de operaciones o métodos que definen su comportamiento. El estado usualmente es dinámico, afectado por la realización de los diversos métodos. Todos los objetos de una misma clase comparten el con junto de métodos y la existencia de un conjunto de atributos que representarán el estado, el cual puede caracterizarse como un conjunto de atributos con sus respectivos valores en un instante dado.

Cuando se habla de probar una clase, en realidad no se puede hacer directamente; en vez de ello se prueba a través de instancias de la misma, es decir objetos, así que en lo que sigue todo lo que se diga de clase se refiere a objetos pertenecientes a la misma.

Cuando se realizan pruebas, se busca encontrar situaciones que rompan los supuestos que dan validez a la pieza de software que se está probando (o que confirmen que sí los cumple). En el caso de los objetos, además de la función que debe satisfacer cada método, se tiene el supuesto de que el estado es consistente sin importar el orden de ejecución de los métodos. Piense por ejemplo en una clase Cuenta que represente una cuenta bancaria con un atributo saldo y dos métodos deposita y retira; no importando cuántas veces se invoque a los métodos ni el orden en que se haga, el saldo debe ser consistente, es decir, que no se pierda ni aparezca dinero y que no se pueda retirar dinero no depositado. Así pues, las pruebas de unidad tendrán que verificar de alguna manera que esa consistencia existe para todas las combinaciones de ejecución de los métodos.

Ahora bien, el ejemplo mencionado es extremadamente simple y su estado consiste de un solo atributo; en clases más complejas el estado tendrá más atributos y también existirán muchos métodos, volviendo impráctica la prueba de todas las combinaciones de ejecución de métodos. Por lo tanto, se buscarán formas de reducir el problema.

Al considerar el estado con detenimiento, puede observarse que no todos los métodos interactúan con todos los atributos, sino que cada método se relaciona con un subconjunto de éstos. Cambiando el enfoque, si se fija la atención en un atributo se tendrá que sólo un subconjunto de los métodos de la clase se relacionando una manera u otra con dicho atributo. De ahí surge la idea de “cortar” la clase de tal manera que en cada rebanada exista sólo un atributo y los métodos que se relacionan con éste. Claramente, cada rebanada será más fácil de probar que la clase entera y la combinación de rebanadas aún será inferior a la complejidad de probar toda la clase de una vez.

Formalmente podemos escribir que una clase está formada por una pareja

C = <A,M>,
donde **A** es un conjunto de atributos y **M** un conjunto de métodos. Si tomamos un atributo $\mathbf{a_i} \in \mathbf{A}$, tendremos que $\mathbf{M_i} \subseteq \mathbf{M}$ será el subconjunto de métodos que forman parte de la rebanada de $\mathbf{a_i}$.

El método de rebanadas consiste de dos grandes pasos:

1) Generar las rebanadas correspondientes a una clase
2) Para cada rebanada generar secuencias de métodos

Para la aplicación práctica del método se requieren algunos elementos adicionales, que se verán en la Sección 4, como el uso de cabos y manejadores para automatizar la ejecución.

7.2 Generación de rebanadas

Para generar las rebanadas se emplean dos estructuras de datos auxiliares: un Grafo de Llamadas Mejorado (GLLM) y una Matriz de Uso Mínima (MUM). A continuación se describen ambos y su relación.

7.2.1 Grafo de Llamadas Mejorado

Un Grafo de Llamadas Mejorado (GLLM) de una clase es un grafo constituido por uno o más árboles, donde cada árbol tiene como raíz un método de la clase y de ella parten ramas a los atributos a que hace referencia y a los métodos de la misma clase que invoca. Los métodos se representan como nodos circulares y los atributos como nodos cuadrados. Mientras sea posible se siguen expandiendo los nodos correspondientes a métodos. Nótese que los atributos son pasivos y por tanto siempre estarán en las hojas del árbol. En la Figura 16 se muestra uno de tales árboles.

El nombre de “grafo de llamadas” viene del hecho de representar las llamadas entre métodos; lo de mejorado porque incluye los atributos.

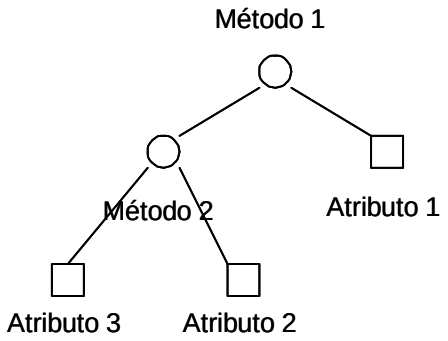


Figura 16. Grafo de llamadas mejorado

En los siguientes ejemplos se puede apreciar mejor la estructura del GLLM.

Ejemplo 1. Clase Cuenta.

Suponga que se tiene una clase que representa una cuenta bancaria un poco más desarrollada que la descrita en la sección anterior, como se muestra en la Figura 17.

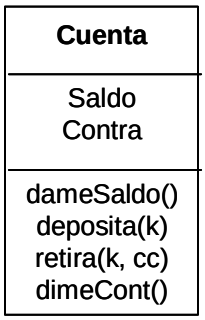


Figura 17. Clase Cuenta en UML

Brevemente descritos, los métodos tienen el siguiente comportamiento:

Cuenta()	El constructor, inicia saldo en cero
dimeSaldo()	Regresa el monto de saldo
dimeContra()	Regresa el valor de la contraseña (contra)
deposita(k)	Aumenta el saldo por una cantidad k
retira(k, cc)	Si la contraseña es igual a cc, entonces si el saldo es mayor o igual a k, retira dicha cantidad disminuyéndolo; regresa la cantidad retirada, cero si no alcanza y menos uno si la clave no corresponde.

En la Figura 18 se muestra el grafo de llamadas mejorado para esta clase.

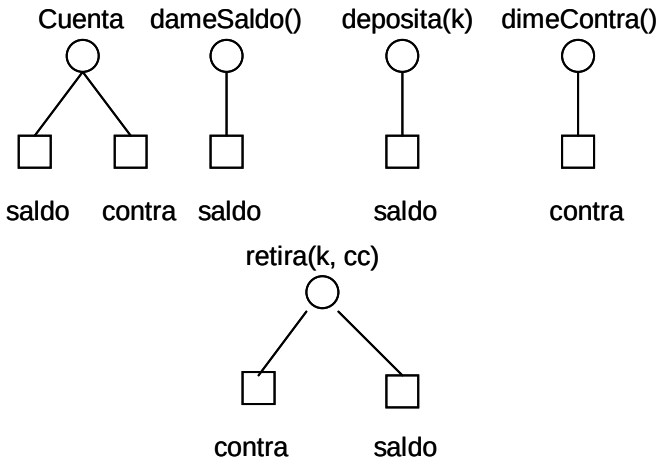


Figura 18. GLLM para la clase Cuenta

Ejemplo 2 Clase PoligonoRegular

En la Figura 19 se muestra la estructura de esta clase en UML y luego en la Figura 20 se muestra su GLLM. La descripción de los métodos es como sigue:

PoliginoRegular(a,l,n)	constructor; inicializa alto, lado y numlados
perimetro()	regresa el valor del perímetro, calculado como lado * numlados
area()	regresa el valor del área, calculada como (alto*lado/2)*numlados
areaCirc()	regresa el valor del área del círculo inscrito en el polígono,

como: π * alto * alto / 2

dimeAlto() regresa alto
dimeLado() regresa lado
dimeNum() regresa numlados

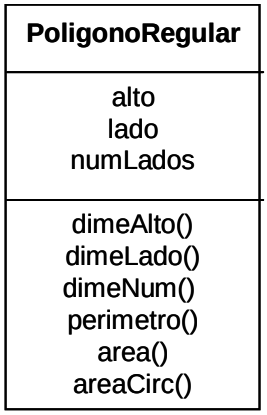


Figura 19. Clase PoligonoRegular en UML

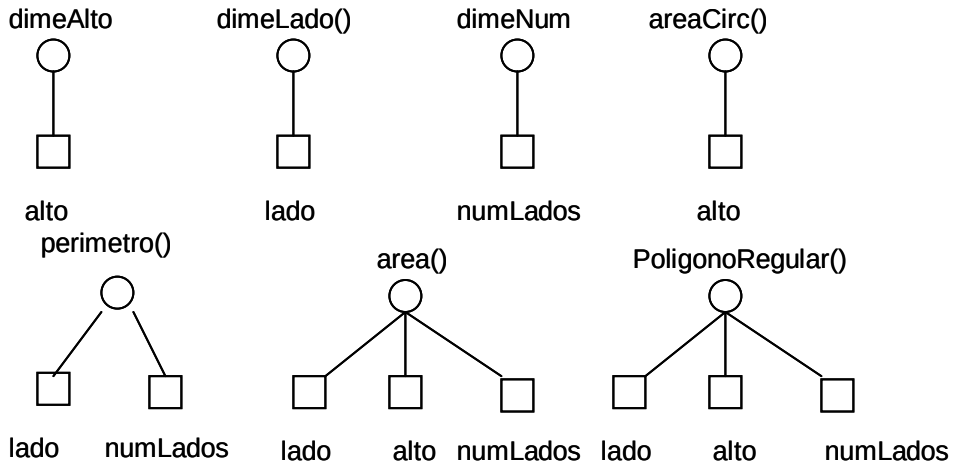


Figura 20. GLLM de la clase PoligonoRegular

Ejemplo 3 Clase CalcRacional

La clase CalcRacional representa una calculadora que opera sobre números racionales, es decir fracciones o quebrados. Así, cada número requiere un numerador y un denominador. Sólo se ilustran la suma y la multiplicación, para no complicar el ejemplo. Los métodos simplifica, maxcomdiv (máximo común divisor) y mincommult (mínimo común múltiplo) son auxiliares. En la Figura 21 se muestra la estructura de la clase y en la Figura 22 el GLLM correspondiente. Los métodos operan así:

CalcRacional()	Constructor, deja numerador y denominador indefinidos
carga(n,d)	Carga un número en forma de numerador y denominador, actualizando nume y deno
suma(n,d)	Suma el valor actualizando nume y deno; usa mincommult para calcular el nuevo denominador y simplifica para reducir la fracción

multiplica(n,d)	Multiplica el número almacenado por el nuevo; usa simplifica para reducir la fracción
mincommult(x,y)	Obtiene el mínimo común múltiplo de los números enteros x e y
maxcomdiv(x,y)	Obtiene el máximo común divisar de los números enteros x e y
simplifica()	Reduce la fracción a los menores valores posibles de numerador y denominador usando la función maxcomdiv; deja el resultado en nume y deno
dimeNume()	Regresa el numerador
dimeDeno()	Regresa el denominador
limpia()	Deja nume y deno en cero

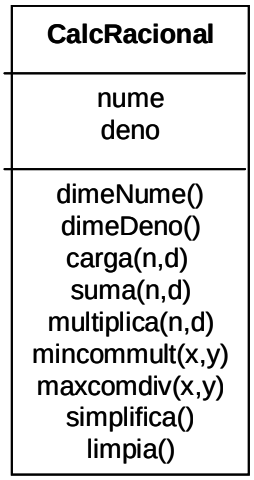


Figura 21. Clase CalcRacional en UML

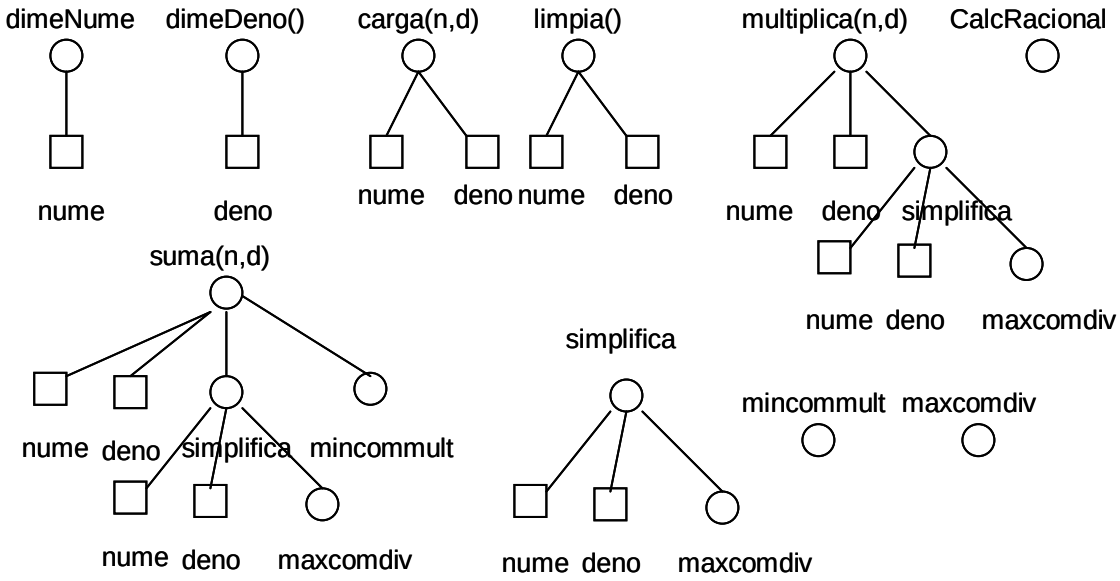


Figura 22. GLLM de la clase CalcRacional

7.2.2 Matriz de uso mínima (MUM)

La Matriz de Uso Mínima es una matriz que tiene una fila por cada atributo de una clase y una columna por cada método de la misma, incluyendo los constructores. La celda tendrá un contenido que depende de la relación entre el método j y el atributo i, como sigue:

MUM(i,j)

t	si el método j transforma o modifica el valor del atributo i
r	si el método j reporta o informa el valor del atributo i
o	si el método j utiliza el valor del atributo i en alguna operación, sin cambiarlo
Vacío	si no hay relación entre ellos

La MUM se puede construir directamente observando cada método y marcando el uso que hace de los atributos. Esta forma es útil para clases muy sencillas, con pocos atributos. Para clases más complejas y que incluyen invocación de métodos de la propia clase, es más conveniente construir primero el grafo GLLM.

7.2.3 Construcción de la MUM a partir del GLLM

Si se cuenta con el GLLM se construye la MUM como sigue:

Para cada árbol del grafo:
Para cada hoja atributo:
se marca la celda correspondiente a su fila y la columna del método que está en la raíz y junto se anota un número que indica el número de arcos entre la hoja y la raíz. Si el mismo atributo se conecta en más de un camino, se anotan todos.

Ejemplo de anotaciones: t1 (el método transforma al atributo y el camino es de un arco), o2 (el método usa el atributo y el camino tiene dos arcos); o1,o2 (existen dos usos del atributo, uno a un arco de distancia o directo, y otro a dos, es decir a través de una llamada a otro método).

Ejemplos completos.

Ejemplo 1 Clase Cuenta
A partir del grafo de la Figura 3, se construye la matriz de la Tabla 11, como sigue:

a) en el primer árbol los atributos saldo y contra son inicializados por el constructor Cuenta, en un solo arco, por lo que se anota t1 en cada uno.
b) en el segundo árbol, el valor de saldo es reportado por el método dameSaldo, por lo cual se anota r1.
c) en el tercero, el atributo saldo es afectado por el método deposita, anotándose t1.
d) en el cuarto, contra es reportado por el método dimeContra, anotándose r1.
e) en el quinto, saldo es modificado por el método retira, anotándose t1; contra es usado para comprobación, por lo que se marca o1.

Tabla 10. MUM de la clase Cuenta

Atributo	Cuenta	dameSaldo	deposita	dimeContra	retira
saldo	t1	r1	t1		t1
contra	t1			r1	o1

Ejemplo 2 Clase PoligonoRegular
En forma similar al ejemplo anterior, a partir del grafo de la Figura 5 se construye la matriz presentada en la Tabla 12:

Tabla 11. MUM de la clase PoligonoRegular

Atributo	Polígono Regular	dime Alto	dime Lado	dime Num	area Circ	perimetro	area
alto	t1	r1			o1		o1
lado	t1		r1			o1	o1
numLados	t1			r1		o1	o1

Ejemplo 3 Clase CalcRacional
A partir del grafo de la Figura 7 construimos la MUM que se muestra en la Tabla 13, para lo cual la mayoría de los árboles son similares a los de los ejemplos anteriores. Sin embargo, los árboles de los métodos suma y multiplica son más complejos. En el árbol de suma, el atributo nume es transformado por el método directamente (distancia 1) y también indirectamente por el método simplifica, con distancia 2; así pues habrá de anotarse t1,t2. Otro tanto ocurre con el atributo deno. La misma situación ocurre en el árbol del método multiplica.

Tabla 12. MUM de la clase CalcRacional

Atri-buto	Calc Racio nal	Dime Nume	Dime Deno	carga	Lim pia	suma	multi plica	min com mult	max com div	sim pli fica
nume		r1		t1	t1	t1,t2	t1,t2			t1
deno			r1	t1	t1	t1,t2	t1,t2			t1

En la Tabla 3 note que la primera columna, correspondiente al constructor, está vacía, ya que no carga ningún dato.

7.2.4 Minimización de la matriz

La matriz MUM indica ser mínima, pero la matriz que obtuvimos aún no lo es, al menos no siempre lo es. Cuando todos los caminos son de longitud uno, la matriz es mínima; en cambio, cuando existen caminos de mayor longitud, es posible que se puedan eliminar algunos elementos, haciendo más sencilla la matriz y reduciendo el número de pruebas necesarias.

Si existen dos árboles como en la Figura 23, con la matriz correspondiente en Tabla 14, tenemos la siguiente situación:

- el atributo a2 es utilizado por el método me2, directamente, por lo cual se anota o1 en su columna;
- también es usado por me1, a través de la llamada a me2, es decir en dos pasos, por lo que se anota o2 en su columna
- si comparamos las dos columnas de la fila de a2, veremos que sólo difieren en el número; esto significa que una de ellas es redundante.

En consecuencia, se deja únicamente la que corresponde a me2. La otra es una llamada repetida, ya que me1 no accede directamente a a2.

En cambio, para a1 las dos entradas son o1, o2 y o1; como se ve, son diferentes y deben ser probadas por separado. En este caso, me1 accede a a1 directamente y también a través de me2.

De aquí se puede establecer como regla:

- Para la fila de un atributo, si dos columnas tienen el mismo contenido en cuanto a tipo (o,r, t) y sólo difieren en el número, elimine aquella de número mayor.

Finalmente la matriz quedaría como en Tabla 15.

Para los ejemplos presentados no existe ninguna simplificación, por lo que ya son matrices mínimas.

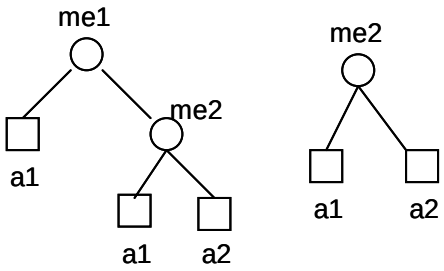


Figura 23. GLLM de ejemplo

Tabla 13. MUM para el ejemplo de Figura 8

Atributo	me1	me2
a1	o1,o2	o1
a2	o2	o1

Tabla 14. MUM minimizada correspondiente al ejemplo de Figura 23

Atributo	me1	me2
a1	o1,o2	o1
a2		o1

7.2.5 Revisión de la clase usando la MUM

La matriz MUM, además de su función en las pruebas, nos brinda una herramienta para revisar el diseño de la clase que representa. Algunos elementos que podemos extraer de ella son los mostrados en la Tabla 16.

Tabla 15. Revisión del diseño de una clase a partir de su MUM

Situación	Qué representa
Fila sin entradas	Muy posiblemente se viola el principio de ocultamiento de la información, ya que no se requiere un método para acceder al atributo. Otra posibilidad es que el atributo está sobrando. En todo caso conviene revisar el diseño.
Columna sin entradas	El método correspondiente sólo es un auxiliar de cálculo o bien no pertenece a la clase y debería acomodarse en otra parte. Conviene revisar diseño.
Las filas y columnas pueden reordenarse de modo de formar dos submatrices disjuntas	La clase en realidad contiene dos conceptos mezclados, uno con un grupo de atributos y métodos y otro separado, sin conexión. Debe rediseñarse separando en dos. Dejarla así es un error de Cohesión
Fila con más de 5 entradas de transformadores	Posiblemente se descompuso excesivamente alguno de los métodos. Revisar el diseño para balancear el número de métodos que operan sobre el atributo y la longitud de los mismos

A partir de esta información se puede diseñar mejores clases, que no caigan en los problemas mencionados. Los ejemplos de arriba no sufren de tales problemas.

7.3 Secuencias de métodos para prueba

Una vez que se cuenta con la matriz mínima de uso, se procederá a generar una serie de secuencias de prueba. Para esto se procede como sigue:

a) Se categoriza cada método como sigue: constructores, reporteros, transformadores y otros. La categoría transformadores incluye los métodos que realizan modificaciones sobre los atributos y también los que operan con sus valores, es decir los que se marcaron con “t” y con “o”. En otros se dejan los que no caen en ningún caso anterior, que se supone son excepciones, como métodos que reportan errores y mandan avisos auxiliares, sin relación con el estado del objeto.
b) Se prueba a los métodos reporteros. En este paso sólo interesa que reporten adecuadamente el valor del atributo que les corresponde. Si existe un atributo sin reportero, se tratará más abajo. Pueden generarse secuencias de reporteros al azar.
c) Se prueban los diversos constructores, con diferentes valores y se verifican éstos usando los reporteros. Debe cuidarse que los reporteros cubran todas las rebanadas.
d) Se prueban los transformadores como sigue:
    a. Se elige un atributo y los transformadores que corresponden a su rebanada y el reportero del atributo.
    b. Se forman las diversas permutaciones de los diferentes transformadores, colocando siempre el reportero al final
    c. Cada una de las secuencias anteriores será una secuencia de prueba; para cada una de ellas deberán considerarse varios casos de prueba,

correspondientes a los diferentes tipos de valores que pueden tomar sus parámetros. Ésto se trata en la Sección 7.4.

e) Una vez probados los métodos anteriores, sólo quedan los métodos que no caen en ninguna de las categorías anteriores, pero que resultan más sencillos al no afectar ningún atributo.

Ejemplos

Clase Cuenta.

categoría	
reporteros	dameSaldo() y dimeContra()
constructores	Cuenta(k, c)
Rebanadas de transformadores	Rebanada de saldo: {deposita(k), retira(k,cc)} Rebanada de contra: {retira(k,cc)}
Secuencias de transformadores	deposita(k), retira(k,cc),dimeSaldo() retira(k,cc), deposita(k), dimeSaldo() retira(k,cc), dimeContra()
Otros	No hay

Clase PoligonoRegular.

categoría	
reporteros	dimeAlto(), dimeLado(), dimeNum()
constructores	PoligonoRegular(a, l, n)
Rebanadas de transformadores	Rebanada de alto: {areaCirc(), area()} Rebanada de lado: {perímetro(), area()} Rebanada de numLados: {perímetro(), area()}
Secuencias de transformadores	areaCirc(), area(), dimeAlto() area(), areaCirc(), dimeAlto() perimetro(), area(), dimeLado() area(), perimetro(), dimeLado() perimetro(), area(), dimeNum() area(), perimetro(), dimeNum()
Otros	No hay

Clase CalcRacional.
{Como ambas rebanadas son iguales y siempre se opera sobre los dos, usamos un solo juego de permutaciones con los dos reporteros al final.}

categoría	
reporteros	dimeNume() y dimeDeno()
constructores	CalcRacional()
Rebanadas de transformadores	Rebanada de nume: {carga(n,d), limpia(), suma(n,d), mult(n,d), simplifica()} Rebanada de deno: {carga(n,d), limpia(), suma(n,d), mult(n,d), simplifica()}
Secuencias de transformadores(¡)	carga(n,d), limpia(), suma(n,d), mult(n,d), simplifica(), dimeNume(), dimeDeno() carga(n,d), limpia(), suma(n,d), simplifica(), mult(n,d), dimeNume(), dimeDeno() carga(n,d), limpia(), simplifica(), suma(n,d), mult(n,d), dimeNume(), dimeDeno() carga(n,d), simplifica(), limpia(), suma(n,d), mult(n,d), dimeNume(), dimeDeno() simplifica(), carga(n,d), limpia(), suma(n,d), mult(n,d), dimeNume(), dimeDeno() carga(n,d), limpia(), mult(n,d), suma(n,d), simplifica(), dimeNume(), dimeDeno() carga(n,d), mult(n,d), limpia(), suma(n,d), simplifica(), dimeNume(), dimeDeno() mult(n,d), carga(n,d), limpia(), suma(n,d), simplifica(), dimeNume(), dimeDeno() carga(n,d), suma(n,d), limpia(), mult(n,d), simplifica(), dimeNume(), dimeDeno() suma(n,d), carga(n,d), limpia(),mult(n,d), simplifica(), dimeNume(), dimeDeno() limpia(), carga(n,d), suma(n,d), mult(n,d), simplifica(), dimeNume(), dimeDeno()....{hasta completar 120 secuencias, ya que corresponden a las permutaciones de los métodos que las forman}
Otros	No hay

El procedimiento anterior suena largo, pero sin complicaciones. Sin embargo, la programación orientada a objetos introduce algunos problemas que afectan a la prueba. Por el momento se tratarán dos:

1. Carencia de reportero. Un atributo puede carecer de reportero, lo cual acarrea problemas para aplicar las pruebas como se ha descrito. Si el atributo es privado, será imposible probar adecuadamente la clase, a menos que se modifique el código. Si el atributo es protegido (friend), se puede generar una clase derivada de la clase que se está probando, agregando los reporteros necesarios, lo cual no altera la clase original.
2. Conexiones con otras clases. Una clase no existe de manera aislada; generalmente es parte de un conjunto que realizan una cierta funcionalidad deseada por un usuario. Esto hace que existan un gran número de interacciones en la forma de invocaciones a métodos de diversas clases. Sin embargo, en la prueba de unidad se busca probar

unidades aisladas, sin interacción. Así pues, se requiere sustituir las clases invocadas y también las que invocan a la clase que se prueba. Esto se hace con clases de utilería, que contienen métodos auxiliares, pero sin código específico del sistema. Los métodos de clase invocadas se llaman cabos (stubs) y su única función es regresar un resultado (y valor) que resulte típico. Las clases que llaman a la clase bajo prueba se llaman Manejadoras (drivers) y se concretan a invocarla. En la sección 7.5 se describen auxiliares automáticos para estos trabajos.

Para garantizar que no ocurran problemas como el de la carencia de reporteros, existen una serie de recomendaciones de diseño que se conocen generalmente como diseño facilitador de pruebas ("design for testability"), entre las cuales podemos citar:

Siempre que escriba una clase, haga privados todos los atributos y escriba un reportero para cada uno.

7.4 Generación de casos de prueba

Una vez que se tiene una secuencia de pruebas, con la forma que sigue, se debe ejecutar.

Transformador 1 – Transformador 2 - … - Transformador n – Reportero

En este momento surge una complicación: un método transformador usualmente tiene parámetros y por tanto debe decidirse qué valor se dará a éstos. En casos simples, cualquier valor que se de al parámetro origina el mismo tipo de respuesta; por ejemplo, un método ***deposita(k)*** esperamos que agregue el valor de k al de saldo, no importa cuál sea el valor de k. Sin embargo, en otros casos no es así; por ejemplo, un método que recibe un valor entero positivo y regresa el logaritmo de ese número, en caso de recibir un número negativo deberá regresar un error; así pues hay dos casos.

Para asegurarse de haber probado adecuadamente cada transformador, deben cubrirse todos los casos. Para eso existen varios métodos, estudiados en otra parte. Por el momento, recomendamos usar el de dominios (particiones) y valores a la frontera. Los diferentes casos se reemplazarán donde aparezca el transformador en cada secuencia de prueba. Note que ésto aumenta considerablemente los casos de prueba que deben ejecutarse.

Como el proceso es muy laborioso y bastante aburrido, se debe preferir el uso de herramientas automáticas, como se describe en la sección siguiente.

Aplicación a los ejemplos.

Clase Cuenta:

Aquí los dos métodos son indiferentes al número que se use, así que bastará emplear un valor típico y, para más seguridad, un valor negativo.

Clase PoligonoRegular:

Aquí no hay problema pues no hay parámetros en los métodos transformadores y por tanto bastará un caso de prueba para cada uno.

Clase CalcRacional:

En carga, suma y mult se pueden presentar dos casos: el de un número racional normal (una fracción x) y un número con denominador cero, así que las 120 secuencias se convierten en muchas más. Es posible que se puedan reducir algunas, pues el caso del denominador cero es una excepción.

7.5 Herramientas auxiliares

De los elementos presentados anteriormente, se tiene que para probar una clase se requiere:

a) clases de utilería con cabos para los métodos que puedan ser llamados, capaces de regresar un valor típico
b) una clase manejadora que dirija la prueba

Además se requiere generar las permutaciones de secuencias de métodos, lo cual puede hacerlo una un método dentro de la clase manejadora.

De todo ésto lo más difícil es la clase manejadora, quien deberá llevar control de secuencias, reportar fallas, etc.

En el mercado existen diversas herramientas que automatizan parcialmente el proceso y también existen herramientas de software libre de muy buena calidad. Entre éstas se tiene a JUnit, creada para programas escritos en Java y que se puede bajar libremente; además se encuentra integrada en la plataforma Eclipse, también de software libre, así como en NetBeans, que se puede bajar gratis, aunque sea propiedad de SUN.

A partir de JUnit se han desarrollado herramientas equivalentes para otros lenguajes, como DUnit, que se adapta a la plataforma Delphi, CUnit para probar programas escritos en C++, etc.

Estas herramientas proporcionan clases básicas para probar clases, permitiendo preparar la prueba (crear objetos, preparar el estado, etc.), aplicar la prueba, reportar resultados y limpiar estado al final

Si no se cuenta con una de estas herramientas y no se desea obtenerla, deberá desarrollarse software auxiliar que la reemplace. Los autores Bashir y Goel [BAS 2000] recomiendan crear una clase auxiliar que hereda de la clase bajo prueba y realiza varias funciones auxiliares. Para más detalles, ver su obra.

7.6 Prueba de clases derivadas (subclases)

Un concepto de gran importancia en la programación orientada a objetos es la herencia. Las clases forman árboles de jerarquía donde unas clases son más generales y otras más particulares. Las clases particulares heredan los atributos y comportamiento generales de una clase base y lo extienden o modifican para darle características diferentes.

Para probar una clase derivada de otra que ya se probó, podría hacerse como si se careciera de información, aplicando el método como se vio en las secciones 7.2 a 7.4. Sin embargo, una de las ventajas de la herencia es el no repetir cosas que ya están hechas. Así pues, se busca aprovechar el trabajo anterior sin repetirlo.

En una clase derivada se presentan cuatro posibles escenarios:

i) métodos nuevos[5] o redefinidos que afectan atributos de la clase base
ii) métodos de la clase base que afectan atributos de la clase base
iii) métodos de la clase base que afectan atributos nuevos
iv) métodos nuevos o redefinidos que afectan atributos nuevos

De los cuatro casos sólo el segundo se puede considerar libre de nuevas pruebas. Los otros dependen de la existencia de algún cambio que corresponda a ellos. El más improbable, pero no imposible, es el caso tres, ya que los métodos preexistentes no pueden actuar directamente sobre datos nuevos, a menos que exista una llamada indirecta a través de un método redefinido.

El procedimiento de prueba para una clase derivada utiliza las mismas ideas de las rebanadas, con algunas modificaciones. Puede resumirse así:

1. construir el Grafo de Llamadas Mejorado (GLLM)
2. construir la matriz MUM modificada
3. probar las rebanadas correspondientes a atributos nuevos
4. probar las rebanadas correspondientes a atributos originales que hayan sufrido cambios

Para ejemplificar utilizaremos una clase derivada de la clase Cuenta definida anteriormente, como se muestra en la Figura 24 y que se define abajo.

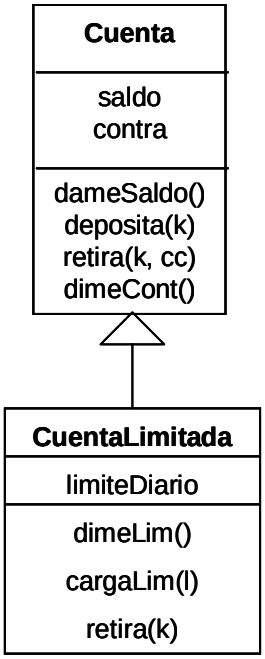


Figura 24. Clase CuentaLimitada

La clase CuentaLimitada es en todo similar a Cuenta, excepto que tiene un límite de retiro por día. Cada día se carga el limiteDiario usando el método cargaLim(l). Durante el día, si se retira una cantidad menor o igual al límite y aceptable por el saldo, se realiza y se disminuye el límite. Si se pretende retirar más del límite, se rechaza. Así pues, se debe

[5] Se entiende como nuevo método y nuevo atributo aquellos que son propios de una clase hijo o subclase, es decir, que heredan de otra previamente definida y probada.

modificar el método retira y supondremos que se reescribió completamente. También se agregará un método dimeLim() que reporta el límite en ese momento.

7.6.1 Grafo de llamadas mejorado

En forma similar al método básico, se construye el GLLM, con los siguientes cambios: cuando se invoca a un atributo de la clase original, se marcará con un cuadrado doble y cuando se invoque a un método de la clase base, se marcará con un círculo doble. En la Figura 25 se muestra el GLLM para CuentaLimitada.

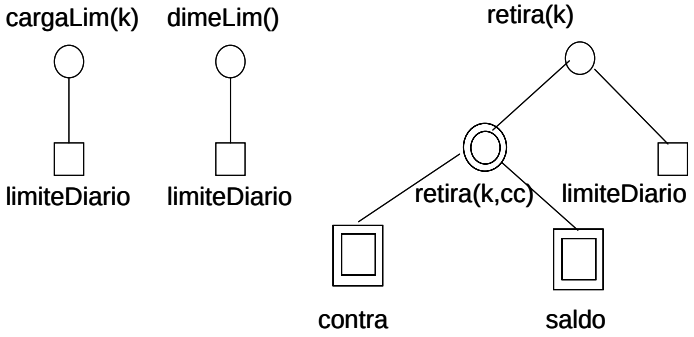


Figura 25. GLLM para la clase CuentaLimitada

Puede observarse en la Figura 10 que el método retira (nuevo) hace una llamada al método antiguo (con doble círculo).

7.6.2 Construir MUM modificada

El siguiente paso es construir la MUM, para lo cual se formará en cuatro cuadrantes, como se muestra en la Tabla 17, indicados con números romanos. El cuadrante II es idéntico a la MUM de la clase base y el cuadrante IV corresponde a las interacciones entre métodos y atributos nuevos. Los otros dos corresponden a interacciones entre métodos nuevos y atributos antiguos y viceversa.

Tabla 16. MUM modificada

	Métodos de clase base	Métodos nuevos
Atributos de clase base	**II**	**I**
Atributos nuevos	**III**	**IV**

El llenado de la MUM debe realizarse como sigue:

a) primero se llena el cuadrante II con los datos de la clase base;
b) después se llena el cuadrante IV, que es totalmente nuevo
c) después se llena el cuadrante III que probablemente esté vacío
d) finalmente se llena el cuadrante I

Para el ejemplo de la clase CuentaLimitada se obtendrá una MUM como la de la Tabla 18. Las líneas obscuras sirven únicamente para indicar los cuatro cuadrantes y en aplicaciones prácticas no se usan.

Tabla 17. MUM de la clase CuentaLimitada

	CuentaLimitada	**dameSaldo**	**deposita**	**dimeContra**	**retira**	**dimeLim**	**cargaLim**	**retira**
saldo	t1	r1	t1		t1			t2
contra	t1			r1	o1			o2
limite Diario						r1	t1	o1

En la Tabla 18 se puede observar que el cuadrante III quedó vacío, lo cual es el caso más común.

7.6.3 Generación de rebanadas y secuencias

Teniendo la MUM de la clase derivada, se procede a generar las rebanadas de los atributos agregados y a partir de ellos las secuencias de prueba. Para esto se usan los cuadrantes inferiores (III y IV).

Una vez terminado el paso anterior, se procede a analizar las rebanadas de la clase base y observar si hubo cambios. Si los hay, entonces se deben generar las secuencias de prueba igual que antes. Para las rebanadas que no sufrieron cambio no es necesario hacer nada. En esta etapa se usan los cuadrantes superiores (I y II).

Para la clase de ejemplo, se tiene:

Con los sectores III y IV:
Para el atributo limiteDiario sólo hay un método transformador, cargaLim(), así que bastará probarlo seguido del reportero dimeLim().

Con los sectores I y II:
Sólo hay cambios de transformadores en el atributo saldo y sólo es un nuevo método, que reemplaza al anterior, por lo cual podemos reemplazarlo en las secuencias que ya se habían generado en la clase original.

Parte II. Prácticas

El objetivo de esta parte es presentar las prácticas que conforman este manual y que están divididas en cinco grupos: prácticas de herramientas, prácticas de modelado, prácticas de prueba de software y prácticas de métricas.

Práctica 1. Creación de la Red Semántica Natural de un Sistema de Software

I. Objetivo: El alumno entenderá la forma en que se crea una RSN utilizando una hoja de cálculo.

II. Equipo necesario:

- Computadora con MS-Windows preferentemente conectada a Internet.

III. Material de apoyo:

- Una hoja de cálculo como MS-Excel
- Tarjetas con la información que brindó el usuario
- Hojas de cálculo de RSN de sistemas anteriores
- Libro de la metodología Áncora
- Diapositivas sobre Áncora

IV. Procedimiento:

1. Abrir la hoja de cálculo
2. Introducir las tarjetas, en la hoja que le corresponda, una por columna
3. Reunir todos los datos de todos los entrevistados
4. Calcular Pesos Semánticos
5. Realizar gráficas
6. Escoger las definitorias resultantes por tipo

V. Resultados esperados:

Cuatro listas de definitorias escogidas y sus gráficas correspondientes sobre la hoja de Excel

Práctica 2. Creación de la Encuesta de Actitud

I. Objetivo: El alumno será capaz de construir encuestas que mida la actitud de los usuarios potenciales ante la eventualidad de hacerles un nuevo software.

II. Equipo necesario:

- Computadora con MS-Windows preferentemente conectada a Internet.

III. Material de apoyo:

- Una procesador de palabras como MS-Word
- Una hoja de cálculo como MS-Excel
- RSN: sensaciones y actividades en la hoja Excel
- Encuestas realizadas de sistemas anteriores
- Libro de la metodología Áncora o diapositivas

IV. Procedimiento:

1. Abrir la hoja de cálculo con las RSN
2. Abrir MS-Word y crear una tabla con 6 columnas y el siguiente número de filas:
    Para todas las sensaciones negativas poner:
        Una fila por cada sensación
        Debajo de cada sensación una fila por cada actividad
    Para todas las sensaciones positivas (contraparte de la lista anterior) poner:
        Una fila por cada sensación
        Debajo de cada sensación una fila por cada actividad
3. Introducir sensaciones
4. Introducir actividades debajo de cada sensación

V. Resultados esperados:

Una encuesta que se aplicará a cada usuario potencial del sistema o al menos a una muestra significativa que incluya representantes de los diversos grupos de interés identificados por el cliente.

Práctica 3. Captura de la Encuesta de Actitud

I. Objetivo: El alumno será capaz de construir una hoja de cálculo con los enunciados de la encuesta de actitud y llenarla con los resultados de ésta que fue aplicada previamente a los usuarios potenciales ante la eventualidad de hacerles un nuevo software.

II. Equipo necesario:

- Computadora con MS-Windows preferentemente conectada a Internet.

III. Material de apoyo:

- Una procesador de palabras como MS-Word
- Una hoja de cálculo como MS-Excel
- Encuestas de actitud llenadas por los usuarios
- Libro de la metodología Áncora o diapositivas

IV. Procedimiento:

1. Abrir una hoja de cálculo
2. Introducir en la primera columna los enunciados
3. A partir de la segunda columna introducir a cada usuario; ejemplo: si son diez usuarios habrá diez columnas.
4. Junto a cada enunciado poner la calificación que el usuario el dio de la siguiente manera:
   - Afirmaciones a favor de que se realice el sistema de 5 a 1.
   - Afirmaciones en contra de la realización del sistema de 1 a 5.
5. Al finalizar la captura haga:
   - Para cada columna sumar las respuestas y dividir entre el número de preguntas, es decir su promedio.
   - Para cada renglón sacar promedio
   - Sacar el promedio de todos los usuarios
6. Analizar resultados según se marca en el libro e incluirlo en un documento de Word

V. Resultados esperados:

Una encuesta de actitud capturada y analizada que se indicará los problemas más graves del sistema actual y la actitud general de los usuarios

Práctica 4. Aplicación y captura de encuestas de perfilado de usuario y contexto de uso

I. Objetivo: El alumno será capaz de construir: 1) perfilado de usuarios potenciales del software desde el punto de vista de usabilidad y 2) establecimiento de su ambiente de trabajo.

II. Equipo necesario:

- Computadora con MS-Windows preferentemente conectada a Internet.

III. Material de apoyo:

- Una hoja de cálculo como MS-Excel
- Cuestionario de perfilado de usuario
- Cuestionario sobre el contexto empresarial y computacional
- Diapositivas sobre artefactos de Usabilidad con Áncora (http://www.uv.mx/personal/asumano/material-de-cursos/).

IV. Procedimiento:

1. Abrir la hoja de cálculo con los cuestionarios de perfilado y contexto (CapturaEncuestaUsabilidad.xls).
2. Capturar el cuestionario de perfilado, una columna por usuario.
    - Escriba un uno en la casilla correspondiente de las preguntas: edad, 1 a 7 y 10 a 11
    - Teclee de 1 a 5 según lo señalado en la pregunta 8.
    - Teclee de 5 a 1 según lo que se haya tachado en la pregunta 9.
3. Capturar el cuestionario de contexto, una columna por espacio físico diferente.
    - Teclee de 1 a 3 según lo señalado en la pregunta 1 a 5.
    - Escriba un uno en la casilla correspondiente de las preguntas: 6, 7 y de la 11 a la 15.
4. Observar resultados: frecuencias en caso de haber captura con unos y promedios en caso de haber capturado valores y anotar conclusiones en el documento fundacional.

V. Resultados esperados:

Documento fundacional, mismo que permitirá realizar los otros artefactos de usabilidad y con ello establecer una mejor Interfaz de Usuario.

Práctica 5. Construcción del Guión de la Situación Actual

I. Objetivo: Basado en los resultados de: RSN, encuesta de actitud y usabilidad, el alumno será capaz de construir el Guión-Áncora que refleje la situación actual del sistema.

II. Equipo necesario:

- Computadora con MS-Windows preferentemente conectada a Internet.

III. Material de apoyo:

- Un procesador de palabras como MS-Word
- RSN: nombre, elementos, sensaciones y actividades
- Análisis de la encuesta de actitud
- Análisis de las encuestas de usabilidad
- Libro de la metodología Áncora o diapositivas

IV. Procedimiento:

Con base en los documentos de obtención de requerimientos y de usabilidad, crear un Guión que refleje la manera actual en que trabaja el usuario en el sistema de estudio, fíjese en los siguientes puntos:

1. De la RSN de nombres escoja el de mayor votación y ese será el nombre del guión
2. Vea si hay alguna agrupación lógica de las tareas que le permita separa el sistema en varias pistas, de ser así, hágalo. Recuerde que un guión no debe ser mayo a una hoja tamaño carta con letra Arial de 12 puntos o equivalente.
3. En cada pista anote los papeles identificados en las tarjetas tipo 4 usadas para construir la RSN de Actividades.
4. Haga la lista de utensilios basándose en la RSN de Elementos
5. Escriba las escenas de cada pista basándose el la RSN de actividades
6. Subraye las quintetas que, según la encuesta de actitud, resultan con un promedio mayor o igual a 3. Nota: tales quintetas serán en donde se debe centrar la propuesta computacional.

V. Resultados esperados:

El Guión de la Situación Actual en donde las quintetas más problemáticas serán subrayadas.

Práctica 6. Construcción del Guión de la Propuesta Computacional

I. Objetivo: El alumno construirá una propuesta de automatización del nuevo software utilizando la construcción del Guión-Áncora para iniciarla.

II. Equipo necesario:

- Computadora con MS-Windows preferentemente conectada a Internet.

III. Material de apoyo:

- Un procesador de palabras como MS-Word
- El software ÁncoraSoft
- Guión de la situación Actual con la lista de principales problemas a resolver mediante el uso de un software
- Libreta con el guión de la propuesta computacional esbozado
- Libro de la metodología Áncora
- Diapositivas de la página www.uv.mx/personal/asumano.
- De este documento: Parte I. Capítulo 1, Sección 4

IV. Procedimiento:

1. Abrir los documentos asociados
2. Tener a la mano la libreta
3. Si ya tiene clave de trabajo, abrir el programa Usar_ÁncoraSoft y empiece a crear su guión
4. Si no tiene clave utilizar el Administrar-AncoraSoft para crear su clave.

NOTA: Recuerde que en la propuesta computacional deben ir ***solamente las escenas que se automatizarán***, si lo considera necesario puede agregar comentarios en cada quinteta.

V. Resultados esperados:
El Guión-Áncora de la propuesta computacional cargada en AncoraSoft.

Práctica 7. Construcción de Artefactos a partir del Guión de la propuesta computacional

I. Objetivo: El alumno construirá una serie de artefactos que muestran los diversos elementos involucrados en el establecimiento de los requerimientos del software y que se derivan y asocian al Guión-Áncora de la Propuesta Computacional. Todos estos artefactos servirán como conexión con las etapas subsecuentes del desarrollo de software y muestran diversos puntos de vista de la propuesta computacional del problema a resolver.

II. Equipo necesario:

- Computadora con MS-Windows preferentemente conectada a Internet.

III. Material de apoyo:

- Un procesador de palabras como MS-Word
- El software:
  - ÁncoraSoft para revisar el guión
  - Un IDE como Eclipse o NetBeans, para definir las ventanas
  - Un SMBD como DBDesigner, para dibujar en modelo E-R
- Libro de la metodología Áncora
- Diapositivas de la página www.uv.mx/personal/asumano.
- De este documento: Parte I. Capítulo 1, Sección 4

IV. Procedimiento:

Para generar los artefactos asociados al guión de la propuesta computacional, haga lo que se pide:

1. Prototipo rápido
   a. Abra un archivo Word con nombre “Manual Preliminar del <nombre del software>”.
   b. Incluya la lista de Pistas, escenas y quintetas del guión de la propuesta computacional.
   c. Para cada quinteta agregue las ventanas de captura, reporte o resultado que están asociados como sigue:
      i. Abra el IDE
      ii. Cree la ventana
      iii. Copie la imagen y pegue en el texto de Word
   d. Para cada ventana incluya la descripción de la lista de campos que incluye
   e. Para las situaciones anómalas empiece incluyendo los cursos de acción alternos, fallidos e indeseables de la bitácora de desarrollo (siguiente inciso).
2. Bitácora de Desarrollo
   a. Abrir Usar_AncoraSoft>Herramientas>PrepararBitacora
   b. Crear Bitácora, si no existe
   c. Abrir Pista para escoger la pista que se trabajará
   d. Dar clic en Comprobación>Forma de Comprobar
   e. Teclear todos los cursos de acción de cada quinteta y para cada curso:
      i. Teclear lo que se pide y al terminar el curso de clic en el icono con un disco para que aparezca en la ventana de la quinteta.
   f. Al terminar los cuatro cursos de cada quinteta dar clic en el icono de doble disco.

g. Para navegar entre las quintetas de una pista use los iconos de punta de flecha.
h. Al terminar de teclear los curso de acción de cada quinteta del guión, o antes si así lo desea, puede generar el reporte en formato de MS-Word.

3. Modelo Entidad Relación.
   a. Abrir DBdesigner
   b. Se debe considerar cada utensilio y cada papel como entidad, aunque algunos utensilios pueden ser atributos de otros utensilios o papeles).
   c. Relacione las Entidades con las acciones de las quintetas donde se usan. No todas las relaciones deben incluirse, escoja las más adecuadas.
4. Calcule los Puntos de Función usando el archivo Excel que se le proporcionará. Recuerde que este material lo revisó y empleo en el curso de Ingeniería de Software I.

V. Resultados esperados:
El Guión-Áncora de la propuesta computacional corregida y sus artefactos asociados: Bitácora de Desarrollo, Manual Preliminar del Usuario, Modelo Entidad – Relación y Cálculo de Puntos de Función.

Práctica 8. Planeación de pruebas de Sistema

I. Objetivo: El alumno construirá una serie de Casos de Prueba que le permitirán: establecer la forma en que se probará el software al terminar su implementación y refinar los requerimientos sobre todo aquellos detalles que no había considerado al exponer su propuesta de solución.

II. Equipo necesario:

- Computadora con MS-Windows preferentemente conectada a Internet.

III. Material de apoyo:

- Un procesador de palabras como MS-Word
- El software:
  - ÁncoraSoft para revisar la bitácora de desarrollo
- Notas sobre Prueba de sistema y Particiones, disponibles en: http://www.uv.mx/personal/jfernandez/

IV. Procedimiento:

Para generar los casos de prueba de aceptación del sistema asociados al guión de la propuesta computacional, haga lo que se pide:

1. Abra su bitácora de desarrollo empleando AncoraSoft
2. Cree un archivo MS-Word
3. Agregue el título Casos de prueba del Sistema de <nombre de sus sistema>
4. Agregue una Tabla de cinco columnas y (NMCP + 1) filas, como sigue

Núm. Progresivo	Quinteta y curso de acción	Condiciones de entrada	Datos de Entrada	Salida Esperada	Condiciones de salida

5. Llene cada fila con el caso de prueba que le corresponda.

V. Resultados esperados:

Un conjunto de casos de prueba cuyo número mínimo (NMCP) será:

$$NMCP = NumQuintetas * 4CA$$

Donde:

NumQuintetas = Número de Quintetas de todas las pistas que conforman el guión de la propuesta computacional.

4CA = Número 4 (cuatro) que corresponde a los cuatro cursos de acción de la bitácora de desarrollo por cada pista y el dos es porque debe haber dos casos de prueba por cada curso de acción.

NOTA: Habrá quintetas que no valga la pena probar por separado y deben reunirse en una sola. Esto le dará la pauta para eliminar aquellas quintetas innecesarias y poner su aclaración en un diálogo.

Práctica 9. Traducción de los Guiones Áncora a los modelos de: casos de uso y robustez

I. Objetivo: El alumno construirá una serie de artefactos que muestran los diversos elementos involucrados en el establecimiento de los requerimientos del software y que se derivan y asocian al Guión-Áncora de la Propuesta Computacional. Todos estos artefactos servirán como conexión con las etapas subsecuentes del desarrollo de software.

II. Equipo necesario:

- Computadora con MS-Windows preferentemente conectada a Internet.

III. Material de apoyo:

- Una procesador de palabras como MS-Word
- El software ÁncoraSoft
- El software StarUML
- Guión de la Propuesta Computacional cargado en ÁncoraSoft
- Libro de la metodología Áncora o diapositivas de la página www.uv.mx/personal/asumano.
- De este documento: Parte I. Capítulo 1, Sección 6

IV. Procedimiento:

1. Abrir el proyecto
2. Abrir el Guión de la Propuesta Computacional con AncoraSoft>Herramientas>SigaUML
3. Generar un archivo .xmi por cada pista.
    a. Abrir el programa Utilizacion_ÁncoraSoft
    b. Abrir proyecto
    c. Escoger SIGAUML del menú herramientas
    d. Dar nombre del guión
    e. Ordenar la generar de artefactos UML.
    f. Después de eso usted podrá ver los diagramas de casos de uso y los de robustez dando clic en su correspondiente de la Ventana de Artefactos UML
    g. Usando el icono de un documento, puede generar el archivo .xmi que le servirá para conectarse con alguna herramienta CASE (como StarUML) y proseguir su análisis.
4. Importar archivo xmi a StarUML
    a. Abra StarUML
    b. Escoja la versión de Modelo StarUML
    c. De clic en siguiente sin modificar parámetros
    d. Escoje el menú File>Importa>XMI
    e. Indique que desea construir los modelos de análisis
5. Utilizando el Model Explorer (si no lo ve vaya a View y de clic en Select Model Explorer) que se encuentra en la parte derecha superior de la ventana de StarUML, abra el Main del modelo de casos de uso y dibuje los paquetes (uno por cada pista) y asócielos con la relación “usar”.
6. Desde el Analysis Model arrastre con el Mouse el diagrama de casos de uso correspondiente a cada paquete. Fíjese que coincida con el diagrama que SigaUML dibujó y haga los ajustes estéticos que le parezcan..

7. Abra el Analysis Model y para cada caso de uso (escena) corrija el diagrama para que se asemejen visualmente a los diagramas de robustez de ICONIX como sigue:
    a. Dar clic en la clase, escoger su estereotipo (ver ventana de propiedades), dar clic Stereotype, clic en botón y elegir cuál estereotipo (boundary, entity o control).
    b. Para ver el icono: dar clic sobre la clase, seguido de botón derecho, Format, Stereotype Display, por último escoger Icon.
8. Dibuje los cursos, alternos (fallidos, indeseables, etc).

V. Resultados esperados:
El modelo de casos de uso y los diagramas de robustez de cada caso de uso.

Práctica 10. Cálculo de Puntos de Casos de Uso

I. Objetivo: El alumno calculará la métrica de Puntos de Casos de Uso basándose en algunos de los artefactos que se elaboraron para el sistema de software que se está modelando en el semestre. Con la métrica obtenida se podrá dar cuenta del tiempo que le llevaría desarrollar el software, según la propuesta de Clemmons, Roy K. (2006).

II. Equipo necesario:

- Computadora con MS-Windows preferentemente conectada a Internet.

III. Material de apoyo:

- Una hoja de cálculo como MS-Office
- El ejemplo que les proporcione el profesor cargado en Excel.
- Los artefactos creados necesarios para el cálculo: diagramas de robustez de cada caso de uso, restricciones no funcionales, conocimiento de las habilidades de su equipo de trabajo.
- Diapositivas sobre Puntos de Casos de Uso que están en la página www.uv.mx/personal/asumano.
- Capítulo 2 de este documento.

IV. Procedimiento:

1. Abrir los documentos asociados
2. Para el cálculo de la complejidad de cada caso de uso:
    a. Cuente las clases que aparecen en su diagrama de robustez de ese caso de uso, ese será el número de clases.
    b. Cuente las clases de control que tenga el diagrama de robustez, ese será el número de transacciones.
    c. Consulte la tabla dada en las diapositivas.
3. Para el cálculo de la complejidad de los actores, revise la tabla correspondiente.
4. Calcule peso de casos de uso sin ajustar (UUCW), usando la tabla de pesos de casos de uso.
5. Calcule peso de actores sin ajustar (UAW), usando la tabla de pesos de actores.
6. Sume UUCW con UAW para obtener los Puntos de Casos de Uso sin Ajustar (UUCP).
7. Revise los trece puntos de la lista de Factores técnicos:
    a. Haga una sublista sólo con aquellos factores técnicos que tiene influencia sobre su sistema.
    b. Asigne su grado de influencia.
    c. Sume los productos obtenidos en la variable Total factor técnico
    d. Aplique la fórmula para obtener el TCF
8. Revise los ocho puntos de la lista de Factores Ambientales:
    a. Asigne su grado de influencia.
    b. Sume los productos obtenidos en la variable Total factor ambiental
    c. Aplique la fórmula para obtener el ECF
9. Calcule los UCP con la fórmula respectiva
10. Calcule los meses que necesita para terminar el producto, suponga que es su primer proyecto.

V. Resultados esperados:

Los puntos de casos de uso de su proyecto.

Práctica 11. Crear el Modelo del Dominio

I. Objetivo: El alumno establecerá las clases persistentes del dominio de la aplicación que está desarrollando.

II. Equipo necesario:

- Computadora con MS-Windows preferentemente conectada a Internet.

III. Material de apoyo:

- Una herramienta CASE
- El ejemplo que les proporcione en el profesor cargado en la herramienta CASE.
- El modelo Entidad-Relación o de Objetos Semánticos creado durante la especificación de requerimientos.
- Los diagramas de robustez creados durante el análisis.
- El libro de la metodología ICONIX
- Diapositivas sobre ICONIX que están en la página www.uv.mx/personal/jfernandez.
- El libro de Áncora.
- Notas sobre Áncora que están en la página www.uv.mx/personal/asumano.

IV. Procedimiento:

1. Abrir los documentos asociados
2. Abrir la herramienta CASE
3. En el Modelo de Diseño, dar clic derecho y pedir Add Diagram>Class Diagram. Nombrarlo como Modelo de Dominio del sistema, ejemplo: MD SisNomina.
4. Crear una clase por cada entidad u objeto de los modelos E-R u Objetos Semánticos según corresponda.
5. Fijarse si hay nuevas clases Entity en los diagramas de robustez que no se habían considerado y agregarlas.
6. Revisar los atributos
7. Agregar los métodos que les correspondan.
8. Dibujar las relaciones correspondientes: asociación, generalización, agregación.
9. Incluir la cardinalidad de las relaciones.

V. Resultados esperados:
El Modelo de Dominio de la aplicación que están construyendo.

Práctica 12. Diagramas de Secuencia y Diagramas de Clases por casos de uso

I. Objetivo: El alumno aprenderá a realizar Diagramas de Secuencia y los Diagramas de Clases asociados que intervienen en la realización de los Casos de uso de diseño de su sistema de software.

II. Equipo necesario:

- Computadora con MS-Windows preferentemente conectada a Internet.

III. Material de apoyo:

- El CASE elegido, que puede ser StarUML
- El IDE elegido que maneje el lenguaje Java, el cual puede ser Eclipse o NetBeans.
- Ejemplos que les proporcione el profesor cargado en el CASE y en el IDE.
- El libro de la metodología ICONIX
- Diapositivas sobre ICONIX que están en la página www.uv.mx/personal/asumano

IV. Procedimiento:

1. Crear o cargar las formas (ventanas) que se hayan realizado para el prototipo en un IDE asociado con Java. Haga las modificaciones o añadidos que surgieron durante el análisis y que correspondan a las clases *Boundary*.
2. Realizar reingeniería de la GUI
    i. Abra el CASE:
    ii. Abra o cree su proyecto con el enfoque StarUML.
    iii. Use la opción Tools>Java>Reverse Engineerining
    iv. De la ruta en donde se encuentra su proyecto Java
    v. Añada las clases que conforman la GUI dentro del *Modelo de Diseño.*
    vi. De clic en *Run (R).*
    vii. Revise que todas las clases se hayan incluido, si falta alguna, repita los pasos c a f para incluir las clases que faltaron.
3. Crear los Diagramas de Secuencia (uno por cada curso de acción que se haya dibujado en el diagrama de robustez del CU).
    i. Posicione el ratón en Design Model
    ii. De botón derecho y Add Diagram
    iii. Elija Sequence Diagram
    iv. Nombre a su diagrama de acuerdo al CU que realizará y al curso de acción, ejemplo: *CUActualizar*.
4. Dibujar el diagrama de secuencia curso normal
    i. A la izquierda del dibujo agregue un comentario por cada interacción que contendrá el diagrama.
    ii. Arrastre al área de dibujo, desde el Model Explorer (ventana a la derecha de la pantalla de StarUML).:
        1. Un actor que iniciará la acción,
        2. La clase frame (Boundary) que inicia la acción, opcionalmente, nombre al objeto, ejemplo: *OBGUIactualizar.*
        3. Las clases entidad necesarias para ese CU
        4. Las clases de control que decidió que permanezcan en el diseño.

b. Conecte los objetos mediante mensajes o eventos:
    i. Un evento clic o intro, que ira del actor al objeto representante de la clase frame (Boundary).
    ii. Mensajes que contienen métodos del objeto receptor, los cuales pueden:
        1. Ya existir, en cuyo caso selecciónelo de la lista que le muestra el CASE
        2. Necesitarse en el momento, en ese caso agréguelo con el botón rojo a la derecha del nombre del mensaje.

5. Crear diagramas de clases. Al final de la realización de los casos de uso de diseño con diagramas de secuencia, se habrán agregado o modificado clases. Además, es muy posible que ya se haya previsto que el sistema correrá en más de un procesador. Por lo que resulta conveniente que se hagan los siguientes diagramas.
    i. Diagrama de subsistemas, ocupe el diagrama Main del Modelo de Diseño para dibujar los subsistemas use la notación que viene indicada como paquete estereotipado. Asocie los subsistemas según se haya planteado.
    ii. Para cada subsistema dibuje las clases que le correspondan, para ello de clic derecho estando sobre el icono de subsistema, escoja Add Diagram>Class Diagram. Asocie las clases según correspondan.

V. Resultados esperados:
Diagramas de clases por subsistema y diagramas de secuencia por caso de uso, tantos como cursos de acción (pueden agruparse en un solo DS algunos cursos de acción si son muy pequeños).

Práctica 13. Prueba de Integración.

I. Objetivo: El alumno aprenderá a planificar casos de prueba de integración con el enfoque derivado de casos de uso a partir de la descripción de éstos y sus diagramas de secuencia.

II. Equipo necesario:

- Computadora con MS-Windows preferentemente conectada a Internet.

III. Material de apoyo:

- El CASE elegido, que puede ser StarUML
- Los diagramas de secuencia realizados para cada caso de uso
- Notas sobre prueba de integración OO en la página: www.uv.mx/personal/jfernandez

IV. Procedimiento:

1. Para cada diagrama de secuencia, generar los estados posibles de los elementos involucrados.
2. Para cada método en las interacciones incluidas en el diagrama de secuencia: determinar los conjuntos de valores adecuados, según particiones internas y si permiten continuar la cadena de llamadas a otras clases involucradas.
3. Con cada estado y cada representante de conjunto de datos se forma un caso de prueba, combinando los diferentes valores de cada categoría.

V. Resultados esperados:

Para cada caso de uso:

Para cada diagrama de secuencia

Tabla de estados por cada elemento del DS

Tabla de valores válidos y no válidos

Tabla con casos de prueba

Práctica 14. Prueba de Unidad Orientada a Objetos

I. Objetivo:
El alumno utilizará una clase programada por otra persona y aplicará lo visto en clase para planificar, codificar y correr los casos de prueba necesarios para asegurar la correctez de la clase. Lo anterior utilizando el método de rebanada.

NOTA: Esta práctica se divide en tres partes, mismas que se realizan una por cada sesión de dos horas de práctica.

II. Equipo necesario:
- Computadora con MS-Windows preferentemente conectada a Internet.

III. Material de apoyo:
- Texto relativo a Pruebas de rebanadas (ver http://www/uv.mx/personal/jfernandez).
- Texto relativo a JUnit (ver http://www/uv.mx/personal/jfernandez).
- El ambiente de desarrollo Eclipse (opcionalmente puede utilizarse Netbeans).
- Plug-in de JUnit (viene por defecto)
- Plug-in eclemma (bajar de http://www.eclemma.org)
- Impreso de la Clase a probar
- Hojas de papel para la planificación de los casos de prueba
- El maestro entregará a cada equipo (máximo 2 personas) una jarra de Java que contiene la versión ejecutable de la clase a probar, pero no su código fuente. Esto se decide así para evitar que el probador se distraiga tratando de corregir el código.

Parte 1. Preparación de casos de prueba

IV. Procedimiento:
Para esta etapa se realizarán las actividades siguientes:
a) construir el grafo de llamadas mejorado (GLLM) usando la información de de las notas sobre Prueba de Rebanadas
b) construir la matriz de uso mínima (MUM)
c) minimizar la MUM, si es posible
d) analizar la matriz, identificando anomalías, si es que existen
e) generar las rebanadas correspondientes a cada atributo
f) generar las permutaciones necesarias para cada rebanada

V. Resultados esperados:
a) Estructuras de apoyo GLLM y MUM
b) Secuencias de transformadores

Parte 2. Implementación de casos de prueba

En esta etapa se crearán: valores de prueba, casos de prueba, una clase de prueba con métodos correspondientes a cada caso de prueba identificado. *Se recomienda cargarlos en grupos pequeños, para asegurar un avance progresivo y sostenido.*

IV. Procedimiento
a) Para cada parámetro de entrada a los métodos de la clase generar valores de parámetros de acuerdo al criterio de dominios y al de valores extremos (a la

frontera). Se puede consultar las notas sobre Particiones y valores a la frontera en la página http://www/uv.mx/personal/jfernandez.

b) Para cada permutación, generar casos de prueba que incluyan los diferentes tipos de valores identificados en el punto anterior. Cuide de asegurarse que los valores atraviesen los métodos de la secuencia.

Para usar la jarra que se le proporcionará, siga los pasos siguientes:

c) Crear un proyecto en Eclipse.
d) Crear la clase de prueba en el mismo directorio.
e) Copiar la jarra al directorio *default* o *src*, según la versión, dentro de la carpeta Ensayo
f) Registrar la jarra: Project > Properties > Java Buid Path >Libraries > Add External Jar (Browse para que la localice).
g) Escriba los casos de prueba conforme las notas de JUnit
h) Corra la clase de prueba y observe los resultados, no se detenga a ver

V. Resultados esperados:

a) Tabla de valores permitidos y no válidos a considerar en los casos de prueba, usando métodos de particiones y valores a la frontera
b) Tabla de casos de prueba
c) Clase de prueba codificada y correcta
d) Pantalla de corrida de la clase de prueba con los resultados de la aplicación de las pruebas planteadas en la parte 1.

Parte 3. Análisis de resultados

Verificar el grado de cobertura de sus pruebas usando EclEmma. A partir de sus anotaciones acerca de qué casos de prueba pasaron y cuáles no, revise la descripción de la clase y trate de identificar dónde se encuentran los defectos, si es que existen.

IV. Procedimiento:

Para esta etapa se realizarán las actividades siguientes:

g) Instale Eclemma en su ambiente Eclipse.
h) Utilice Eclemma para el análisis de cobertura.

V. Resultados esperados:

c) Ventana de Eclemma con los resultados del análisis como se muestra en la Figura 4 del Manual de Eclemma.

Bibliografía

1. Clemmons, R.K. (2006). Project Estimation with Use Case, CrossTalk.
2. Galindo – Monfil, A.R. (2007). Incorporación de Actividades de Ingeniería de Usabilidad dentro de Áncora, tesis de maestría, Universidad Veracruzana.
3. Jacobson, I., Rumbaught, J., Booch., G. (2000). Proceso Unificado de Desarrollo de Software, Addison – Wesley.
4. Pressman, R. S. (2006). Ingeniería de Software. Un enfoque práctico, sexta edición, McGraw – Hill.
5. Rosenberg, D. (2007). Use Case Driven Object Modeling with UML: a practical approach, Addison-Wesley.
6. Sommerville, I. (2005). Ingeniería de Software, séptima edición, Addison Wesley
7. Sumano, A. (2006). ÁNCORA. Análisis de Requerimientos de Software, Universidad Veracruzana.