1. 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[1] 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.

1.1 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

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

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 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.

1.1.1 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.

1.1.2 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.

1.1.3 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.

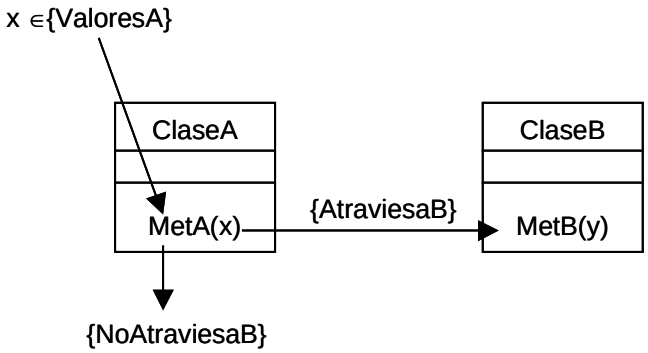

Figura 1 Valores que llegan a ClaseB y otros que no lo hacen

Ahora bien, para probar una interacción entre las clases ClaseA y ClaseB necesitamos 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.

1.2 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.

1.3 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 2 se muestran las clases que participan en un caso de uso de identificación del usuario. Una es una clase fronteriza, otra sirve como 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.

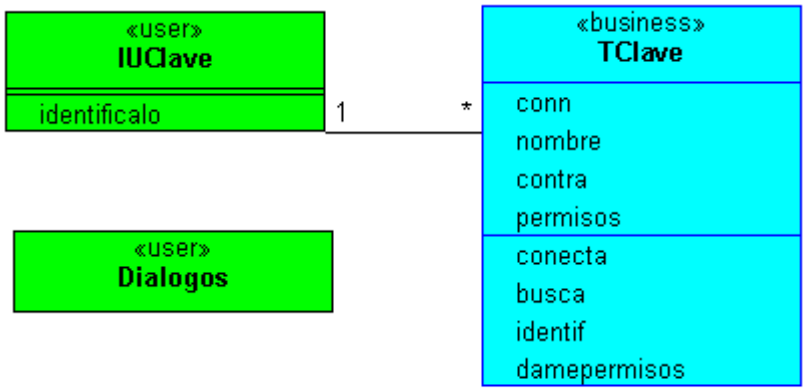

Figura 2. Diagrama de clases del ejemplo

Pasando directamente al diseño, la Figura 3 muestra un diagrama de secuencia para el escenario típico, donde el usuario se identifica satisfactoriamente.

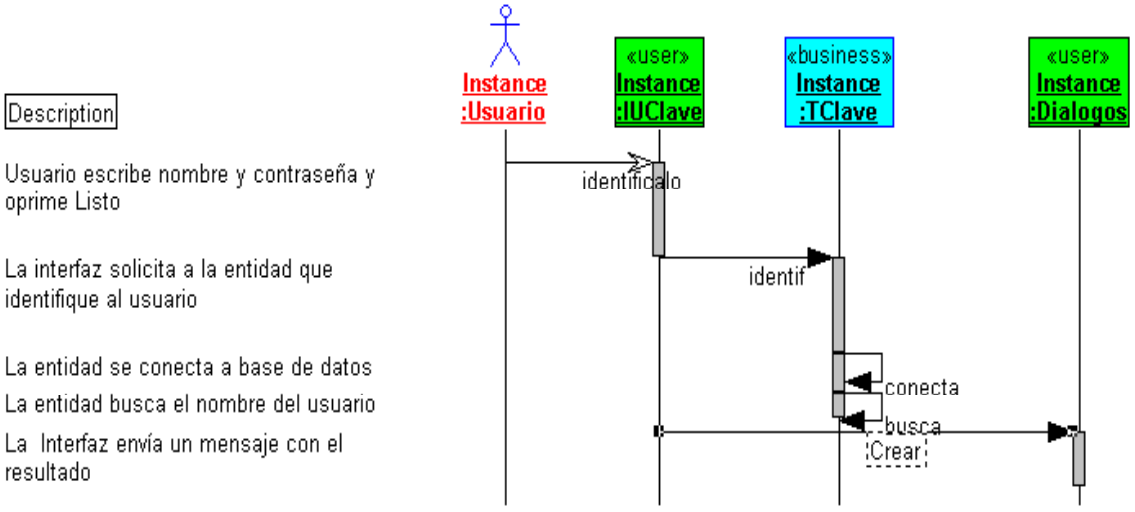

Figura 3. Diagrama de secuencia exitoso de la identificación de usuario

En las Figuras 4 y 5 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 4 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 5 Código de Tclave, del ejemplo

1.3.1 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 identificamos varios 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 2.

Tabla 2 Estados del ejemplo

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

1.3.2 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 3. Se incluyen valores aceptables y no aceptables.

Tabla 3 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)

1.3.3 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 4. 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 4 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.

1.4 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.