DESARROLLO DE UN TESTING CONFIABLE PARA LAS DIFERENTES API'S QUE SE MANEJAN EN LA EMPRESA MAYASOFT INGENIERÍA LTDA

CRISTIAN FERNANDO ALMEIDA ORTEGA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERIAS FÍSICO – MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2011

DESARROLLO DE UN TESTING CONFIABLE PARA LAS DIFERENTES API'S QUE SE MANEJAN EN LA EMPRESA MAYASOFT INGENIERÍA LTDA

CRISTIAN FERNANDO ALMEIDA ORTEGA

Proyecto de Grado para optar al título de
INGENIERO DE SISTEMAS

Director:
EMIRO MUÑOZ JEREZ
Ing. Escuela de Sistemas - UIS

Tutor:
ING. JESÚS DAVID RUEDA POLO
Mayasoft Ingeniería Ltda.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERIAS FÍSICO – MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2011

AGRADECIMIENTOS

Agradezco a mi familia por haber confiado en mí y tener la paciencia para conmigo y mi trayectoria en la universidad. También agradecer a la gente que de alguna forma me ayudó a terminar mi carrera, a los profesores, los cuales con sus enseñanzas me formaron como profesional y persona de bien, a la Empresa Mayasoft Ing., por aceptarme en su institución y hacerme parte de ellos. Y a mi novia Luisa Uribe por apoyarme en todos los momentos buenos y malos.

CONTENIDO

Pág.

LISTA DE TABLAS

Pág.

LISTA DE FIGURAS

Pág.

GLOSARIO

API: Una Interfaz de programación de aplicaciones o API (del inglés Application Programming Interface) es el conjunto de funciones y procedimientos (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser actualizado por otro software como una capa de abstracción.

FRAMEWORK: Estructura conceptual y tecnológica de soporte definida, normalmente con artefactos de software concretos, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

SQL: Siglas de Structured Query Lenguaje (Lenguaje de Consulta Estructurado), es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas (algebra, calculo relacional).

TESTCASES: Son los casos de pruebas, las diferentes circunstancias en las cuales el método puede ser evaluado (hablando a nivel de programación).

XML: Siglas en inglés de eXtensible Markup Language (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Permite definir la gramática de lenguajes específicos. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

RESUMEN

Título: **DESARROLLO DE UN TESTING CONFIABLE PARA LAS DIFERENTES API'S QUE SE MANEJAN EN LA EMPRESA MAYASOFT INGENIERÍA LTDA.*

Autor: **CRISTIAN FERNANDO ALMEIDA ORTEGA****

Palabras Claves: **PRACTICA EMPRESARIAL, OUTSOURCING, API, TEST, TESTNG.**

DESCRIPCIÓN

El presente proyecto de grado en modalidad práctica empresarial fue desarrollado en la EMPRESA MAYASOFT INGENIERÍA LTDA. En convenio con la universidad Industrial de Santander (UIS), esta empresa se dedica al desarrollo de software y presta el servicio de OUTSOURCING.

El proyecto tiene como objetivo principal, tener un seguimiento de las API's, la funcionalidad y consistencia de los métodos que son usados. Esto con el fin de llevar el control de los métodos tanto localmente para detectar los errores que van resultando a nivel del desarrollo (el cual está en un continuo crecimiento) como la empresa contratista también desea llevar un seguimiento y una verificación del resultado.

Este desarrollo está basado en la plataforma MyEclipse, usando un plug-In (TestNg), el cual permite un mejor desarrollo de los test y mejor visualización de los resultados de los test. Y con la capacidad de poderse ejecutar las pruebas en diferentes bases de datos.

El apoyo de la empresa, con la capacitación, colaboración y asesoramiento prestado. La atención que se tiene con los recién ingresados a la empresa es inmediata, es una ayuda certera y rápida, haciendo más fácil la integración con la empresa y más fácil adquirir la habilidad de solucionar los problemas que se puedan presentar por cuenta propia.

---

* Proyecto de Grado.

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Ing. Emiro Muñoz Jerez. Tutor: Ing. Jesús David Rueda Polo.

SUMMARY

Title: **DEVELOPING A CONFIABLE TESTING FOR THE DIFFERENTS API'S THAT IS HANDLED IN MAYASOFT INGENIERÍA LTDA.***

Author: **CRISTIAN FERNANDO ALMEIDA ORTEGA.****

Keys Words: **BUSINESS PRACTICUM, OUTSOURCING, API, TEST, TESTNG.**

DESCRIPTION

The current degree project in internship modality was developed in MAYASOFT INGENIERÍA LTDA. Company, in partnership with Universidad Industrial de Santander; this company works on the software development and provides outsourcing service.

The main objective of this project is to monitor the API's related to the functionality and consistency of the methods employed locally in order to detect mistakes that arise during the development (which is continuously increasing) as well as to keep track of the results.

This project is developed on the plataform MyEclipse, using a plug-in (TestNg) which provides with a higher development of the test and a better display of the test results carrying out proofs in different data bases.

The support of the company, with the training, collaboration and borrowed advice. The attention that one has with those recently entered to the company it is immediate, it is a good and quick help, making easier the integration with the company and easier to acquire the ability to solve the problems that can be presented self-employed.

* Project of Grade.

** Ability of Physical-Mechanical Engineerings. School of Engineering of Systems and Computer science. Director: Engineer Emiro Muñoz Jerez. Tutor: Ing. Jesús David Rueda Polo.

INTRODUCCIÓN

Dado que la aplicación (la cual desarrolla la empresa (Mayasoft) para la empresa alemana) está en constante crecimiento, este crecimiento se ve reflejado en un mayor grado de complejidad y por ende más difícil de llevar un control de la funcionalidad y robustez de esta aplicación. Por esto, se necesitaba un mecanismo para llevar una revisión, un seguimiento del progreso que día a día se va presentando. Y que sea flexible para que cualquier desarrollador (si hace algún cambio a un método o un método nuevo) pueda hacer él sus casos de prueba o revisar los cambios a los que sometió el método para verificar que su funcionalidad sigue igual.

Con el fin de encontrar un alivio para tener un control sobre los métodos (control con respecto a la calidad de estos), se ve la necesidad de realizar una forma de pruebas, los cuales permitieran llevar un seguimiento de las pruebas hechas, es decir, verificar que aún estas pruebas son satisfactorias, tener la capacidad de ejecutar estas pruebas en las diferentes bases de datos que están usando el sistema e igualmente tener una visualización (más comprensible) de los resultados de las pruebas, para que las personas involucradas en el proyecto puedan conocer el estado del desarrollo, por esto se diseñó unas pruebas las cuales cumplieran con las características anteriormente nombradas y quedara un material que pudiera ser usado durante el tiempo de desarrollo por parte de Mayasoft Ltda.

Es importante destacar el apoyo de la empresa (tanto por parte de Mayasoft) con la capacitación, colaboración y asesoramiento prestado, como por parte de la empresa Alemana dando a conocer los requerimientos y sugerencias que ellos tuvieran con respecto al trabajo realizado. La atención que se tiene con los recién ingresados a la empresa es inmediata, es una ayuda certera y rápida, haciendo más fácil la integración con la empresa y más fácil adquirir la habilidad de solucionar los problemas que se puedan presentar por cuenta propia.

1. PRESENTACIÓN

1.1 DESCRIPCIÓN DE LA EMPRESA

Mayasoft Ing. es una compañía conformada por profesionales especializados en el desarrollo e integración de soluciones informáticas capaces de satisfacer las más diversas necesidades a través de software amigable y de fácil manejo para el apoyo de los procesos empresariales.

Desde la constitución de la empresa, se ha dedicado exclusiva y permanentemente a la investigación, difusión y desarrollo de proyectos de informática, que tienen como características especiales la utilización de conceptos modernos de gerencia de proyectos e ingeniería de información. Esta política ha permitido alcanzar un alto grado de especialización convirtiéndose en nuestra principal fortaleza.

El objetivo de la empresa es convertirse en el mejor aliado de negocios ofreciendo soluciones integradas, desarrollos a la medida y consultoría que le permitan contar con información confiable para la toma de decisiones oportunas y acertadas orientadas a garantizar la sostenibilidad del negocio del contratante.

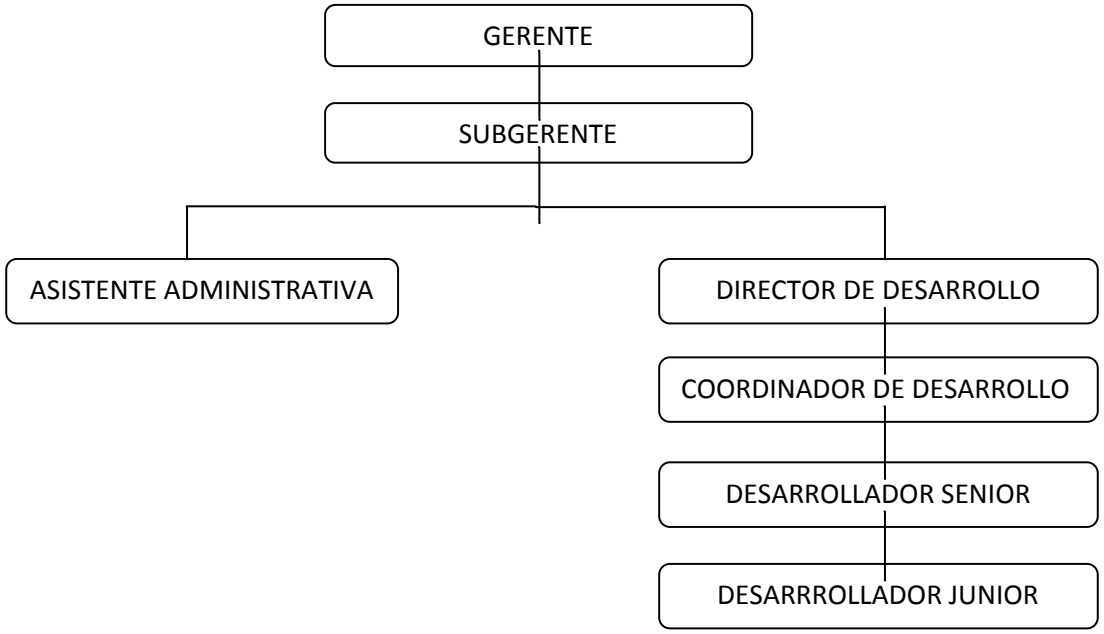
1.1.1 Misión. Construir soluciones informáticas capaces de satisfacer las necesidades de los clientes, proporcionándoles diversas herramientas que les permitan ser más eficientes y así obtener la diferencia que les haga más fuerte frente a sus competidores. Mantenernos a la delantera en temas de tecnología invirtiendo en ello los recursos necesarios en investigación y desarrollo que nos permitan conocer los avances que se presentan en el mundo.

1.1.2 Visión. Construir soluciones informáticas capaces de satisfacer las

necesidades de los clientes, proporcionándoles diversas herramientas que les permitan ser más eficientes y así obtener la diferencia que les haga más fuerte frente a sus competidores. Mantenernos a la delantera en temas de tecnología invirtiendo en ello los recursos necesarios en investigación y desarrollo que nos permitan conocer los avances que se presentan en el mundo.

1.1.3 Estructura Organizacional:

Figura 1. Estructura organizacional.



1.1.4 Situación Actual. Mayasoft Ing. ha estado trabajando en diferentes proyectos. Pero hay uno en especial, es un proyecto que está siendo desarrollado de la mano de una empresa alemana y que este a medida que pasa el tiempo este proyecto va creciendo, y con este crecimiento la forma de poder controlar todos los procesos es cada vez más difícil, a su vez el nivel de complejidad también

crece, gracias a la forma en la cual se llevan los procesos internamente, del cual muchos métodos están entrelazados, por esto también se hace complejo el seguimiento de los posibles errores. Es por esto que surgió la necesidad de tener una forma de hacer pruebas, que pudiera ser comprensible, fácil de llevar un seguimiento de las pruebas, obtener de una forma clara los resultados y que sea robusto para que perdure a medida que el proyecto avanza y además debe ser flexible para que se pueda modificar si las circunstancias lo requieren.

1.1.5 Responsabilidades a cargo. Hacer unas pruebas confiables, para asegurar que los métodos a los que se les hacen las pruebas están funcionando bien y que todas las validaciones son correctas y trabajando en pro de la funcionalidad del método de API.

Asegurar que las pruebas hechas sean fiables al pasar el tiempo y el progreso del proyecto, que estás sean capaces de detectar alguna posible falla por parte de los desarrolladores, ya que estos cuando hagan algún tipo de cambio en los métodos, ellos mismos puedan ejecutar los test y verificar el funcionamiento del método que se acabó de modificar.

Cuando se presente un proceso nuevo y entregado al encargado del proyecto, éste hará entrega del proyecto para realizar los respectivos test, para garantizar el perfecto funcionamiento del proyecto (esta incluido que cualquier detección de un error este se comunicara al desarrollador del proyecto para su respectiva corrección), a los desarrolladores y a los directores del proyecto en Alemania.
Terminado el proceso de testeo subir los métodos a un sitio web, para que puedan ser descargados y ejecutados por cualquier persona involucrada en el proyecto.

1.1.6 Equipo de trabajo. El equipo de trabajo, el cual estuvo involucrado para que este proyecto pudiera realizarse, y en las funciones que realizan en la empresa aporto su conocimiento y tiempo para poder realizar un trabajo eficiente

para que cumpliera con las expectativas de la empresa y de la práctica, que se acoplara a las necesidades que el desarrollo requiera y exige, este equipo está conformado por los siguientes elementos:

1.1.6.1 Director. Es el encargado de dirigir el proyecto. Hace el análisis respectivo y establece los requerimientos y pasos a seguir para el desarrollo del proyecto. Además de llevar un seguimiento del proyecto y brindar el apoyo necesario, ya sea al desarrollador o al tester (persona encargada de las pruebas).

1.1.6.2 Desarrollador. Es la persona a la cual se le ha encomendado el desarrollo del proceso que el director le ha asignado. También se debe encargar de realizar la documentación para una mayor comprensión del proyecto desarrollado. Y entregar el proyecto al tester y brindar soporte al tester en cualquier eventualidad que resulte mientras se realiza el proceso de pruebas.

1.1.6.3 Tester. Está a cargo de realizar el testeo del proyecto y dar la certificación de que (al menos a nivel de api) el proceso está funcionando y tiene las validaciones correspondientes. Estas pruebas se podrían complementar con unas pruebas a nivel de cliente para verificar que el resultado es satisfactorio.

2. OBJETIVOS

2.1 OBJETIVO GENERAL

Desarrollar un sistema de pruebas para los diferentes módulos del software que actualmente está desarrollando la Empresa, procurando un mayor nivel de calidad, y además verificar que el desarrollo y modificación de los métodos cumpla con los requerimientos establecidos.

2.2 OBJETIVOS ESPECÍFICOS

- Explorar la Empresa, su funcionamiento, el método de trabajo, las herramientas utilizadas, para un correcto acoplamiento a las actividades realizadas en ella.

- Desarrollar el sistema de tal forma que permita a:

1. Desarrolladores: Conocer en qué situaciones los métodos están fallando, y así puedan aplicar las correcciones respectivas.
2. Contratistas[1]: Llevar un control sobre el desempeño de los métodos, agilizando la revisión sobre el funcionamiento de estos y respectivas sugerencias que surjan con las revisiones.

- Llevar un control (por parte del sistema) sobre las pruebas, indicando el estado de estas (cuáles pruebas fallaron y cuáles pasaron satisfactoriamente), para que así los desarrolladores revisen las causas por las cuales las pruebas fallaron.

[1] ***"contratistas"*** se hace referencia a los supervisores de Mint (Empresa Alemana) que tienen un vínculo con la empresa Mayasoft en la modalidad Outsourcing.

3. JUSTIFICACIÓN

Es difícil que un software integrado (varios módulos) cumpla con los estándares de calidad, es decir, que no salgan errores. Para alcanzar estos estándares se deben implementar las pruebas para las API's, las pruebas tendrán el objetivo de revisar minuciosamente la calidad de cada módulo; estas pruebas son necesarias porque en la empresa antes se realizaban pruebas que probaban las funciones básicas, pero se omitían algunas pruebas de funciones que están dentro de los módulos.

Los módulos tendrán un autogenerado (Se usa el concepto de **RMI**[2], una buena parte del proceso queda oculto al programador y el cual la clase que comunica el cliente con el servidor (la clase "STUB") es generada automáticamente), el cual será la base para realizar los "testcases[3] ", de esta manera se llevará un seguimiento de los métodos a los cuales se le harán las pruebas e igualmente los métodos que se añaden a lo largo del proyecto. Por tanto al conocer los métodos se procederá a desarrollar las pruebas con todas las variantes de los parámetros que un método pueda tomar, para así detectar los errores. Se empleará la herramienta ***testNG***[4], que facilitará visualizar los resultados de las pruebas y permitirá ejecutar las clases donde las pruebas se encuentran. El lenguaje que se usará para el desarrollo es java. La herramienta de desarrollo usada en la Empresa es MyEclipse.

Al finalizar la práctica se busca tener un control sobre los métodos de las API's (es decir, detectar los posibles errores, aún después de ser probados, ya que al estar creciendo el software de la empresa y su complejidad, se realizan cambios que

[2] **"Remote Method Invocation"** que hace que un objeto invoque a otro objeto remoto con independencia de la JVM o el servidor en el que se encuentra, como si fuera un objeto local.
[3] "***testcases"*** son las funciones que prueban los métodos de las API's
[4] ***testNG*** Es la herramienta que permite ejecutar las clases que contienen los test y facilita visualizar los resultados de estos.

afectan otros métodos y que no se contemplan al momento de concluir con el desarrollo de nuevos requisitos), De esta manera se espera haber contribuido al mejoramiento de la calidad en el desarrollo.

Entenderse con profesionales es importante para el estudiante, ya que complementa el perfil de ingeniero, entendimiento del trabajo en equipo y adquisición de nuevos conocimientos, por lo expuesto anteriormente se considera una oportunidad para aprender sobre el ambiente de trabajo, las tecnologías manejadas y fortalecer los conocimientos adquiridos en la carrera.

En la actualidad, la Empresa está trabajando en la modalidad de Outsourcing para una Empresa Alemana llamada Mint. La cual envía los nuevos requisitos o las correcciones que surjan a través de las revisiones que ellos realizan y crean pertinentes para cada caso; también entregan las especificaciones y los requerimientos para los nuevos desarrollos. Esta es la razón por la cual se utilizan extranjerismos ya que es la única forma de comunicación con la Empresa Alemana es por medio del idioma Inglés.

4. METODOLOGÍA

Para facilitar el desarrollo del proyecto, se debe buscar una metodología que tenga en cuenta: la importancia y necesidad del proyecto. El software desarrollado va en constante evolución y se acoge a la metodología evolutiva. Y para el desarrollo del proyecto se acogerá también esta metodología, ya que especifica una base para el desarrollo de las pruebas, luego se hace un diseño de cada uno de los módulos y se desarrollan las pruebas para que sean revisados y en consecuencia, se atenderán luego las especificaciones que salgan de estas revisiones. Finalmente, cuando el proyecto crece, las pruebas deben estar en constante evolución para acoplarse a este crecimiento. A continuación se expone la idea:

Un modelo de ciclo de vida define las fases por las cuales se desarrollará el proyecto; es una vista de las actividades que se realizarán durante el desarrollo de software e intenta determinar el orden de las etapas involucradas y los criterios de transición asociadas entre estas etapas, suministrando una guía con el fin de ordenar las diversas actividades técnicas en el proyecto y un marco para la administración del desarrollo y el mantenimiento, en el sentido en que permiten estimar recursos, definir puntos de control, monitorear el avance, etc.

Y los modelos de ciclo de vida se resumen en los siguientes pasos:

- Describe las fases principales de desarrollo de software.
- Ayuda a administrar el progreso del desarrollo, y
- Facilita un ambiente de trabajo para la definición detallada del proceso de desarrollo de software.

La metodología de desarrollo evolutiva está relacionada con el desarrollo por prototipos, esta metodología consiste en desarrollar un producto inicial y mostrarlo a los usuarios para que hagan sus solicitudes (cambios en requerimientos o petición de requerimientos nuevos), así se estarían liberando versiones del producto, que sean funcionales y en cada liberación, el producto tendrá incorporado nuevos requerimientos o modificaciones pedidas por los usuarios, que no estaban consideradas en las especificaciones iníciales, satisfaciendo las necesidades o caprichos de los clientes.

Figura 2. Redefinición de problemas en el modelo evolutivo.

Fuente: http://es.scribd.com/doc/18286606/Modelo-de-Desarrollo-Evolutivo.

La ventaja de un proceso del software que se basa en un enfoque evolutivo, es que la especificación se puede desarrollar en forma creciente. Tan pronto como los usuarios desarrollen un mejor entendimiento de su problema, éste se puede reflejar en un sistema de software. Por otro lado, los cambios son inevitables en todos los proyectos de software grandes. Los requerimientos del sistema cambian cuando el negocio que procura el sistema responde a las presiones externas[5].

4.1 LEYES DE LA EVOLUCIÓN DEL SOFTWARE SEGÚN LEHMAN:[6]

- Ley I: CAMBIO CONTINUO.
- Ley II: COMPLEJIDAD.
- Ley III: AUTORREGULACIÓN.
- Ley IV: CONSERVACIÓN DE LA ESTABILIDAD ORGANIZATIVA
- (VELOCIDAD DE TRABAJO INVARIANTE.
- Ley V: CONSERVACIÓN DE LA FAMILIARIDAD.
- Ley VI: CRECIMIENTO CONTINUO.
- Ley VII: CALIDAD DECRECIENTE: Los programas (se hace referencia al software desarrollado por la empresa) serán percibidos como de calidad decreciente a menos que se mantengan de manera rigurosa y se adapten al entorno operativo cambiante. Esta percepción de la calidad decreciente tiene que ver con los cambios en los criterios de aceptabilidad de los usuarios.

[5] Tomado de: http://es.scribd.com/doc/18286606/Modelo-de-Desarrollo-Evolutivo
[6] Lehman, M.M. (1997) Laws of Software Evolution Revisited, pos. pap. EWSPT96, Oct. 1996, LNCS 1149, Springer Verlag, 1997, pp. 108-124

4.2 PLAN DE TRABAJO

A continuación se presentan las principales fases del proyecto y el plan de trabajo a seguir en cada una:

4.2.1 Planeación. En esta primera etapa de la práctica, se proyecta una exploración del lugar a realizar el trabajo, el funcionamiento de la empresa y la apropiación del método de trabajo para un correcto acoplamiento a las actividades realizadas en la empresa. Para el cumplimiento de este objetivo, se planeó la realización de algunas actividades:

- Introducción a la empresa (conocimiento de metodología de trabajo y estructura organizacional de la empresa).
- Definición de la eficacia del proyecto (De la utilidad y efectividad del proyecto).
- Identificación de la metodología.
- Estudio de las herramientas de desarrollo.
- Estudio del software (los módulos que contienen los métodos de las API's a los cuales se les desarrollara las pruebas) que desarrolla la Empresa.
- Revisión y ajuste del plan de trabajo (es necesario luego del estudio los métodos de las API's que contiene el software).

4.2.2 Desarrollo. En este periodo se pondrá en práctica los conocimientos adquiridos y de igual manera se aplicarán y aprenderán nuevos, innovando la metodología que utilizaba la empresa Mayasoft Ingeniería Ltda. (ya que antes se usaban test por separado para cada API, y solo se realizaban pruebas para algunos métodos y los otros métodos del API eran omitidos, entonces se hizo un cambio para poder ejecutar las pruebas más fácil, poder llevar un control sobre todos los métodos del API, y una mayor comprensión de los resultados de estas pruebas, por esto se mejoró con respecto a la forma como se hacían las pruebas,

en el control de las pruebas y la forma de análisis de los resultados, cubriendo así un mayor rango el API contra posibles errores), siguiendo el plan de trabajo definido:

- Estudio de prioridades de los módulos de la aplicación que desarrolla la empresa.
- Diseño de un boceto (tener una primera visión del método y su función para ir analizando los posibles casos de prueba) para facilitar la realización de las pruebas en los diferentes métodos de los módulos.
- Observación sobre las variantes y número de casos que pueden resultar por cada método.
- Revisión de la descripción de cada uno de los métodos en el Javadoc[7] y reporte sobre posibles fisuras de este.
- Elaboración de los test-cases (código).
- Entrega de los test-cases y correspondiente informe sobre fallos encontrados para que se realice la reparación pertinente y sean nuevamente revisados.
- Revisiones sobre estructura de las pruebas y elaboración de los cambios sugeridos.

[7] ***Javadoc:*** es la documentación del código. Debe contener la descripción del método, su función, descripción de los parámetros del método, las excepciones que ocurren, el autor…etc. En la visualización de los resultados mientras y después que se ejecutan, se pueden observar. En una pestaña que tiene exclusiva en el eclipse los resultados están apareciendo a medida que un test haya terminado

5. MARCO TEÓRICO

5.1 ORACLE

“Oracle es básicamente una herramienta cliente/servidor para la gestión de base de datos, es un producto vendido a nivel mundial, aunque la gran potencia que tiene y su elevado precio hace que solo se vea en empresas muy grandes y multinacionales”[8].

En la empresa Mayasoft se tiene un servidor (Sistema operativo Linux, Red Hat), el cual tiene Oracle instalado y en este servidor se instalaron las bases de datos de prueba, de los diferentes clientes que usan el software desarrollado en la empresa.

Los desarrolladores se conectan a este servidor, para poder acceder a estas bases, para poder ejecutar el software y hacer sus respectivas pruebas (pruebas de desarrollador) con diferentes bases de datos y así comprobar que el desarrollo está bien.

También se puede acceder a la base de datos para realizar consultas sobre los datos y probar las sentencias SQL necesarias para el desarrollo de las aplicaciones.

“Entre sus logros cuentan con la construcción del primer sistema comercial de base de datos relacional. Vendieron el primer producto que empleaba SQL (lenguaje de preguntas estructuradas), hoy un estándar en la industria”[9].

Oracle se puede ejecutar prácticamente desde cualquier maquina (hardware),

[8] Tomado de: http://www.articulosinformativos.com.mx/Que_es_Oracle-a1073091.html
[9] Tomado de: http://www.articulosinformativos.com.mx/Que_es_Oracle-a1073091.html

además no solo soporta los datos tradicionales, sino también es capaz de almacenar archivos completos, como lo son archivos de texto, imágenes, archivos de audio, etc.

Oracle es un manejador de base de datos relacional y posee tres grandes aspectos:

- Estructuras: Definición de objetos que contengan datos y que son accesibles por los usuarios.
- Operaciones: Definir acciones que manipulen datos u objetos.
- Reglas: Leyes para gobernar la información, cómo y qué manipular.

5.2 INTRODUCCIÓN A JDBC

"La idea básica de JDBC (Java Data Base Connectivity) es asistir a los programadores de API (Application Programming Interface), puedan codificar de manera independiente del fabricante del gestor de base de datos, una conexión entre la aplicación que desarrolla y la base de datos la cual desea utilizar"[10].

JDBC posee dos partes importantes: el API JDBC el cual contiene las clases que usa el programador y el controlador el cual contiene las ordenes que envía el programador y este las interpreta para el gestor de la base de datos.

5.2.1 Conexión a la base de datos. Para realizar la conexión a la base de datos se realizan los siguientes pasos:

1. Registrar el controlador: por medio del método **Class.forName(String de driver)** (forma dinámica) o por **System.setProperties("jdbc.drivers", String**

[10] Tomado de: http://www.proactiva-calidad.com/java/jdbc/index.html

de driver) (forma estática).

2. Con el método **DriverManager.getConnection(String de la URL de base de datos, String de login, String de password)** se establece la conexión a la base de datos.

   Ejemplo de conexión en Oracle:

   - Connection con = DriverManager.getConnection("jdbc:oracle:thin:servidor:1521:prueba", "log", "pwd");

5.2.2 Consultas a la base de datos:

5.2.2.1 Sentencias SQL. Se mostrará cómo se pueden hacer consultas a la base de datos[11]:

1. Se define la sentencia deseada y se almacena en una variable tipo **String.**:

   String orden_SQL = "SELECT cliente.codigo, cliente.nombre, cliente.edad

   FROM cliente ORDER BY cliente.nombre";
2. Se crea un objeto de la clase **Statement.**:

   Statement sentencia = con.createStatement();
3. Se hace un llamado al método **executeQuery(String)** para realizar consultas, para inserciones actualizaciones o borrado, se usa el método **executeUpdate(String)**:

   ResultSet rs = sentencia.executeQuery(orden_SQL);

[11] El tema Sentencias SQL esta basado en la pagina WEB: http://www.proactiva-calidad.com/java/jdbc/index.html

4. El método **executeUpdate(),** retorna los datos encapsulados en un objeto de la clase ResultSet, entonces para obtener los resultados (fila por fila) se debe definir un bucle, se muestra un ejemplo a continuación:

```
while ( rs.next() ) {
String res = rs.getString( "codigo" ) + ", " + rs.getString( "nombre" ) + ", " +
rs.getInt( "edad" );
System.out.println( res );     }
```

5. Esta la necesidad de cerrar el objeto de la clase **Statement,** con esto se genera el cierre del objeto ResultSet también. Tampoco se debe olvidar el manejo de las excepciones.

```
catch (SQLException e) { e.printStackTrace();  }
```

6. Y por último, no se debe olvidar el cierre de la conexión **objetoClaseConnection.close()**. En el siguiente ejemplo se muestra la toma de precauciones con los posibles errores:

```
public static void cerrar_conexion( Connection con ) {
   try {
    if ( con != null )
 if ( !con.isClosed() )    // Si no está cerrada, la cierro
     con.close();
         }
 catch (SQLException e) { e.printStackTrace();  }
}
```

5.2.3 Concepto de transacción. “Una transacción es un conjunto de sentencias SQL, que el programador agrupa por razones de la lógica del dominio. Una

transacción se define para cumplir las restricciones de integridad de la base de datos"[12].

Para el manejo de transacciones se manejan los siguientes pasos:

1. Se definen las órdenes que conforman la transacción.
2. Se realiza la confirmación de la operación (COMMIT).
3. En caso de fallo se deshace la operación (ROLLBACK).

5.2.4 Commit y Rollback. Como se nombró en el anterior tópico, cualquier transacción esta delimitada por al confirmación (Commit) y el deshacer (Rollback). Para aplicar el deshacer, se debe capturar el error después de la confirmación, con la excepción SQLException.

Se expondrá un ejemplo (tomado del proactiva) que contiene las operaciones Commit y Rollback:

```
void metodo() {
        try {
            stat.executeUpdate( "INSERT INTO tab_venta VALUES ...");
           stat.executeUpdate( "INSERT INTO tab_usu VALUES ...");
           con.commit();
            catch (SQLException e) {
           deshacer();
        }
}
void deshacer( ) {
```

[12] Tomado de: http://www.proactiva-calidad.com/java/jdbc/transacciones.html

```
            try { con.rollback();  }
                catch (SQLException e) { System.out.println("Error. No hemos
                    podido deshacer." + e.getMessage() ); }
        }
```

Esta fue una vista general de el manejo de las conexiones a base de datos y como se pueden realizar las operaciones, las cuales son los que transforman y manejan los datos (que son el corazón del software, por ellos y para ellos es que se realizan los sistemas) y es importante tenerlo en cuenta ya que en las pruebas, se manejan datos que pueden ser transformados o modificados durante el proceso de prueba.

5.3 JAVA

"Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel"[13].

5.3.1 Orientado a Objetos. Permite hacer un diseño de software de forma tal, que los datos usados están unidos a las operaciones, este conjunto (datos y operaciones) es el objeto.

Los objetos que ahora son coherentes e independientes, que facilitan el diseño de software, para que este sea más estable. También estos objetos permiten que sean usados en otros paquetes del proyecto, es decir, que sean reusables en otras partes del proyecto, haciendo más fácil la creación del software, esto es si se

[13] Tomado de: http://www.cad.com.mx/historia_del_lenguaje_java.htm

hacen objetos más genéricos que permitan esta reutilización y lograr disminuir el tiempo de desarrollo. Esto hace que proyectos grandes sean fáciles de gestionar y manejar.

5.3.2 Independencia de la plataforma. Una característica importante de Java, es la independencia de la plataforma, esto significa que el software desarrollado en java, puede ejecutarse (poner a correr la aplicación) sin importar el hardware.

Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como "bytecode" (específicamente Java bytecode) — instrucciones máquina simplificadas específicas de la plataforma Java. El bytecode es ejecutado entonces en la máquina virtual (JVM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código[14].

5.3.3 El recolector de basura (Garbage Collector). El "garbage collector" o resolector de basura, es el responsable del ciclo de vida de los objetos cuando el programador los ha creado, esto sucede cuando los objetos no tienen referencias en memoria y en ese momento el recolector de basura borrará este objeto, esto evita gastos de memoria innecesarios.

5.4 MYECLIPSE

MyEclipse es un entorno de desarrollo integrado, se hará un breve repaso sobre el producto que es utilizado en la empresa, MyEclipse Enterprise Workbench 7.5. Esta versión ofrece mejoras de velocidad, incluyendo VisualVM Java Profiler, Visual SQL Query Builder, ICEfaces 1,8 además de brindar apoyo y actualizaciones a MyEclipse 's Visual Designers.

[14] Tomado de: http://es.scribd.com/doc/51234443/5/Independencia-de-la-plataforma

MyEclipse 7,5 permite a los usuarios personalizar y mantener sus entornos MyEclipse tanto a nivel individual como empresarial. Las suscripciones a MyEclipse están disponibles anualmente por un precio alrededor de $ 30, $ 50 o $ 150 (dólares) por el Standard, Professional y Blue ediciones respectivamente.

Desarrolladores de todo el mundo eligen MyEclipse porque es el más asequible y completo de J2EE IDE y un conjunto de herramientas de desarrollo Web. MyEclipse es el plugin basado en Eclipse para dar solución a las exigencias de UML, Ajax, Web, Web Services, J2EE, JSP , XML, Struts, JSF, Java Persistence, EJB, incluyendo un amplio soporte de base de datos y una integración de un servidor de aplicaciones el cual cubre muchas necesidades de los desarrolladores. MyEclipse fue lanzado por Genuitec en 2003, con un conjunto de características y el compromiso de ofrecer un entorno de desarrollo completo y accesible a través de comunicados rápidos e iterativos.

La inspiración vino de los fundadores de MyEclipse Genuitec, quienes encabezaron la empresa en 1997, proporcionando consultoría TI. Como una forma de comercializar los servicios de consultoría, pronto comenzaron a ofrecer herramientas de desarrollo libre para descargar en el sitio Web de Genuitec. Después de varios cientos de miles de descargas, el equipo se dio cuenta del potencial en el mercado de las soluciones basadas en Eclipse, y comenzó a desarrollar el primer lanzamiento comercial de MyEclipse. En 2005, Genuitec fue principalmente una empresa producto-base la cual se centró en la evolución de MyEclipse Workbench Enterprise.

Hoy en día Genuitec y MyEclipse son innovadores y liderando la industria con muchas novedades, incluyendo:

- Primer IDE basado en Eclipse con bajo costo, todos los precios de suscripción incluido.

- Primer depurador JSP de niveles para Eclipse.
- Primer lugar en aplicaciones comerciales RCP.
- Primero y en tener la más grande serie de conectores a servidor.
- En primer lugar "Hot Sync", desplegador de aplicación EE/J2EE.
- Primer lugar en depurador JavaScript nativo.
- En primer lugar en Web 2.0 y entorno de desarrollo AJAX.
- Primer IDE en apoyar plenamente Hibernate.
- Primero en tener un completísimo editor de imagen.
- Primer IDE en integrar características NetBeans en Eclipse.
- Primero en crear una plataforma independiente para aplicaciones RCP con SNAPS.

Mirando hacia el futuro, Genuitec continuará sus planes de añadir alta productividad Visual y herramientas para aplicaciones rápidas (RAD) a MyEclipse.

5.5 TESTNG[15]

TestNg es un framework para pruebas, diseñado para tener un rango amplio de pruebas y análisis de resultados que sea comprensible, desde una unidad de prueba (prueba de una clase en particular, aislada de los otras) hasta las pruebas integradas (pruebas de sistemas enteros, varias clases las cuales integran el sistema, varios paquetes (packages)).

Escribir una prueba es básicamente un proceso de tres pasos:

- Escribir la lógica de la prueba e insertar anotaciones de TestNg apropiadas en el código, para garantizar que la ejecución de la prueba sea satisfactoria y muestre el resultado esperado.

[15] Conceptos tomados de la pagina web y traducidos: http://testng.org/doc/documentation-main.html

- Agregar la información acerca de la prueba (por ejemplo, el nombre de clase, los grupos que se desean ejecutar, etc...) en el archivo de testng.xml. Se puede tener la configuración en este archivo para correr las pruebas.
-
- Ejecutar las pruebas en TestNG. Ejecutar las pruebas para conocer los resultados y si es necesario hacer las correcciones correspondientes de las pruebas mismas o de los métodos probados.

Los conceptos utilizados en esta documentación son los siguientes:

- Una “suite” es representada por un archivo XML. Puede contener una o más pruebas y están definidas por la etiqueta <suite>.
- Una prueba es representada por <test> y puede contener una o más clases TestNG.
- Una clase TestNG es una clase Java que contiene (al menos) una anotación TestNG. Está representado por la etiqueta <class> y puede contener uno o más métodos de prueba.
- Un método de prueba es un método Java anotado por @Test en su fuente.

Una prueba de TestNG pueden ser configurada por @BeforeXXX y @AfterXXX, estas anotaciones permiten realizar una lógica de Java antes y después de cierto punto, estos puntos pueden ser cualquiera de los Ítems enumerados anteriormente (<suite>, <test>….).

5.5.1 Información de configuración para una clase de test. @ BeforeSuite: El método anotado dentro de esta etiqueta, se llevará a cabo antes de que todas las pruebas en esa suite se ejecuten.

@ **AfterSuite:** El método anotado dentro de esta etiqueta, se ejecutará después de todas las pruebas en esa suite hayan corrido.

@ **BeforeTest:** El método anotado se llevará a cabo antes de la prueba (<Test>).

@ **AfterTest:** El método anotado se llevará a cabo después de la prueba (<Test>).

@ **BeforeGroups:** La lista de grupos que pertenecen a esta configuración se ejecutarán antes. Este método está garantizado para la ejecución poco antes de que el primer método de prueba que pertenece a cualquiera de estos grupos sea invocado.

@ **AfterGroups:** La lista de grupos que pertenecen a esta configuración se ejecutarán después. Este método está garantizado para funcionar poco después del último método de prueba que pertenece a cualquiera de los grupos que se invocaron.

@ **BeforeClass:** El método anotado se llevará a cabo antes de que el primer método de prueba en la clase actual se invoque.

@ **AfterClass:** El método anotado se llevará a cabo después de todos los métodos de prueba en la clase actual se hayan ejecutado.

@ **BeforeMethod:** El método anotado se ejecutara antes de cada método de prueba.

@ **AfterMethod:** El método anotado se llevará a cabo después de cada método de prueba.

5.5.2 Anotaciones disponibles para aplicar en un test:

@ DataProvider		Marca a un método como el suministro de datos para un método de prueba. El método anotado debe devolver un Object [] [] en la que cada Object [] se puede asignar en la lista de parámetros del método de prueba. El método (@Test) de prueba que desee recibir los datos de este DataProvider necesita utilizar el nombre del dataProvider igual al nombre con el que se hizo esta anotación.
1.	Name:	El nombre de este DataProvider.
@ Test		Marcas de una clase o un método como parte de la prueba.
1.	alwaysRun	Si se establece en true, este método de prueba siempre se ejecutará aun cuando depende de un método que fracasó. Este atributo será ignorado si de dicho test no depende algún método o grupo.
2.	dataProvider	El nombre del DataProvider para este método de prueba.
3.	dataProviderClass	La clase donde buscar el dataProvider. Si no se especifica, el proveedor de datos se examinará en la clase del método de prueba actual o una de sus clases base. Si se especifica este atributo, el método de dataProvider debe ser estático en la clase especificada.
4.	dependsOnGroups	La lista de los grupos del cual el método con la etiqueta depende.
5.	dependsOnMethods	La lista de los métodos que se necesitan que hayan terminado para que el método con la etiqueta se ejecute. No está garantizado el orden en el cual los métodos usados en la dependencia correrán, pero se estará seguro que todos estos métodos correrán antes del método que tiene la anotación al ejecutarse. (Además) si alguno de los estos métodos no fue exitoso, este test no será ejecutado y será marcado con la bandera de (skip) test saltado.
6.	description	La descripción de este método.

7.	enabled	Indica si los métodos de esta clase o método están habilitados.
8.	expectedExceptions	La lista de excepciones que un método de prueba espera que lance. Si no se produce una excepción o se lanza una excepción diferente de las que están en la lista, esta prueba se marcara como fallida.
9.	groups	La lista de los grupos a la cual esta clase o método pertenece.
10.	invocationCount	El número de veces que este método debe ser invocado.
11.	invocationTimeOut	El número máximo de milisegundos el cual esta prueba debería tener por tiempo acumulado de todos los invocationcounts. Este atributo se ignora si el invocationCount no se especifica en el metodo. Si no se no ha tenido retorno o respuesta en este tiempo, el test será marcado como falla.
12.	successPercentage	El porcentaje de éxito que se espera de este método.
13.	sequential	Si se establece true, todos los métodos en esta clase de prueba están garantizados para funcionar de forma secuencial, incluso si las pruebas son en la actualidad se ejecuta con parallel="methods". Este atributo sólo se puede utilizar a nivel de clase y se ignora si se utiliza a nivel de método.
14.	timeOut	El número máximo de milisegundos esta prueba debe tomar. Si no se ha obtenido respuesta del test, se marcara como un error.
15.	threadPoolSize	El tamaño de la piscina de hilos para este método. El método será invocado desde varios subprocesos (piscina) según lo especificado por invocationCount. Nota: Este atributo se ignora si no se especifica invocationCount.

6. DESCRIPCIÓN DE LA PRÁCTICA

6.1 REALIZACIÓN DE LOS TEST

6.1.1 Análisis y Diseño del Proyecto:

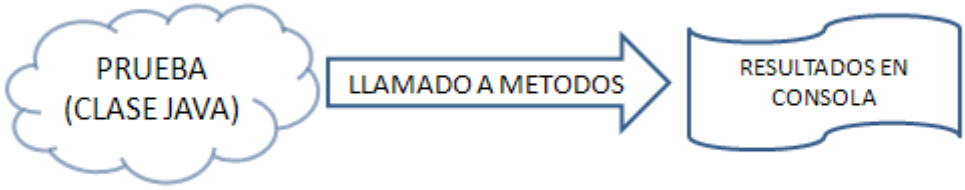
6.1.1.1 Análisis. Había una necesidad de tener una forma de hacer pruebas diferente al modo como se hacía antes. La forma inicial de hacer pruebas, solo cubría los principales métodos, dejando a un lado los métodos que cumplían una función secundaria pero de igual importancia. Al igual que no se podía llevar un registro el cual dejara una constancia las pruebas.

Además no se tenía una conexión directa sobre la base de datos para hacer una comparación de los resultados arrojados por los métodos puestos a prueba. Las pruebas anteriormente se hacían de una forma artesanal.

Las antiguas pruebas eran clases que invocaban a los métodos de API y así se les pasaba los parámetros que el desarrollador o el tester requería necesario para hacer las comprobaciones. En la clase se realizaba cualquier número de llamados al método con las variantes en los parámetros y se imprimía los resultados en consola mientras esta clase era ejecutada.

También en las clases no se llevaba claramente las pruebas hechas, es decir, no se tenía una separación entre pruebas que indicara visiblemente cuales casos de pruebas estaban ya implementados.

Figura 3. Antiguo esquema de pruebas



Teniendo en cuenta las falencias de los casos de pruebas que se realizaban en la empresa. Se debía busca la forma de hacer pruebas de tal forma que:

Un mayor rango de trabajo. (Base de pruebas para las API's existentes)
Facilidad de manejo por parte del desarrollador y el tester. (Facilidad de ejecución y desarrollo de las pruebas, Explicación a desarrolladores sobre las pruebas).

Posibilidad de hacer comparaciones de resultados con la DB. (Comparación de datos, con la base de datos, con los métodos creados para la consulta de datos: getSqlValue, etc.…).

Mayor control sobre los métodos de las API's. (Mejor visualización de resultados para hacer una revisión más rápida y certera de éstos resultados).

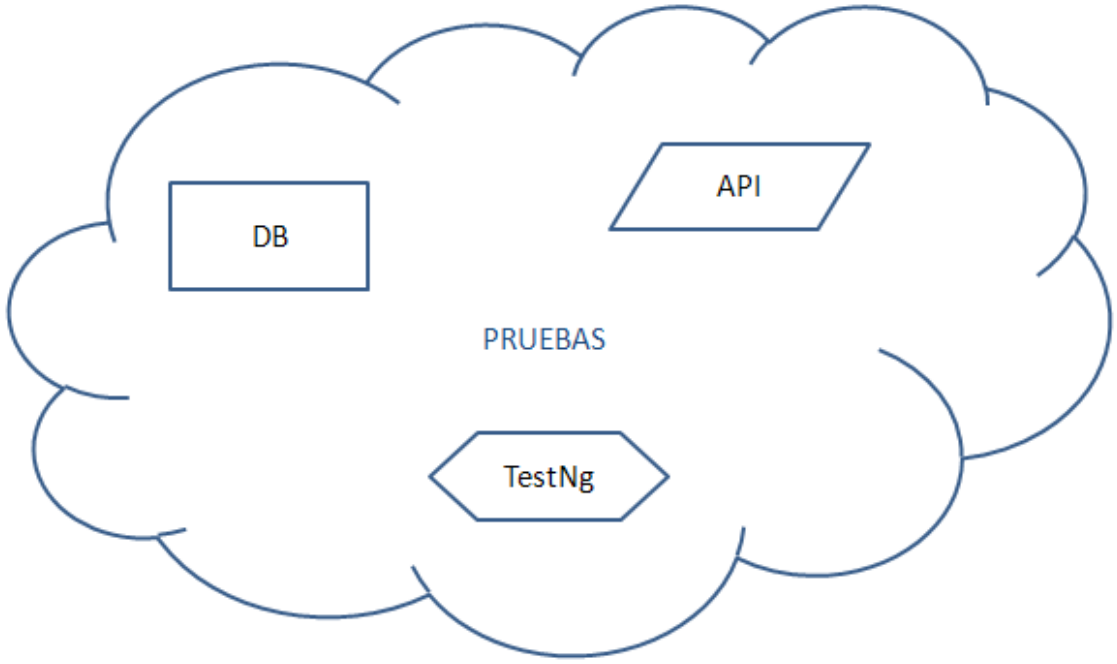
Obtener resultados de las pruebas entendibles por la gente involucrada en el proyecto. (Mejor visualización de resultados, en ejecución o versión html)

6.1.1.2 Diseño. Ya que en la empresa se manejaba el concepto de RMI (el cual comunica servidor con el cliente-GUI por medio de autogenerados), se usó este mismo concepto para generar la base para las pruebas.

Se creó una conexión a la base de datos en una clase padre para poder lograr hacer la conexión a esta y realizar las comparaciones que las pruebas requerían. Se encontró un framework (TestNg) que además de facilitar la visualización de los resultados, ayuda a la creación de las pruebas (tags).

Teniendo lo anterior, se debía crear la conexión entre todas estas herramientas para poder empezar con la codificación de las pruebas.

Figura 4. Esquema actual de pruebas



6.1.1.3 Entrada y salidas. Se define como entradas: las pruebas hechas por el tester (Pruebas en código), ya que en éstas pruebas se definen los datos que entrarán al proceso.

Se define como proceso: los procedimientos internos de los métodos de API según los parámetros que se establecieron al principio (entradas) y que arrojan un resultado.

Se define como salida: los resultados de las pruebas, en el proceso los métodos arrojan resultados, éstos se comparan y con las comparaciones definen si la prueba fue satisfactoria o el resultado retornado no fue el correcto. El listado de pruebas satisfactorias o fallidas son nuestras salidas y con esto se puede hacer un análisis de resultados y determinar los pasos a seguir.

6.1.2 Cómo se empezaron los test. Se debe crear un sistema de pruebas para encontrar lo errores y además llevar el control de los métodos. Para esto se debió hacer una modificación de un archivo Build.xml (usando el concepto que brinda el RMI sobre los autogenerados y que además muestra los métodos a probar, realiza la comunicación entre los test y el API). Este archivo hace una construcción del proyecto, compilando cada paquete para controlar que: no se tengan errores (de compilación) y garantizar que el programa estará limpio para cuando se vaya a usar.

Se modificó este archivo para que creara la interfaz entre el servidor y las pruebas, además de esto, con esta ejecución del Build.xml se creará la lista de métodos que son aptos para hacerles las pruebas (según los requisitos de Alemania se deben hacer pruebas de los métodos que son public static), con esta lista se llevara el control de los métodos a los cuales se les debe hacer las pruebas y además, cuando se introduzca un método nuevo y se ejecute el Build.xml (si cumple los requerimientos) este se añadirá a la lista, cuando haya terminado la ejecución del archivo XML antes nombrado. Esta lista se renueva cada vez que el Build.xml se ejecute.

6.1.3 Creando el ambiente de pruebas y los test. Se tenía la lista de los métodos a los cuales se le debe hacer las pruebas, pero al poco tiempo se puso en evidencia que faltaban datos en algunas bases de datos y los cuales eran esenciales para que una prueba se pudiera ejecutar; al tener este problema se optó por hacer “un ambiente de prueba” el cual consiste en ejecutar antes de cada prueba de una clase, un archivo SQL, el cual contiene sentencias para realizar inserciones en algunas tablas de la base de datos para crear este ambiente de pruebas y que los test (los principales, casos positivos) no fallen por falta de datos, con esto se podían correr los test prácticamente en cualquier base de datos; pero sin dejar a un lado la política para las pruebas en la empresa, la cual consiste en: que al ejecutar las pruebas, la base de datos no sintiera que estas se ejecutaron, en otras palabras, dejar la base de datos como estaba en un principio, entonces se tuvo que añadir otro archivo SQL, el cual debería tener las sentencias que borraran todo lo que se insertó en el primer archivo SQL, esto se hace por que el software es usado por diferentes clientes, que tienen distintos ambientes de datos y por esto se debía buscar la forma de ejecutar las pruebas en estas diferentes bases. Asimismo, se tuvo que considerar que los test se podrían detener en algún momento (ya sea porque se bloquearon las pruebas durante la ejecución o se perdió la conexión de las pruebas con el servidor) y por tanto el último archivo SQL no se ejecutaría, por esto, las sentencias de borrado de los datos también se incluyeron en el archivo SQL que se ejecuta al principio, claro está, estas sentencias de borrado deben estar antes de las inserciones.

Ya que se contaba con un ambiente de pruebas, ahora se necesitaba que se pudieran hacer consultas a la base de datos, esto para que se obtuvieran datos que pudieran ser de utilidad en las pruebas, como por ejemplo la id (una llave única que identifica un empleado) de un empleado que cumpla ciertas características, entonces se realizó una conexión a la base de datos la cual retornara un objeto (Object) (ya fuera una lista de un arreglo de objects, un arreglo de objects o un object) y están identificados por los siguientes nombres

(getSqlRows, getSqlRow, getSqlValue), esto también es útil a la hora de comprobar los datos devueltos por los métodos ya que el método puede que no lance una excepción pero puede que no retorne los datos que se están pidiendo o se están esperando, por eso es muy importante esta clase de consultas y con ayuda de los asserts (assert, assertNotNull, assertEquals) esta comprobación es más eficiente que otras formas de hacer comparaciones.

6.1.4 Aplicando conceptos de Calidad. El término de prueba, no es solo aplicar pruebas a un proceso para verificar su resultado.

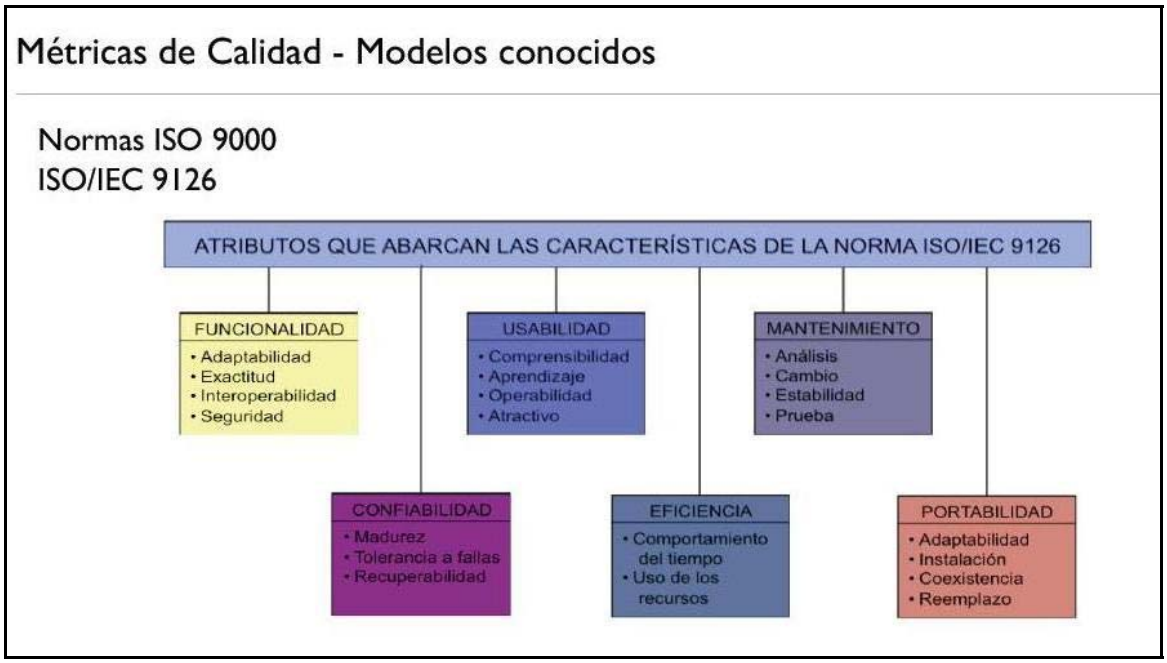
Se debe tener cuidado al darle el visto bueno a un producto, ya que esta es la última fase de desarrollo para que el producto pueda salir a producción, al pasar por las pruebas y si estas son satisfactorias, quiere decir que el producto tiene una calidad aceptable y podrá ser usado por los clientes. Se hará referencia al ISO 9126, el cual es un estándar internacional para la evaluación del software. El ISO 9126 clasifica la calidad del software en un conjunto de características[16] mostradas como sigue:

- Funcionalidad - Un conjunto de atributos que se relacionan con la existencia de un conjunto de funciones y sus propiedades específicas. Las funciones son aquellas que satisfacen las necesidades implícitas o explícitas (Interoperabilidad, Seguridad).
- Fiabilidad - Un conjunto de atributos relacionados con la capacidad del software de mantener su nivel de prestación bajo condiciones establecidas durante un período establecido (Tolerancia a fallos, Recuperabilidad).
- Usabilidad - Un conjunto de atributos relacionados con el esfuerzo necesario para su uso, y en la valoración individual de tal uso, por un establecido o implicado conjunto de usuarios (Operatividad, Comprensión).

[16] Características de la ISO 9126 fueron tomadas de: http://es.wikipedia.org/wiki/ISO/IEC_9126

- Eficiencia - Conjunto de atributos relacionados entre el nivel de desempeño del software y la cantidad de recursos necesitados bajo condiciones establecidas (Comportamiento en el tiempo, Comportamiento de recursos).
- Mantenibilidad - Conjunto de atributos relacionados con la facilidad de extender, modificar y corregir errores en un sistema software (Estabilidad, Facilidad de cambio).
- Portabilidad - Conjunto de atributos relacionados con la capacidad de un sistema software para ser transferido desde una plataforma a otra (Capacidad de instalación, Adaptabilidad).

Figura 5. Métricas De Calidad ISO 9126.



Fuente: Disponible en Internet: http://es.wikipedia.org/wiki/ISO/IEC_9126

Se debe hacer un análisis previo de los requerimientos de los clientes; primero, para dar ideas las cuales pueden causar fallos en la implementación del proceso y segundo, para hacerse una idea de las pruebas que se puedan aplicar al proceso

que está a punto de iniciar. Esto para facilitar y agilizar el proceso de pruebas sobre el proceso.

En la entrega del proceso ya se tiene una idea de lo que el proceso realiza y ya se tiene la idea clara de cuales casos se pueden aplicar, sacando las clases de prueba y sus limitantes para tener en cuenta las pruebas positivas y negativas, para observar el resultado final y analizarlos, para dar a conocer los comentarios positivos y negativos que se observan las pruebas.

Al haber detalles negativos, este proceso debe ser devuelto al desarrollador para que haga sus respectivos ajustes, claro está que deben haber pruebas de desarrollador y estas deben estar documentadas, esta documentación debe tener una descripción del proceso, las variables con las cuales actúa el proceso y las variantes que estas variables puede tener.

Las pruebas también pueden ser revisadas y devueltas al 'tester' para que les haga su respectivo arreglo, si falta alguna validación o el test no está realizando la prueba deseada, o, en otros casos se requiere una robustez del test para que no deba ser cambiado constantemente y perdure un tiempo razonable vigente para el proceso realizado.

Modo de prueba (Concepto de clases de equivalencia)

Cuando se tiene un caso de prueba, a primera vista se puede tener una idea de las pruebas que se puedan realizar. Pero hay q hacer un análisis más profundo ya que hay que tratar de cubrir todas las áreas. Hay que tener en cuenta las fronteras de los casos que se desean probar. Con esto se toma el tema de equivalencias, el cual identifica los estados validos e inválidos; hay generalmente 3 panoramas a planear para definir las clases de equivalencias (limite por derecha (test positivo), limite por izquierda (test positivo), valor invalido (test negativo)). Y para cada clase

identificar los límites para la clase, escribir una lista de valores validos e inválidos en los límites y cuál debe ser el comportamiento previsto.

Los test positivos son los casos en los cuales el proceso debe arrojar un resultado a esa entrada.

Los test negativos son los cuales se espera que el proceso falle y esta falla obviamente debe ser controlada.

Un ejemplo es un proceso el cual tiene como entrada un número entero y se tienen los siguientes puntos:

Tabla 1. Ejemplo entrada números enteros

Entrada: Entero	**Salida: Sumar % al # entrante**
0 - 20	10%
21	15%
22 - 45	20%

Entonces se aplica lo que se comentó al inicio se podrían sacar los siguientes casos de prueba:

Tabla 2. Casos de prueba.

Caso (Entrada) P(x)	**Tipo de test**
< 0	- Negativo (Fuera del rango aceptado)
= 0	+ Positivo (Limite por izquierda del proceso)
= 20	+ Positivo (Limite por derecha del proceso)
= 21	+ Positivo (Limite del proceso)
= 22	+ Positivo (Limite por izquierda del proceso)
= 45	+ Positivo (Limite por derecha del proceso)
> 45	- Negativo (Fuera del rango aceptado)

Se puede observar que salieron 7 casos de prueba, 5 casos positivos y 2 casos negativos.

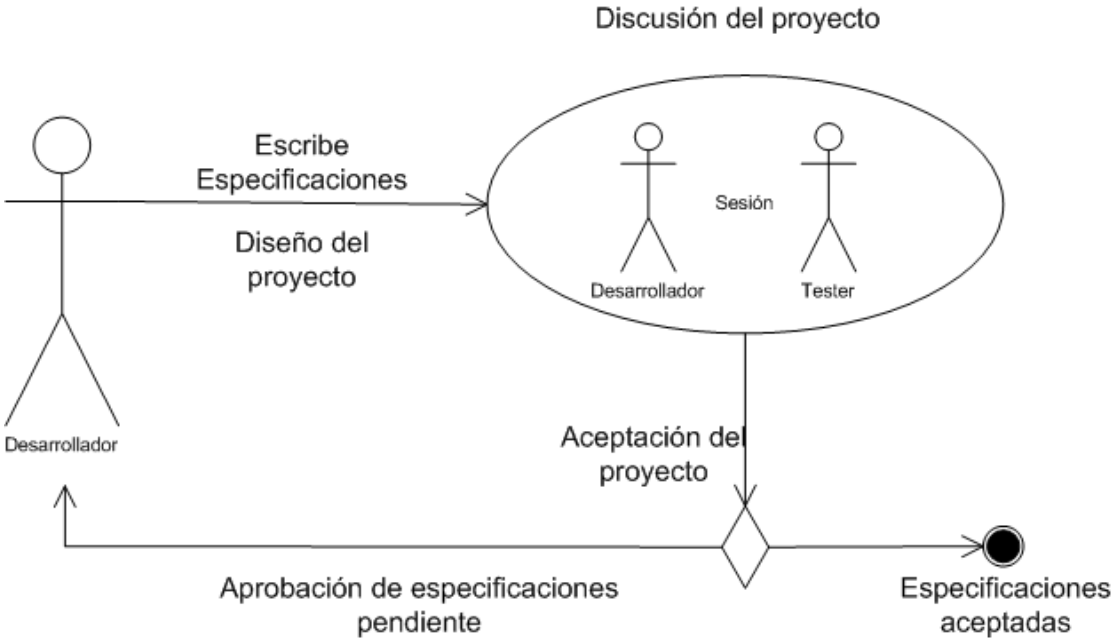
En la mayoría de procesos (y según por experiencia) en casos como el expuesto anteriormente, los desarrolladores no tienen en cuenta los límites del proceso y por esto es clave hacer las pruebas en los límites en los cuales se define el proceso, añadiendo los casos en los cuales en el método no debe trabajar pero debe tener una conducta adecuada cuando esto suceda.

6.1.5 Flujo de trabajo para construir los test. La teoría anteriormente vista, ayuda a tener una idea más clara de forma más acertada de cubrir los casos (métodos) que se van presentando.

Al iniciar un proyecto, se tienen una serie de sesiones las cuales sirven para dar un análisis a los requerimiento iníciales y así evitar errores en el futuro, ya sea por mala comprensión de los requerimiento o bien porque en el transcurso del desarrollo se encontraron detalles que no se tuvieron en cuenta y que se podrían evitar haciendo el análisis de los primeros requerimientos.

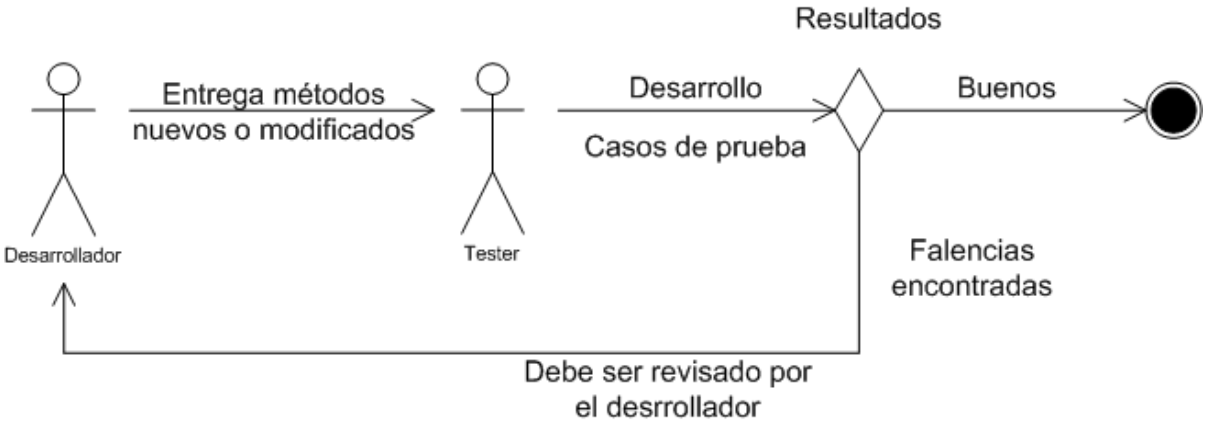
Durante el desarrollo se puede tener una sesión con el 'Tester' para sacar una lista de los posibles casos de pruebas, esto es durante el proceso de desarrollo del proyecto y se hace para tener un bosquejo de las pruebas a realizar.

Figura 6. Sesión para discutir las especificaciones del proyecto



Al finalizar el desarrollo el 'Tester' debe recibir (además del proyecto finalizado) una documentación sobre el proyecto, como se dijo anteriormente, para una mayor compresión del proyecto y tener claro los alcances del nuevo proyecto, y empezar con el proceso de testeo, que en caso de encontrar fallas (el cual es el propósito de hacer este proceso de calidad final) sea devuelto al desarrollador y este pueda conocer cuáles son las fallas encontradas (error de código o de rendimiento). Esto lo puede hacer ejecutando el mismo los casos de prueba que el 'Tester' desarrolló y hacerles el seguimiento respectivo para encontrar el problema y darle una pronta solución, ya que es fácil reconocer el error y pensar en una solución eficaz que sea fiable para el proyecto.

Figura 7. Flujo final del desarrollo (Pruebas).



Cuando se habla de que el desarrollador puede ejecutar los test, es porque las pruebas no quedan en la máquina del 'tester', el proyecto en el cual se trabaja se va guardando en un repositorio que se tiene en un servidor. El cual puede ser accedido con clave y se puede descargar el proyecto. Dentro de este repositorio también se encuentran los test que se realizan y se van subiendo o cargando a este repositorio, por esto, todos los desarrolladores vinculados al proyecto pueden tener en sus máquinas los test y ejecutarlos cuando hayan terminado proyectos en los cuales hayan modificado métodos que ya tuvieran sus respectivos test, para asegurarse de que estén trabajando de manera correcta o al menos las funcionalidades anteriores no hayan sido perjudicadas con el nuevo desarrollo.

6.1.6 Ejemplo, mecánica de los test. El desarrollador termina la elaboración del método, y la entrega al 'Tester' para que este empiece con las pruebas que encontrará las posibles inconsistencias y aseguren la calidad del método.

En este ejemplo se cuenta con un método que retorna la información de empleados, para esto se usa una lista de llaves (único numero para cada empleado) ó con una lista de caracteres (única combinación de caracteres para

cada empleado), el cual debe estar asociado a una registro en la base de datos, para que regrese la información del respectivo empleado o empleados.

Se empieza por ver cuales casos básicos deben funcionar correctamente (casos de prueba positivos).

Figura 8. Casos de prueba positivos

```
/**
 * Test where is used a valid employeeKey in the testGetEmployees method
 * @author Cristian Almeida 26.02.2010
 */

public void testGetEmployees_employeeKey() throws Exception
{
  List<Long> employeeKeys = new ArrayList<Long>();
  Long employeeKey = 380000L;
  employeeKeys.add(employeeKey);
  List<MintEmployee> result = getHumanResourceAPI().getEmployees(employeeKeys, null);
  assert(employeeKey.equals(result.get(0).getKey()));
}


/**
 * Test where is used a valid employeeId in the testGetEmployees method
 * @author Cristian Almeida 26.02.2010
 */

public void testGetEmployees_employeeId() throws Exception
{
  List<String> employeeIds = new ArrayList<String>();
  String employeeId = "id_employee";
  employeeIds.add(employeeId);
  List<MintEmployee> result = getHumanResourceAPI().getEmployees(null, employeeIds);
  assert(employeeId.equals(result.get(0).getId()));
}
```

Luego se revisan las opciones en las cuales las excepciones deben estar controladas (casos de prueba negativos). En la siguiente imagen (Figura 5) se puede observar que se pasan como parámetros valores que son inválidos, por tanto se debe tener un control sobre estos casos; se espera una excepción (en este caso una MintAPIException).

Figura 9. Casos de prueba negativos

```
/**
 * Test where is used an invalid employeeKey null employeeIds in the testGetEmployees method
 * @author Cristian Almeida 26.02.2010
 */

@Test (expectedExceptions = MintAPIException.class)
public void testGetEmployees_invalid_employeeKey_and_null_employeeIds() throws Exception
{
  List<Long> employeeKeys = new ArrayList<Long>();
  Long employeeKey = -1234L;
  employeeKeys.add(employeeKey);
  getHumanResourceAPI().getEmployees(employeeKeys, null);
}


/**
 * Test where is used null employeeKey invalid employeeIds in the testGetEmployees method
 * @author Cristian Almeida 26.02.2010
 */

@Test (expectedExceptions = MintAPIException.class)
public void testGetEmployees_null_employeeKey_and_invalid_employeeIds() throws Exception
{
  List<String> employeeIds = new ArrayList<String>();
  String employeeId = "-@#$%12**++~~34L";
  employeeIds.add(employeeId);
  getHumanResourceAPI().getEmployees(null, employeeIds);
}
```

Ahora se da un vistazo a los casos especiales, en este caso se envían más de 1000 registros para evaluar el rendimiento del método.

Figura 10. Casos especiales keys

```
/**MINT-513
 * used more than 1000 keys in the getEmployees method
 * @author Cristian Almeida 26.04.2010
 */

public void testGetEmployees_with_more_than_1000Keys() throws Exception
{
  List<Long> employeeKeys = new ArrayList<Long>();
  List<Object[]> keys = getSqlRows("select employee_key from employee where rownum < 1300");
  for(Object[] objects : keys){
    employeeKeys.add((Long)objects[0]);
  }
  List<MintEmployee> result = getHumanResourceAPI().getEmployees(employeeKeys, null);
  assertEquals(result.size(),keys.size());
  for (MintEmployee e : result){
      assert(employeeKeys.contains(e.getKey()));
  }
}
```

Figura 11. Casos especiales ids

```
/**MINT-513
 * used more than 1000 Ids in the getEmployees method
 * @author Cristian Almeida 26.04.2010
 */

public void testGetEmployees_with_more_than_1000Ids() throws Exception
{
  List<String> employeeIds = new ArrayList<String>();
  List<Object[]> keys = getSqlRows("select id from employee where rownum < 1300 and id is not null");
  for(Object[] objects : keys){
    employeeIds.add((String)objects[0]);
  }
  List<MintEmployee> result = getHumanResourceAPI().getEmployees(null, employeeIds);
  assertEquals(result.size(),keys.size());
  for (MintEmployee e : result){
      assert(employeeIds.contains(e.getId()));
  }
}
```

Figura 12. Casos especiales null ambos parámetros

```
/**MINT-513
 * Test where is used a valid employeeId in the testGetEmployees method (must return all employees)
 * @author Cristian Almeida 26.02.2010
 */

public void testGetEmployees_null_employeeKeys_and_null_employeeIds() throws Exception
{
  List<MintEmployee> result = getHumanResourceAPI().getEmployees(null, null);
  Long count = (Long)getSqlValue("select count(*) from employee");
  assertEquals(result.size(),count.intValue());
}
```

6.2 UTILIZACIÓN DE LAS ANOTACIONES TESTNG

A continuación se realizara una breve descripción de las anotaciones utilizadas en las pruebas, las cuales son el objeto de la práctica realizada. Además para dar una idea más clara del uso de las anotaciones y su aplicación como puede ayudar durante se el proceso de prueba.

6.2.1 @BeforeClass:

Figura 13. Etiqueta @BeforeClass

```
@BeforeClass
public void setupBeforeClass()
{
  // WL, 30.10.2009: annotation in base class prevents test runs after this failed one time
  super.setupBeforeClass();
}
```

Lo que este dentro de esta anotación se correrá antes de que la clase de prueba empiece a ejecutar los test. En esta ocasión se usa para correr unas sentencias SQL (hacer unos insert), que son los que crean un ambiente de prueba para garantizar que la mayoría de los test puedan correr satisfactoriamente, como se decía anteriormente la mayoría de estas sentencias SQL son inserciones, ya que se deben hacer pruebas en diferentes bases de datos, y en muchas ocasiones estas bases no tienen datos fundamentales para que los test corran cómodamente.

En ocasiones puede fallar ejecutando estas sentencias SQL, y al pasar esto, los test no se ejecutaran, ya que no se garantiza el resultado que arrojen los test. Saldrá un mensaje indicando que falló en el método @BeforeClass y arrojara también cual fue el error Oracle (nombre del fallo y el motivo del fallo), esto facilita chequear las sentencias SQL y hacer el respectivo arreglo.

6.2.2 @AfterClass:

Figura 14. Etiqueta @AfterClass

```
@AfterClass
public void setupAfterClass()
{
  // WL, 30.10.2009: annotation in base class prevents test runs after this failed one time
  super.setupAfterClass();
}
```

Lo que se encuentra dentro de esta anotación, correrá después del conjunto de las clases de pruebas. Para estas pruebas esta anotación se usó con el fin de escribir sentencias SQL, pero a diferencia de la anterior, lo que está contenido en estas sentencias SQL son (delete), esto se debe a la política que se maneja en la empresa, la cual consiste en dejar (en lo más posible) la base de datos como estaba en un principio, esto se debe a que las bases son diferentes y no se quieren modificar estas diferencias ósea no hacer una especie de unificación de bases, ya que son fundamentales a la hora de hacer las pruebas usuario.

6.2.3 DependsOfMethods:

Figura 15. Anotación dependsOnMethods

```
@Override
@Test (dependsOnMethods = {"testGetCurrencyAdminRequirement" , "testCopyRequirement"})
public void testDeleteRequirement() throws Exception
{
  Long requirementKey = 25002L;
  getTreeAPI().deleteRequirement(requirementKey, null);
}
```

Se puede observar cómo debe quedar la anotación dependsOfMethods en la Figura 14, en este se observa que antes de ejecutar la prueba, se debieron haber ejecutado las pruebas registradas en la anotación, se contempla que se debe copiar el nombre de los métodos los cuales se desea que sean condición de la prueba a la cual se le hace la anotación, cabe anotar que si alguna de las pruebas (que son precondición) no corre satisfactoriamente, el método que usa la anotación no se ejecutara.

6.2.4 TimeOut:

Figura 16. Anotación TimeOut

```
/**
 * set null employee in getCompleteEmployee
 * @author Cristian Almeida 08.01.2010
 */
@Test (timeOut = 30000)
public void testGetCompleteEmployee_null_employeeKeys() throws Exception
{
  Object[] key = getSqlRow("select attribute_key from attr_value_employee where rownum = 1");
  getHumanResourceAPI().getCompleteEmployee(null, Collections.singleton((Long)key[0]), false);
}
```

Esta anotación se usa para verificar el rendimiento de los métodos, esta medición se hace en milisegundos, en la Figura 15 se nota que se tiene un tiempo de espera a 30 segundos. Cuando se ejecuta la prueba tiene un tiempo límite de 30 segundos para que salga con resultado satisfactorio, pero al pasarse de este tiempo la prueba se marcara como error, e indicara que fallo porque se demoró más del tiempo que se estipulo, y esto indica que se debe revisar este método ya que se colca un tiempo razonable y tolerable por parte de los clientes para la ejecución de método.

6.2.5 ThreadPoolSize and invocationCount:

Figura 17. Anotaciones threadPoolSize e invocationCount

```
@Test(threadPoolSize = 7, invocationCount = 100, timeOut = 30000,enabled = false)
public void testGetEmployeeListRecordItems_multiUser() throws Exception
{
MintClientSession session = createNewSession();
try{
  List<Long> employeeKeys = new ArrayList<Long>();
    Long employeeKey = 80005L;
    employeeKeys.add(employeeKey);
    Long recordItemKey = 200008L; //use document_key of employee_document
    MintEmployeeRecordItem result = session.getClientAPI().getResourceAPI().getEmployeeListRecordItems(employeeK
    assertEquals(employeeKey, getSqlValue("select employee_key from employee_document where employee_doc_key = "
    assertEquals(recordItemKey, getSqlValue("select document_key from employee_document where employee_doc_key =
}
finally{
  session.logout();
}
}
```

La anotación de invocationCount es el número de veces el cual se quiere ejecutar un método uno tras otro, y termina cuando el número de iteraciones sea igual al escrito en la acotación. La anotación threadPoolSize debe ir en conjunto con la anotación invocationCount, esta anotación se trabaja colocándole un número, el cual identifica el número de conexiones en la piscina de conexiones de la cual se invocara la prueba. Según el número que haya colocado en esta anotación, la prueba se ejecutara simultáneamente ese número de veces, y se llamara en grupos (igual al número de la piscina de conexiones) hasta completar el número que está en la anotación invocationCount. En la Figura 16 se nota que de una piscina de 7 conexiones el método se invocara 100 veces. Entonces la prueba se ejecutara un número de 7 veces simultáneas, hasta completar los 100 llamados, anotados en el invocationCount.

6.2.6 DataProvider:

Figura 18. Etiqueta @DataProvider

```
@DataProvider(name = "useCases")
protected Object[][] datasourceTasks()
{
  List<Object[]> result = new ArrayList<Object[]>();

  Object[] params = new Object[2];
  params[0] = "location";
  params[1] = EMintPropertyUseCase.AUC_LOCATION;
  result.add(params);

  params = new Object[2];
  params[0] = "organisation";
  params[1] = EMintPropertyUseCase.AUC_ORGANISATION;
  result.add(params);

  params = new Object[2];
  params[0] = "schedule";
  params[1] = EMintPropertyUseCase.AUC_SCHEDULE;
  result.add(params);
```

Esta anotación se usa para poder pasar parámetros a los métodos, y con esto la opción de usar los valores, con los cuales se puebla esta anotación. Esto se usa para cuando se necesitan usar valores específicos y facilitar los llamados de los métodos. Se observa que se pueden usar un número indeterminado de parámetros ya que este método usa una matriz de objetos (Object [][]), cabe resaltar que se debe colocar un nombre identificando esta anotación para cuando se quiera usar. Cuando se vaya a usar esta anotación en algún método de prueba se debe poner en el inicio de la prueba la anotación @Test dataProvider y se le coloca el nombre que se usó para la anotación anteriormente (nombre del @DataProvider). Luego se debe pasar como parámetros la cantidad de valores que tiene cada fila del dataProvider, para ser usados en la prueba. Los resultados se visualizaran por separado ya que se cuenta como una prueba por aparte cada valor que se pasa en la anotación.

Figura 19. Anotación dataProvider

```
/**
 * several useCases
 * @author Cristian Almeida 08.02.2010
 */

@Test (dataProvider = "useCases")
public void testGetPropertyValues(String tableName, EMintPropertyUseCase useCase) throws Exception
{
List<Object[]> keys = getSqlRows("select ao." +tableName+ "_key , ao.attribute_key, ao.value from attr_
                          "tree_attr_" +tableName+ " tao where ao.attribute_key = tao.attribute_key
                          "and o.attr_" +tableName+ "_key = ao.attribute_key and ao." +tableName+ "_
if(keys == null || keys.size() == 0) throw new SkipException("There aren't data in attr_value_" +tableN
for(Object[] objects : keys){
    List<MintPropertyValue> result = getPropertyAPI().getPropertyValues(useCase, Collections.singleton(
    assertEquals(result.get(0).getRawValue(), objects[2]);
    assertEquals(result.get(0).getObjectKey(), objects[0]);
    assertEquals(result.get(0).getKey(), objects[1]);
}
}
```

En Figura 18 se puede observar que se usan dos parámetros, los cuales son llenados con los valores que se desean en el test, luego se coloca en la prueba el

nombre que se colocó al @DataProvider y se escriben los parámetros para los test, los dos valores que se usaron para llenar las filas de la matriz. En los test se utilizan para facilitar unas consultas SQL.

6.2.7 ExpectedExceptions:

Figura 20. Anotación ExpectedExceptions

```
/**
 * Test used an invalid scheduleId in the getEvents method
 * @author Cristian Almeida 07.07.2009
 */

@Test (expectedExceptions = IllegalArgumentException.class)
public void testGetEvents_invalid_scheduleId() throws Exception
{
  List<MintEvent> result = new ArrayList<MintEvent>();
  List<Long> eventKeys = Collections.singletonList(999999L);
  result = getScheduleDraftAPI().getEventsByKey(null, "invalid-scheduleID", eventKeys);
  assertNotNull(result.get(0).getName());
}
```

Esta anotación se utiliza para dar una lista de excepciones que se esperan que se lancen al ejecutar la prueba, sino es lanzada ninguna de las excepciones de la lista (lanza una excepción diferente a la listada en la anotación) o si la prueba no lanza excepción alguna al terminar la ejecución de esta , la prueba se marcará como error, en la imagen podemos observar que, cuando se ejecute esta prueba, la que tiene un parámetro que es invalido este método debe lanzar una excepción, de no ser así, como se dijo anteriormente la prueba se marcara como error. Debe quedar en nuestras manos revisar si además de venir la excepción, la excepción tenga el mensaje correcto, ósea el mensaje tenga concordancia con el error marcado y sea claro para el cliente.

6.2.8 Override:

Figura 21. Etiqueta @Override

```
/**
 * Test used valid parameters in the getEvents method
 * @author Cristian Almeida 04.05.2009
 */

@Override
public void testGetEventsByKey() throws Exception
{
  List<MintEvent> result = new ArrayList<MintEvent>();
  List<Long> eventKeys = Collections.singletonList(999999L);
  result = getScheduleDraftAPI().getEventsByKey(PUBLISHED_SCHEDULE_KEY, null, eventKeys);
  assertNotNull(result.get(0).getName());
}
```

Esta etiqueta se utiliza para sobrescribir los métodos que se han formado en el autogenerado (el cual indica los métodos que se deben probar) y que al no poner esta etiqueta quedaran dos métodos con el mismo nombre y en los resultados se visualizaran como resultados de dos métodos diferentes. Con esto al correr los test de una api, se podrán visualizar cuales son los métodos que faltan por hacerles pruebas.

6.2.9 Enabled:

Figura 22. Anotación Enabled

```
/**
 * Test used an invalid objectKey in the updateAssignments method (EMPLOYEE_TASK)
 * @author Cristian Almeida 29.04.2009
 */

@Test (expectedExceptions = IllegalArgumentException.class, enabled = false)
public void testUpdateAssignments_invalid_objectKey() throws Exception
{// CD, 05/05/2009;  To get this exception is neccessary ask to DB to employees or resources, this take a l
  List<MintTpAssignment> tpAssignments  = new ArrayList<MintTpAssignment>();
  MintAssignment newAssignment          = new MintAssignment();
  List<Long> eventKeys                  = new ArrayList<Long>();
  MintTpAssignment tpAssignment         = new MintTpAssignment();
  List<MintAssignment> assignments      = new ArrayList<MintAssignment>();
  EMintAssignmentDraftType type         = EMintAssignmentDraftType.TASK_EMPLOYEE;

  assignments = getAssignmentsByType(type);
  if(assignments.size() == 0) throw new IllegalArgumentException("There aren't assignments in the period :
  MintTpAssignment assignment = new MintTpAssignment();
  assignment.setAssignment(assignments.get(0));
  eventKeys.add(assignment.getAssignment().getEventKey());
  newAssignment = assignment.getAssignment();
```

Esta etiqueta se utiliza para permitir que un test se ejecute o simplemente se omita. Como se puede observar en la imagen se colocó la etiqueta y se le asignó el valor de ‘false’ para que el test no se ejecute, esto indica que no lo marcara tampoco como saltado, esto es útil cuando se quiere dejar un método el cual se tiene que revisar con más detalle o se puede deshabilitar la prueba ya que se pudo hacer para una base de datos específica, y que seguramente marcara error en otras bases de datos, por esto se puede habilitar la prueba cuando se desee cambiándole el valor a true.

6.3 CONSULTAS

Un factor importante para las pruebas son las consultas a la base de datos. Y para esto se creó una conexión ODBC, se realiza la conexión pasando el nombre de la base de datos: con la dirección ip, el nombre del servicio; además se debe especificar el usuario y la contraseña de este usuario.

Esta conexión no es contemplada por el framework usado (TestNg) y se debió realizar una conexión a la base de datos para poder usar los datos en las pruebas y así asegurar que los resultados concuerden con los valores devueltos o modificados por los métodos puestos a prueba.

Esta conexión se realizó en una clase padre, para que todas las pruebas heredaran de esta clase y pudieran realizar la conexión con solo llamar el método y fuera reusable para cualquier prueba en la cual se necesitara hacer las consultas a la base de datos.

6.3.1 GetSqlValue:

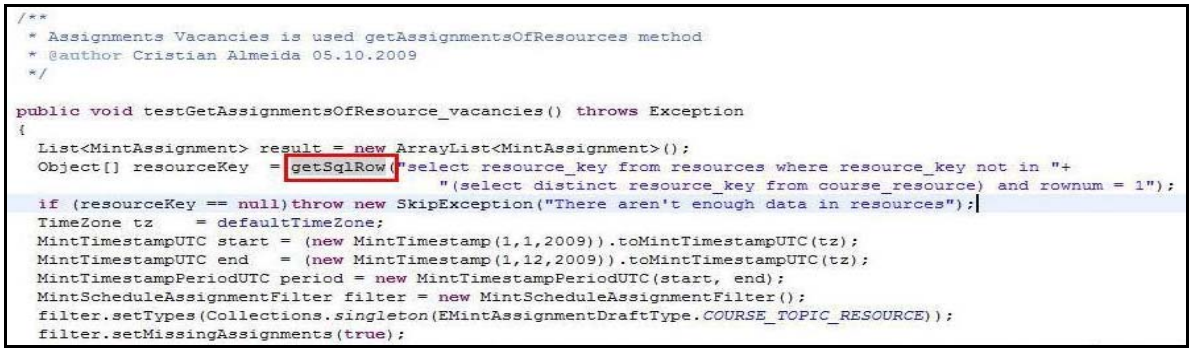
Figura 23. Método getSqlValue para retornar valores (tipo Object) de la BD

```
/**
 * Assignments Vacancies is used getAssignmentsOfEmployees method
 * @author Cristian Almeida 05.10.2009
 */

public void testGetAssignmentsOfEmployee_vacancies() throws Exception
{
  List<MintAssignment> result = new ArrayList<MintAssignment>();
  Long employeeKey  = (Long)getSqlValue("select employee_key from employee where employee_key not in "+
                                        "(select distinct employee_key from course_job) and rownum = 1");
  TimeZone tz    = defaultTimeZone;
  MintTimestampUTC start = (new MintTimestamp(1,1,2009)).toMintTimestampUTC(tz);
  MintTimestampUTC end   = (new MintTimestamp(1,4,2009)).toMintTimestampUTC(tz);

  MintTimestampPeriodUTC period = new MintTimestampPeriodUTC(start, end);
  MintScheduleAssignmentFilter filter = new MintScheduleAssignmentFilter();
  filter.setTypes(Collections.singleton(EMintAssignmentDraftType.COURSE_TOPIC_EMPLOYEE));
```

Se empezará explicando la consulta más básica, getSqlValue(String sql) el cual retorna un objeto tipo Object, y como parámetro recibe una cadena (String), en la cual debe ir especificada la consulta con el dato que se quiere obtener, cuando se hace esta consulta se debe estar seguro que retorne un valor, ya que si la consulta retorna un valor nulo, este método lanza una excepción (nullPointerException). Además este método retorna un solo valor, ósea el primer campo de la consulta hecha.

6.3.2 GetSqlRow:

Figura 24. Método getSqlRow que retorna valores (tipo Object[]) de la BD

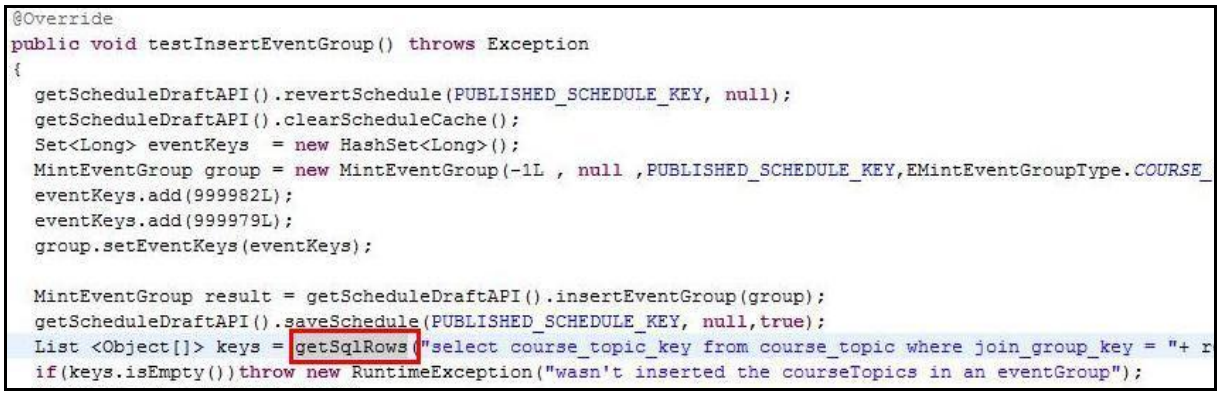


```
/**
 * Assignments Vacancies is used getAssignmentsOfResources method
 * @author Cristian Almeida 05.10.2009
 */

public void testGetAssignmentsOfResource_vacancies() throws Exception
{
  List<MintAssignment> result = new ArrayList<MintAssignment>();
  Object[] resourceKey  = getSqlRow("select resource_key from resources where resource_key not in "+
                                    "(select distinct resource_key from course_resource) and rownum = 1");
  if (resourceKey == null)throw new SkipException("There aren't enough data in resources");
  TimeZone tz    = defaultTimeZone;
  MintTimestampUTC start = (new MintTimestamp(1,1,2009)).toMintTimestampUTC(tz);
  MintTimestampUTC end   = (new MintTimestamp(1,12,2009)).toMintTimestampUTC(tz);
  MintTimestampPeriodUTC period = new MintTimestampPeriodUTC(start, end);
  MintScheduleAssignmentFilter filter = new MintScheduleAssignmentFilter();
  filter.setTypes(Collections.singleton(EMintAssignmentDraftType.COURSE_TOPIC_RESOURCE));
  filter.setMissingAssignments(true);
```

Sigue el método getSqlRow(String sql) el cual devuelve un objeto de tipo Object[] (un array de objetos), el cual permite que hagamos consultas por mas columnas (a diferencia del getSqlValue que solo devuelve un valor), para cuando se necesitan más de un campo de una fila de la base de datos, esto quiere decir que este método retorna la primera fila de la consulta hecha dependiendo de los campos que se hayan especificado en el 'select' de la consulta. Este método si acepta que retorne un valor nulo. Para utilizar los valores retornados por este método se usa Object[0] para el primer campo, Object[1] para el segundo campo, etc.…

6.3.3 GetSqlRows:

Figura 25. Metodo getSqlRows que retorna valores (tipo List <Object[]>) dela base de la BD.



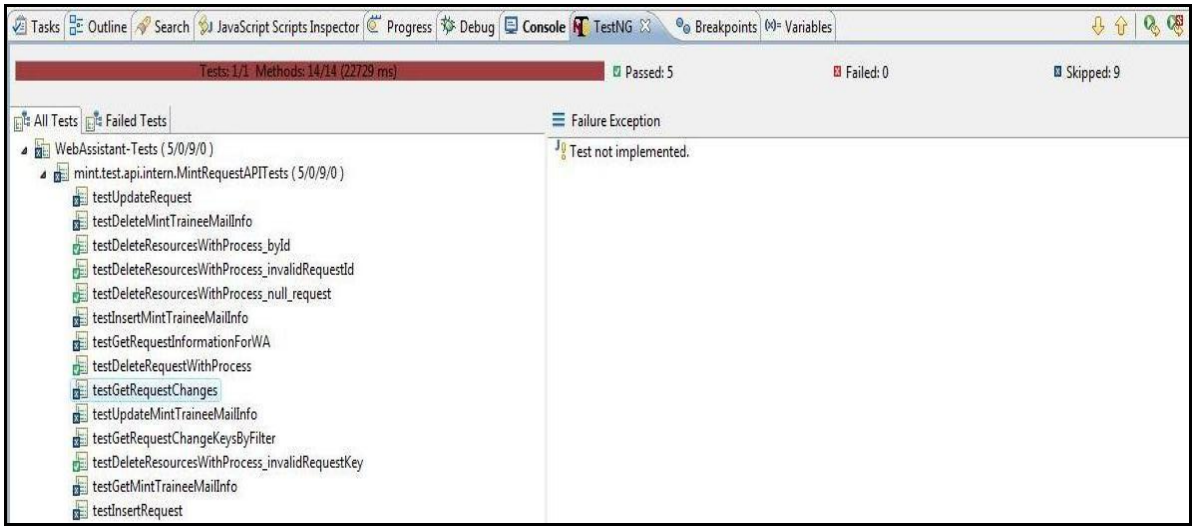
```
@Override
public void testInsertEventGroup() throws Exception
{
  getScheduleDraftAPI().revertSchedule(PUBLISHED_SCHEDULE_KEY, null);
  getScheduleDraftAPI().clearScheduleCache();
  Set<Long> eventKeys  = new HashSet<Long>();
  MintEventGroup group = new MintEventGroup(-1L , null ,PUBLISHED_SCHEDULE_KEY,EMintEventGroupType.COURSE_
  eventKeys.add(999982L);
  eventKeys.add(999979L);
  group.setEventKeys(eventKeys);

  MintEventGroup result = getScheduleDraftAPI().insertEventGroup(group);
  getScheduleDraftAPI().saveSchedule(PUBLISHED_SCHEDULE_KEY, null,true);
  List <Object[]> keys = getSqlRows("select course_topic_key from course_topic where join_group_key = "+ r
  if(keys.isEmpty())throw new RuntimeException("wasn't inserted the courseTopics in an eventGroup");
```

El método getSqlRows(String sql) retorna una List<Object[]>, el cual permite obtener cualquier número de filas de una tabla requeridas para el test. Este método también acepta que la consulta retorne valores nulos, para acceder a los valores que se retornan se debe proceder de la siguiente forma: para la primera fila y el primer valor Object.get (0)[0] o para la segunda fila y el tercer valor Object.get(1)[2] , etc.….

6.4 VISUALIZACIÓN DE LOS DATOS

6.4.1 Test no Implementados:

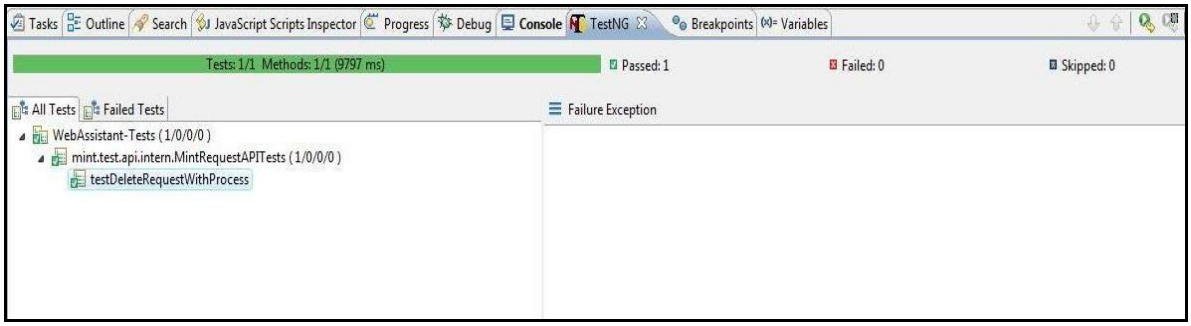
Figura 26. Visualización de datos; Test No Implementados



En la Figura 25 podemos observar como es la visualización de el resultado del test cuando este no se ejecutó y no sobrescribió el test base que se encuentra en otra clase, la cual como se ha dicho contiene los métodos a los cuales se les debe hacer los test. Cuando no ejecuta el test principal el resultado es que el test no se ha implementado para ese método: resultado → 'Test not implemented'. Este tipo de test no implementados se coloca como saltados (Skipped).

6.4.2 Test Satisfactorios:

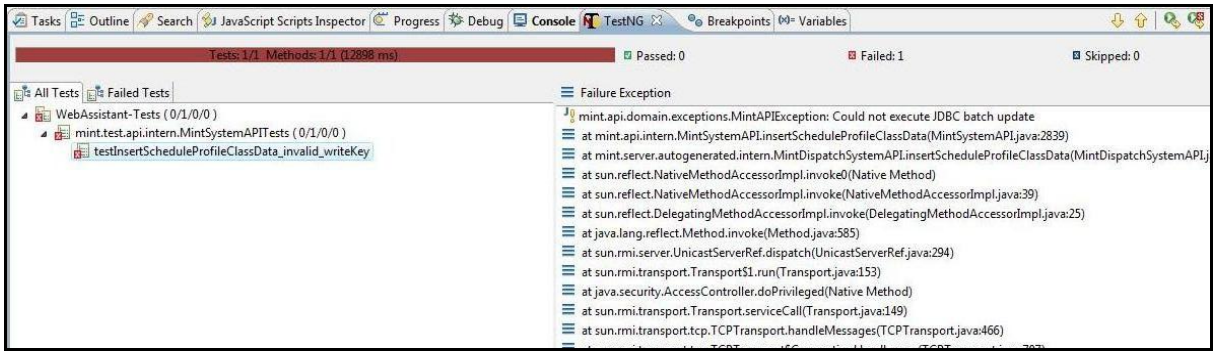
Figura 27. Visualización de datos; Test Satisfactorios



En la Figura 26 podemos observar la visualización de un test que se ejecutó exitosamente. Esta visualización se ayuda con el color verde para tener una mayor comprensibilidad del resultado. También hay una etiqueta (Passed) que indica el número de test que se ejecutaron exitosamente. Dentro de este conjunto de test se encuentran los test que se esperaba un resultado del método el cual coincidió o se esperaba que lanzara una excepción específica la cual fue lanzada por el método según los parámetros pasados.

6.4.3 Test Fallidos:

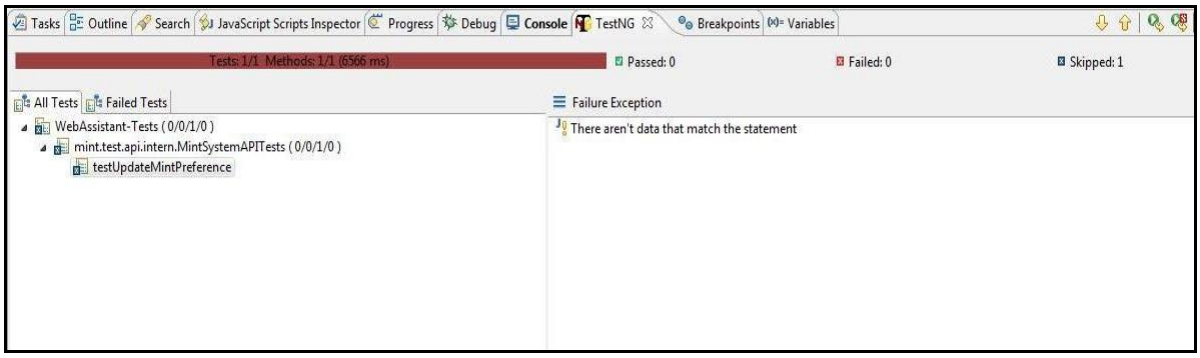
Figura 28. Visualización de datos; Test Fallido



En la Figura 27 se muestra como es el resultado cuando un test que se ejecuto ha fallado. La barra se dibuja de color rojo indicando que hubo un problema con el test. Al igual que en el tópico pasado se observa una etiqueta (failed) que indica el número de test que fallaron. Esto puede ser por que el test lanzo una excepción cuando no se esperaba ninguna o lanzo una excepción diferente a la esperada, también puede marcarse como error cuando el resultado del método no coincide con el esperado entonces en la prueba marcara que hubo un error. Se puede observar que además presenta el tipo de excepción por el cual fallo el test (Failure Exception).

6.4.4 Test Saltado:

Figura 29. Visualización de datos; Test Saltado



En la figura 28 se puede observar que el método ha sido saltado (Skipped), como el la figura 25, pero este tiene una diferencia la cual en el mensaje podemos ver que no dice que no fue implementado, sino que muestra un mensaje que dice que no hay datos que coincidan con la sentencia SQL (esta es una consulta a la base de datos para que devuelva un valor, el cual es necesario para que se ejecute el test), al no devolver valores la consulta este test fallaría y por esto se hace una validación dentro del test para que no marque el test como fracaso, sino que

indique el por qué se evadió el test.

6.4.5 Visualización de Errores en consola:

Figura 30. Visualización de Errores

En la Figura 29 se muestra en consola lo que está sucediendo con el API mientras se corre el test. En la imagen se ve el motivo por el cual falló un test. Aparece el método al cual se le estaba aplicando el test. El tipo de excepción que lanzó ejecutando el método y la razón del fallo. Esto facilita la búsqueda del error y encontrar la solución de manera más efectiva.

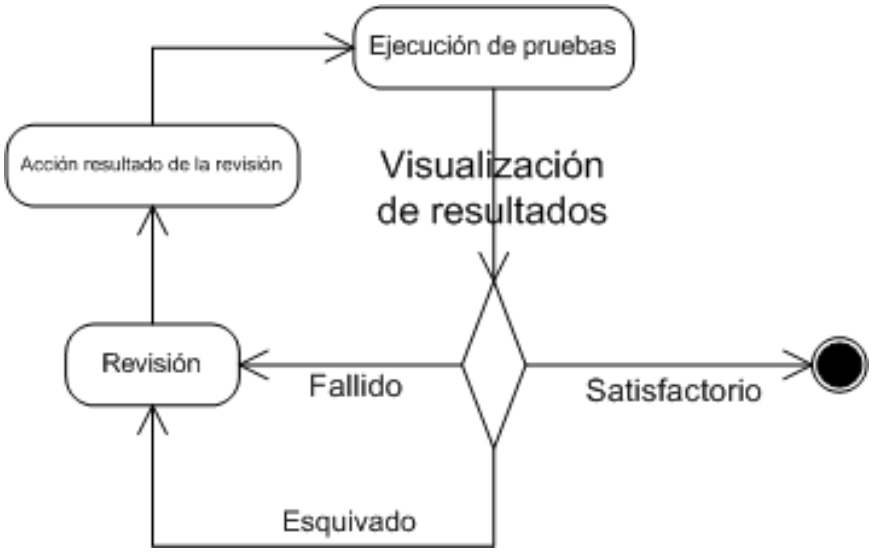
6.4.6 Control sobre las pruebas. Al ejecutar las pruebas realizadas para un API podemos ver la visualización de los resultados, y con estos se pueden sacar conclusiones sobre el estado de las pruebas, al ver los resultados se puede ver cuales pruebas han resultados con errores, cuales siguen con un resultado satisfactorio y cuales pruebas han sido esquivadas.

Se procede a revisar porque hay pruebas que fallaron, las cuales tienen prioridad, ya que estas pruebas habían resultados satisfactorias y ahora está fallando, al encontrar la causa de la falla se busca una respuesta del porqué sucedió esta falla para procurar no cometerla en el futuro, se debe hacer el respectivo arreglo y se ejecutan las pruebas de nuevo para verificar que todo está marchando correctamente.

Luego se procede a revisar las pruebas que se esquivaron, y pueden ser dos causas, una es que no hay suficientes datos en la base de datos para garantizar que esta prueba sea satisfactoria, por esta razón se decide saltar la prueba para evitar registros de pruebas fallidas cuando en realidad es por otra causa. La otra causa es que para un método en particular las pruebas no se han realizado, entonces en los resultados se visualiza como esquivado, acompañado con un mensaje el cual muestra que se han implementado casos de prueba para ese método.

Con esto se puede tener un control inmediato sobre las pruebas a los métodos, para llevar un control con respecto a registro anteriores se deben hacer comparaciones sobre resultados antiguos y llevar una métrica sobre cuales métodos presentan más problemas, cuales métodos aún no se les ha implementado casos de prueba y a cuales métodos no se les ha implementado un arreglo para un caso en particular.

Figura 31. Flujo control de pruebas



CONCLUSIÓNES

Se encontraron dos tendencias al hacer la automatización de las pruebas: pruebas por método (para probar la funcionalidad del método y sus variaciones) o hacer pruebas con varios métodos (para probar el rendimiento de los métodos en conjunto o un caso específico donde están involucrados varios métodos).

Durante la práctica se notó que la herramienta testNg, permitía hacer test a nivel de API pero no era posible hacer un test a nivel del GUI (interfaz); es recomendable complementar estas pruebas (testCases de API) con test manuales, directamente sobre la aplicación.

Es recomendable tener las pruebas en un paquete diferente donde se encuentran las API's, por motivos de ordenamiento y planeación de estas, también para tener conocimiento de lo que se posee y tener un mayor control sobre las pruebas.

Este tipo de test, permite encontrar errores en etapas tempranas de desarrollo y esto significa una reducción de costos a futuro. Debido a que método creado, método que se verifica.

La automatización de las pruebas es una alternativa interesante para las empresas que quieren asegurar cierto nivel de calidad antes de cada liberación de versiones. Aportan tranquilidad al ajustar y mejorar las principales funcionalidades, ya que brindan información sobre los cambios realizados. Es relevante seleccionar la herramienta que mejor se adapte al producto. Es tan importante el costo de la herramienta, como el tiempo que requiere la generación de los "scripts o test Cases" de prueba.

BIBLIOGRAFÍA

1. Disponible en Internet: http://es.scribd.com/doc/18286606/Modelo-de-Desarrollo-Evolutivo Teoría sobre modelo de desarrollo evolutivo.

2. Disponible en Internet: http://es.wikipedia.org/wiki/ISO/IEC_9126 ISO 9126 Conceptos de calidad.

3. Disponible en Internet: http://testng.org/doc/documentation-main.html *Manual de TestNg.*

4. Disponible en Internet: http://www.myeclipseide.com/module-htmlpages-display-pid-7.html *Documentación sobre MyEclipse.*

5. Disponible en Internet: http://www.proactiva-calidad.com/java/jdbc/index.html *Conexión JDBC y sentencias SQL.*

6. Disponible en Internet: http://www.solomanuales.org/manuales_oracle_10g-manuall216216.htm *Tutorial básico de Oracle.*

7. Disponible en Internet: http://www.taringa.net/posts/downloads/1158795/Libros-de-java-recomendados.html *Libro descargable de Java.*

8. Disponible en Internet: http://wwww.wikilearning.com/tutorial/tutorial_de_java/3938 *Tutorial básico de java.*

9. HOULETTE, FORREST FUNDAMENTOS DE SQL MCGRAW-HILL 2003.