Revisión de Código para Calidad de Desarrollo

Óscar Manuel Chávez Zúniga
Universidad Tecnológica Centroamericana, Tegucigalpa, Honduras, ochavez@tigo.com.hn

Profesor Mentor:

Carlos Roberto Arias Arévalo
Universidad Tecnológica Centroamericana, Tegucigalpa, Honduras, cariasa@unitec.edu

ABSTRACT

This paper presents the development of a project that has the goal to implement a system that automates the revision and verification of source code of the development process to determine their quality before they are loaded into the production environment, thus reducing the known problems that arise due to the failure to comply to stablished policy and standards. This Project was developed for an Information Technology department that currently handles more tha 60 software requirements per month, coming from the different information systems platforms and from business inteligence. A high percentage of the problems are detected and corrected after the software was been implemented and run live. For these reasons a solution was made that automatically checks and verifies the source code, and controls the movement of the programs from development to production. The solution has three steps: automatic revision of the code to check programming standards and programming policy, revision of performance to check that the program complies with acceptable performance, and migration of the programs to production.

Keywords: Software Quality Assurance, Automatic Code Revision

RESUMEN

Este artículo presenta el desarrollo de un proyecto que tiene como propósito implementar un sistema que automatice la revisión del código fuente de los programas en el proceso de desarrollo para determinar su calidad antes de ser puestos en producción y de esta forma poder reducir la incidencia de problemas conocidos provocados por el incumplimiento de los estándares y políticas establecidas. El proyecto fue desarrollado para el departamento de información, el cual desarrolla un promedio de 60 requerimientos de software mensuales entre creaciones y modificaciones para las plataformas de información e inteligencia de negocios. Un porcentaje alto de los problemas se detectan y corrigen después de que los programas entran en producción. Por lo tanto, se desarrolló un sistema para la revisión automática de código de programación y el control del proceso de paso a producción. El programa tiene tres procesos principales: revisión automática del código para determinar si el programa cumple con los estándares y políticas de programación establecidas, revisión de parámetros que determinan si los programas van a tener un rendimiento aceptable una vez que se coloquen en producción, y el tercero crea los programas de forma automática en el ambiente de producción.

Palabras claves: Calidad de Software, Revisión Automática

1. INTRODUCCIÓN

Cada día son más las empresas que se preocupan por la calidad del software que desarrollan o adquieren para sus productos o servicios, sin embargo en nuestro país no se ha desarrollado una cultura por la calidad de software, es difícil encontrar en nuestro medio empresas que se especializan en esta área, no obstante es evidente la necesidad

de implementar procesos que garanticen un buen funcionamiento del software antes de que estos sean colocados en producción. Implementar un proceso de calidad de software no resulta sencillo para todas las empresas que lo necesitan, ya que es necesario invertir en tiempo y recursos para realizar las diferentes tareas que este proceso demanda. Es por eso que este proyecto se enfoca en herramientas que ayudan a la automatización de estas tareas, asegurando un grado de calidad, disminuyendo los tiempos de revisión y eliminando tareas operativas del proceso cuando se hace de forma manual. De esta forma es más sencillo para una empresa implementar procesos sostenibles orientados a la calidad de software.

2. MARCO TEÓRICO

A continuación se presentara información del departamento para el cual fue desarrollado el sistema, este fue creado en el año 2002 como una iniciativa de la empresa para tener un sistema de información centralizado y así poder unificar los diferentes criterios y conceptos de información. El sistema cuenta con más de 500 procesos de extracción, transformación y carga de datos (ETL), 1,800 procesos de PL/SQL que se encargan del llenado de información al modelo de datos, 250 procesos para cubos de información y más de 50 página web que contienen información relevante para la toma de decisiones (Dashboard). El departamento recibe una cantidad significativa de requerimientos mensuales entre creaciones, modificaciones y mejoras para el modelo de datos, y demás servicios prestados. Cabe mencionar que la base de datos tiene más de 18,000 objetos y mide más de 36 Terabyte (TB).

Existen varios problemas que son de gran preocupación para el departamento y que se han intensificado por el aumento de información y cambios provocado por el crecimiento de los productos y servicios que ofrece la empresa. Entre estos problemas está el incumplimiento de estándares e implementación de mejores prácticas enfocadas a prevenir problemas de integridad de información y rendimiento en la base de datos. Este último afecta el tiempo de entrega de información y se ha convertido en un problema cíclico, ya que periódicamente se están analizando y modificando los programas para mejorar el rendimiento, pero al mismo tiempo se colocan programas en producción que terminan afectando de nuevo el rendimiento del sistema. El departamento tiene un acuerdo de nivel de servicio (SLA), donde especifica que debe tener actualizada toda la información a una hora específica del día. Se descubrió que el sistema solo puede mantener este tiempo por un período de tres meses, luego se tiene que intervenir tiempo y dinero para restablecer el tiempo acordado. Un estudio realizado este año, indicó que más del 70% de los problemas de rendimiento se pudieron haber evitado si los programadores hubieran aplicado en sus programas una serie de instrucciones que se crearon a partir de una base de datos de conocimiento y que contiene las mejores prácticas para evitar los problemas que más afectan al sistema. Se ha intentado en varias ocasiones implementar un proceso formal y estricto que contenga una serie de controles para el desarrollo, pruebas e implementación de los procesos, sin embargo no se ha logrado mantener de forma constante, ya que los tiempos y costos en el desarrollo se incrementan de forma significativa por la cantidad de requerimientos.

2.1 CONTROL DE CALIDAD DE SOFTWARE

Primero hay que entender el concepto del proceso de control de calidad de software, de acuerdo a (Dorado & Sanz, 2000), "el control de calidad de Software es el conjunto de técnicas y actividades de carácter operativo, utilizadas para verificar los requisitos relativos a la calidad del producto y del servicio". Para realizar este proceso se debe conformar un equipo de personas para crear un área de aseguramiento de la calidad, este equipo debe de llevar a cabo una serie de actividades:

- Verificación: Consiste en revisiones formales para auditorias sobre los cambios de configuración y con respecto a la calidad del producto.
- Validación: Validar a todos los niveles, incluyendo las fases de prueba como parte de la administración de cambios para el control de los productos.

- Medición de Software: En esta actividad es necesario establecer los objetivos de medición y las métricas asociadas a los mismos.

Existen estándares para la implementación de procesos de calidad de software, uno de ellos es la norma ISO-25000 (SQuaRE), que proporciona una guía para el uso de las nuevas series de estándares internacionales, llamadas Requisitos y Evaluación de Calidad de Productos de Software. Es la unión de dos normas: la ISO 9126 y la ISO 14598. El objetivo de esta norma es guiar el desarrollo de los productos de software con la especificación y evaluación de requisitos de calidad. Establece criterios para la especificación de requisitos de calidad de productos de software, sus métricas y su evaluación.

2.2 PROCESADORES DE LENGUAJES

Los módulos de revisión de código y de rendimiento están basados en el concepto de procesadores de lenguajes, es por eso que a continuación se explican brevemente algunos conceptos claves.

(Aho, 2008), introduce el concepto de compilador explicando que es un programa que recibe como entrada el código fuente de un programa escrito en un lenguaje de programación determinado y lo traduce en otro programa equivalente pero en otro lenguaje, este último es llamado lenguaje destino. Una de las funciones principales del compilador es detectar cualquier error que se dé durante la traducción. (Vanegas, 2006) nos explica las primeras tres fases de un compilador, el analizador léxico, el analizador sintáctico y el analizador semántico.

El analizador léxico (explorador), es la primera fase del compilador y se encarga de leer el código fuente del programa de entrada, por cada carácter que lee construye una serie de entidades primarias llamadas tokens. En resumen el analizador léxico convierte el programa de entrada en unidades lexicográficas. El analizador sintáctico, es la segunda etapa del compilador y se encarga de comprobar que todas las sentencias del código fuente estén escritas de forma correcta en concordancia con las reglas gramaticales del lenguaje. El analizador sintáctico recibe como entrada los tokens generados por el analizador léxico y los utiliza para generar una interpretación intermedia en forma de árbol. En esta etapa, así como en las demás, se van mostrando los errores que se detectan. La siguiente fase es el análisis semántico que comprueba la consistencia semántica del código fuente. Las sentencias pueden estar escritas sintácticamente correctas pero pueden no tener sentido según las reglas del lenguaje. El analizador semántico toma como entrada el árbol sintáctico y la tabla de símbolos para realizar el proceso. La tabla de símbolos es la estructura donde se almacena la información acerca del ámbito, enlace, tipo de dato e información relevante de los nombres de las variables, funciones y procedimientos. La tabla de símbolos se consulta cada vez que se encuentra un nombre en el código fuente, si se descubre un nombre que no está en la tabla o una nueva información sobre un nombre ya existente, se produce una actualización en la tabla.

Cuando se construye la gramática de un lenguaje de programación hay que tener mucho cuidado de no diseñar gramáticas que generen ambigüedad, (Aho, 2008) explica que una gramática es ambigua cuando está genera más de un árbol de análisis sintáctico para una cadena dada de componentes léxicos.

2.3 HERRAMIENTAS DE RENDIMIENTO PARA ORACLE

(Navarro, 2009) Comenta que Oracle actualmente maneja dos métodos de optimización, el Optimizador Basado en Costos (CBO) y el Optimizador Basado en Reglas (RBO):

- El optimizador basado en reglas es un método que se basa en reglas obtenidas heurísticamente, estas se actualizan en base a las experiencias adquiridas en versiones anteriores.
- El optimizador basado en costos selecciona un plan de ejecución basándose en las estadísticas recolectadas por la base de datos de las tablas e índices involucrados en la instrucción SQL que se está analizando.

(Rovedo, 2009), comenta que el optimizador basado en reglas (RBO) está quedando obsoleto debido a que se depuso su actualización en 1992. El optimizador basado en costos (CBO) fue haciéndose más sofisticado y se considera por muchos en la actualidad una de las piezas de software más complejas que existe, en sus primeras versiones se encontraron algunos problemas, pero eso fue cambiando posteriormente. Uno de los motivos más frecuentes de que el optimizador de costos no escogiese el plan ideal era la falta o desactualización de estadísticas de los objetos involucrados en las sentencias, en las versiones actuales Oracle incorporó la automatización de la recolección de estadísticas mediante las ventanas de mantenimiento, las cuales se activan por defecto. Explain Plan es la herramienta de Oracle que se utiliza para retornar el plan de ejecución de una sentencia SQL que es ejecutada en ese momento. (García, 2008)

3. METODOLOGÍA

El proyecto fue desarrollado en un período de 5 meses, para el desarrollo se utilizó el método de prototipo, donde se construyeron las pantallas con sus respectivas entradas y salidas, también se programaron funcionalidades como búsquedas, procesos del flujo del sistema, modelo de datos, procedimiento y funciones. Las funcionalidades de la revisión de código para los estándares y la revisión del rendimiento se desarrollaron posteriores al desarrollo del prototipo, ya que esto tomó un tiempo considerable del desarrollo, en general la metodología que se utilizó se resume en 5 pasos:

1) Definición del Requerimiento: primero se trabajó en la etapa del análisis la cual tenía como propósito la elaboración del requerimiento del sistema, el cual reduce la brecha de lo que el usuario requiere y lo que el programador entiende. Se trabajó con personal clave del equipo para definir el documento, se identificaron las salidas, entradas y procesos. Esta fase fue indispensable para comenzar el desarrollo prototipo.
2) Desarrollo del modelo de trabajo: Se elaboró un plan de trabajo con las tareas principales, se definió su duración, fecha de inicio y fecha de finalización. En esta etapa se comenzó a diseñar y construir las pantallas del sistema, así como algunas funcionalidades.
3) Utilización del prototipo: En esta etapa el desarrollo del prototipo se encuentra en una etapa avanzada, se acordaron reuniones para revisar los resultados con los usuarios claves y se identificaron los cambios. Esta reunión fue clave, ya que se identificaron cambios importantes en el módulo de rendimiento.
4) Revisión del Prototipo: Una vez que se terminó el desarrollo del prototipo se concertó una reunión para evaluarlo. En esta etapa los cambios fueron más de forma y no de funcionalidad.
5) Proceso Final: Se incorporaron al prototipo los programas de revisión de código y revisión de rendimiento para continuar el ciclo de evaluación con los usuarios. Surgieron algunos cambios importantes en los módulos de aprobaciones y puesta a producción, se implementaron los cambios y se finalizó el desarrollo.

Se realizaron dos tipos de pruebas para determinar la funcionalidad de sistema:

1) Las pruebas técnicas: estas pruebas se realizaron por el programador durante y al finalizar el desarrollo de cada módulo.
2) Pruebas de Usuario: estas fueron realizadas por el programador y los usuarios, en cada etapa de revisión y una final al terminar el desarrollo.

Es interesante mencionar que cada procedimiento propio del sistema desarrollado en PL/SQL fue revisado por el mismo sistema para determinar si cumplían con revisiones de código y rendimiento. Con esto podemos decir que el sistema pasó por un control de calidad de código.

Para el desarrollo de este proyecto se utilizaron diferentes lenguajes y herramientas de programación:

- Apex (*Oracle Application Express*) V4.0: se utilizó para el desarrollo web.
- Oracle XE 11.2.0.2: se utilizó como base de datos principal.
- JFlex y Cup: se utilizó para desarrollar el analizador sintáctico y semántico para la revisión del código de programación.
- PL/SQL Oracle: se utilizó para crear los programas de que interactúan con la base de datos.
- Java script: se utilizó para algunas animaciones y validaciones de los procesos en las páginas web.
- Java 1.5: se utilizó para programar la detección de errores en la verificación de estándares y la interacción del analizador sintáctico con la base de datos Oracle.

En la Figura 1 se muestra la arquitectura del sistema con los principales componentes, como se puede observar todos los componentes del sistema quedaron dentro de la base de datos de Oracle.

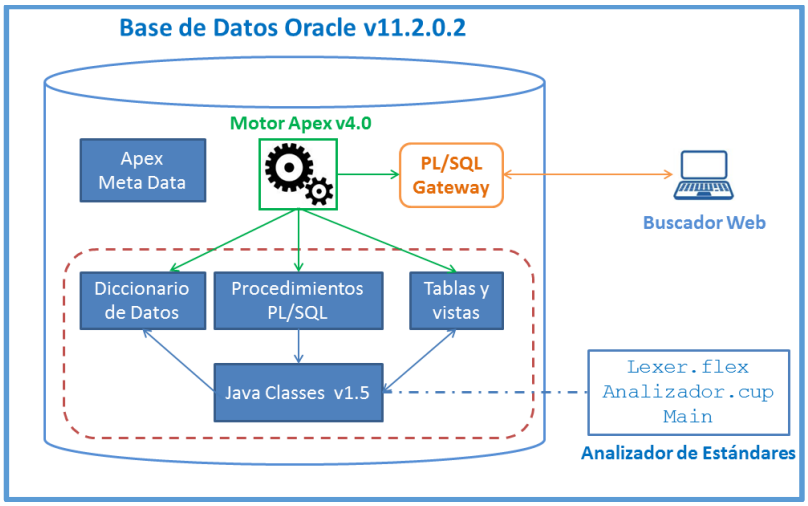

Figura 1: Arquitectura del sistema

Para poder tener una comprensión general del sistema, se presenta a continuación el diagrama de funcionalidad del sistema y se explica de forma general cada uno de sus componentes.

El sistema cuenta con una pantalla principal provista con un menú que contiene las siguientes opciones:

a) Control de Calidad: esta opción inicia el flujo de los procesos de verificación hasta llegar a la puesta a producción de los programas. Como se observa el diagrama este proceso tiene varios pasos que se describen a continuación:
   - Seleccionar Requerimiento: El analista programador selecciona el requerimiento que desea poner en producción.
   - Configuración: Se configuran los parámetros necesarios para iniciar el proceso, entre ellos está la Cantidad de Programas, que se refiere al número de programas a colocar en producción y están relacionados con el requerimiento seleccionado.
   - Obtener Código: Se seleccionan de una lista los programas que serán puestos en producción, la cantidad de programas seleccionados es la misma que el número de programas que se configuró en el paso anterior.
   - Verificar Código: En este paso se ejecuta el módulo de Verificación de Código y para cada programa se verifica cada uno de los estándares que se encuentran configurados, si todas las verificaciones son satisfactorias el sistema habilita el siguiente paso, de lo contrario el flujo se detiene.
   - Verificar Rendimiento: En este paso se ejecuta el módulo de Rendimiento de Código y para cada programa se verifican cada uno de los estándares que se encuentran configurados, si todas las verificaciones son satisfactorias el sistema habilita el siguiente paso de lo contrario el flujo se detiene.
   - Paso a Producción: Se indica el esquema de la base de datos en el que los objetos se pondrán en producción, luego este módulo crea o remplaza los programas en los esquemas indicados.

b) Procesos Pendientes: En esta opción del menú se muestra un listado de los procesos que no han sido finalizados exitosamente, bien sea porque no cumplieron con los requisitos o porque el usuario decidió salirse del proceso. El Analista programador después de revisar sus programas, puede seleccionar cualquier proceso para continuar el flujo y finalizarlo de forma exitosa.

c) Aprobaciones: Se muestran las aprobaciones rechazadas, aprobadas y pendientes. Las aprobaciones pendientes se seleccionan y se puede ver su detalle para decidir si se aprueba o se rechaza.

d) Administración (Mantenimiento): Esta opción del menú hace un llamado a la pantalla de mantenimiento donde se seleccionan otras opciones para cambiar los parámetros del sistema incluyendo la configuración de los estándares de rendimiento.

a) VERIFICACIÓN DE CÓDIGO Y RENDIMIENTO.

En la revisión de código fuente se utilizaron las herramientas JFlex y Cup, se creó el proyecto llamado `AnalizadorE`, este hace el análisis léxico y sintáctico para detectar si cumple con los estándares establecidos. El proyecto se cargó dentro de la base de datos y se ejecuta desde un procedimiento PL/SQL. En la Figura 3 se observa la interacción del `AnalizadorE` con las tablas y el diccionario de la base de datos.

La entrada del AnalizadorE es un código en PL/SQL ya compilado por la base de datos, si esto es así, entonces, ¿por qué se necesita revisar la gramática?, la respuesta es; porque es la forma en que el sistema detecta si los programadores colocaron el código de la forma que los estándares lo requieren. Para hacer esto el AnalizadorE no solo tiene que leer la gramática de cualquier PL/SQL sino que también debe de incorporar las reglas gramaticales de los estándares establecidos.

Una de las estrategias que se siguió para no definir toda la gramática del PL/SQL, fue definir en el analizador léxico solo aquellos *tokens* que son claves para implementar las reglas gramaticales para los estándares, esto quiere decir que algunas palabras reservadas se ven como si fueran un *token* `ID` (Identificador que no es palabra reservada) y en el analizador sintáctico se definieron bloques que consumen estos *tokens* que no siguen las reglas gramaticales del PL/SQL. A pesar de esta optimización se definieron más de 60 terminales y 35 no terminales.

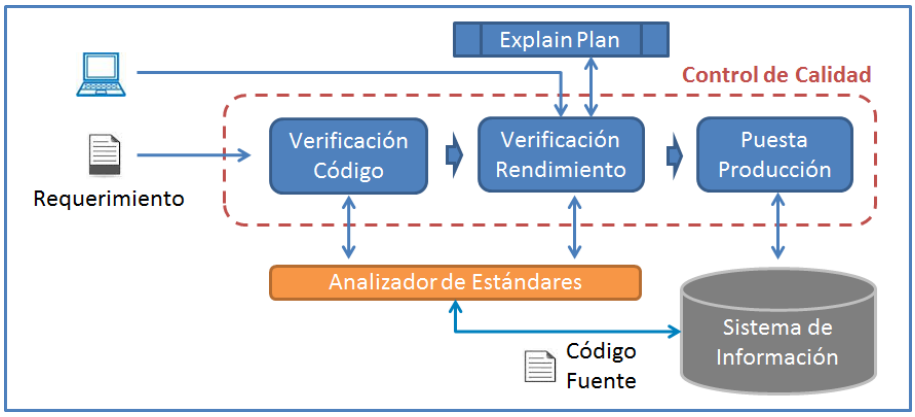

Figura 2: Proceso de verificación de código.

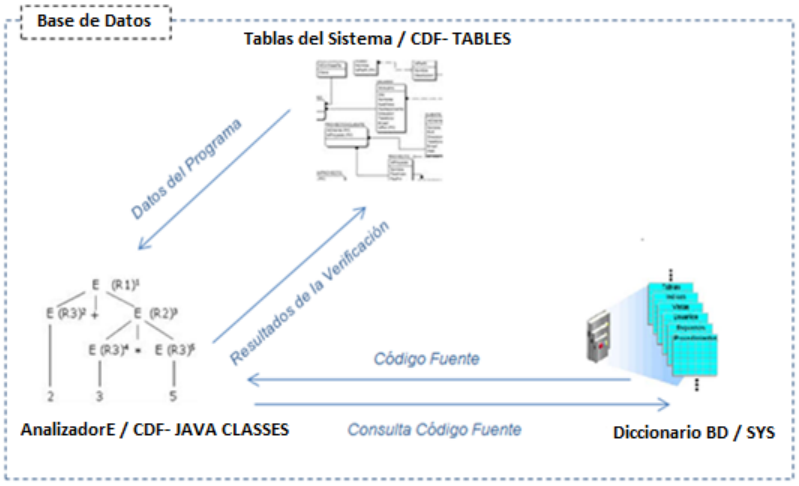

Figura 3: Interacción del AnalizadorE Dentro de la Base de Datos.

Para la revisión de código se definieron para esta etapa cuatro revisiones:

1) Estándar de verificación de funciones de registro: Esta verificación consiste en comprobar si el programador colocó la función `inicio()` después de la instrucción `Begin` y la función `Fin()` antes del sección de `Exception`. Estas funciones son las que registran el tiempo de ejecución del programa, cantidad de registros procesados, errores en tiempo de ejecución y el estatus de la ejecución, esta información es necesaria ya que es utilizada para el monitoreo de procesos. En el siguiente código se observa en la producción `Inicio` la captura del error en el caso que no se cumpla el estándar.

```
Pl      ::= Encabezado  DeclaracionVar  Funciones Main ;
Main    ::= BEGIN  FINDELINEA   Inicio  Bloque ;
Inicio  ::= INICIO  Tpara PUNTOCOMA FINDELINEA
        | {:E(3,"> ESTANDAR(1),
          Se espera la funcion [INICIO], despues del   BEGIN");
         Estandar("EST1", 1,"RECHAZADO" );  :}
```

Para la función fin se hizo algo similar en las producciones correspondiente.

2) Estándar verificación de variables: Se verifica que toda variable declarada dentro del programa comience con "V_", esto hace más legible el código ya que las variables se pueden confundir con los nombres de las tablas en las sentencias SQL. En la gramática se implementó en el momento que se declaran las variables, para las variables se definió un tipo especial de *token* que sigue la regla del estándar, esto se programó en el `lexer.flex`, que es el que realiza el análisis léxico.

```
ID = ({LETRA}|{GUIONBAJO})+ ({LETRA}|{DIGITO}|{PUNTO}|{GUIONBAJO})*
V = "V_"
VV = "v_"
VAR = ({V}|{VV}){ID}
```

Luego el `analizador.cup` verifica si los *tokens* de las variables declaradas son de tipo VAR, de no ser así, la variable declarada en el código fuente no cumple con el estándar y se reporta el error, como se muestra en el siguiente código.

```
DeclaracionVar ::=  |VAR:idv TipoDato
                    |ID:idv TipoDato
                   {: E2((idvleft + 1), (idvright + 1),"{ESTANDAR(2)-
                   VERIFICACION  DE VARIABLES},
                   La variable [" + idv.toString() + "]
                   no cumple con el Estandar ");
                    Estandar("EST2", 2,"RECHAZADO" );
```

3) Estándar Verificación del uso correcto del COMMIT: Se verifica que después de cada DML se coloque el comando `commit`, ya que si no se hace puede provocar errores de memoria debido a que se manejan volúmenes grandes de información.

   En la gramática se crearon dos producciones, una que contiene la ruta con la instrucción `commit` al final del no terminal `Sentencia` y otra igual sin la instrucción `commit`, esta última es la que detecta el error, ya que es la que no sigue el estándar. En el siguiente código se ejemplifica con la instrucción `insert`. Esto se hizo para todas las instrucciones de tipo DML.

```
Bloque ::= Sentencia Insert Sentencia COMMIT  Bloque
           |Sentencia Insert Bloque
         {:
          E(3,"Estandar(3), Falta el [COMMIT] del <INSERT>");
          MSG3 = MSG3.toString() + "> ESTANDAR(3)Error- Falta
          el [COMMIT] del <INSERT> \n";
          Estandar("EST3", 3,"RECHAZADO" );
         :}
```

4) Estándar de Verificación de Sección de Errores: Se verifican dos cosas, la primera que el programador coloque la sección de `exception` antes del `end`, y la segunda que dentro de la sección de excepciones se coloque la función `FinError` que es la que captura y registra los errores en tiempo de ejecución.

Después de la revisión de código el siguiente proceso es la verificación de rendimiento, la cual utiliza la herramienta Explain Plan de Oracle que determina la ruta de ejecución de una sentencia SQL para calcular su costo, basado en el método CBO. El Explain Plan no puede analizar programas PL/SQL ni sentencias SQL que contengan variables, es por eso que el AnalizadorE se encarga de extraer las sentencias DML o SQL y remplaza las variables existentes por valores fijos para que el Explain Plan pueda analizarlas. Se utiliza una tabla de símbolos para determinar el tipo de la variable y así poder remplazarla por valores fijos correspondientes, en la Figura 4 se muestra el proceso de Verificación de Rendimiento.
Como se observa en la Figura 4 el AnalizadorE separa las sentencias DML, esta tarea la realiza el `analizador.cup`, al encontrar una producción en la gramática de una sentencia DML se procede a grabar hasta que la sentencia finalice y así sucesivamente hasta que se encuentra el fin del programa.
El módulo de revisión de rendimiento realiza tres procesos para determinar si el programa es apto para colocarlo en producción, estos procesos son:

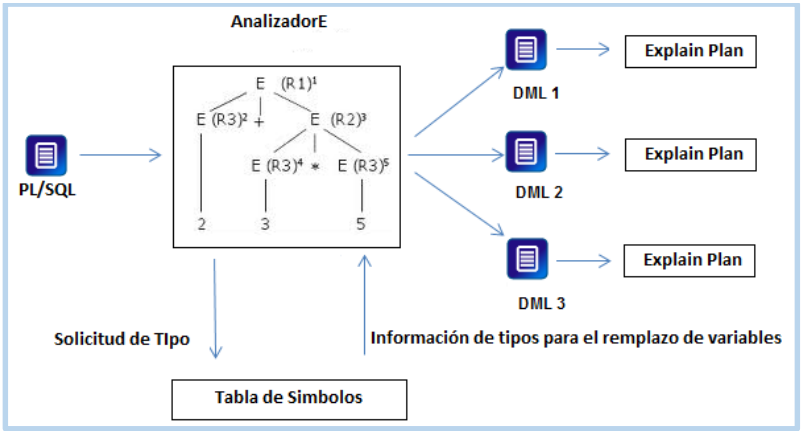

Figura 4: Proceso de Verificación de Rendimiento

1) **Análisis de Estadísticas CBO:** el primer paso es obtener las tablas relacionadas con cada sentencia DML de la tabla `plan_table` que tiene el resultado de la ejecución del comando Explain Plan y luego se busca cada tabla en el diccionario de la base de datos en la vista `sys.user_table` para obtener la última fecha de las

estadísticas, el siguiente paso es calcular los días que tiene el objeto desde su última actualización y este valor se compara con el campo `días_tolerancia` que pertenece a la tabla `fpp_estandar_rendimiento`, si el valor calculado es menor o igual al valor de la tabla, la verificación es exitosa (verificado) de lo contrario se marca como no exitosa (rechazado). Si la tabla nunca ha sido analizada entonces el campo de última fecha de análisis (`last_analyzed`) tendrá el valor de nulo (null), en este caso la verificación no es exitosa y se tiene que revisar si la tabla está incluida en el proceso de análisis o si este falló.

2) **El Análisis de Tiempo:** se calcula primero el factor producto que es la multiplicación del total de registros de cada tabla, este resultado se divide entre 1000 para reducir la cantidad de dígitos ya que el resultado pueden ser cantidades muy grandes. La cantidad de registros se obtiene de las estadísticas, es por eso que es importante que estas estén actualizadas, además que son necesarias para determinar un plan de ejecución óptimo. El valor del producto se compara con el rango de cantidad de registros, cuyos valores son parámetros que están almacenados en la tabla de estándares, de este resultado se obtiene un tiempo propuesto que es el tiempo máximo permitido, esto significa que el tiempo de ejecución no debe de ser mayor al tiempo propuesto, como se observa en la Figura 5. El tiempo ejecución se obtiene de forma automática de la última ejecución del programa en el periodo de pruebas.

Figura 5: Análisis de tiempo

3) **Análisis de Costo:** El proceso de este análisis es muy similar al análisis de tiempo ya que usa el mismo factor producto, pero en vez de analizar el tiempo se analiza el costo del plan de ejecución de cada sentencia DML, como se observa en la Figura 6.

Figura 6: Análisis de Costo

El costo del plan de ejecución se obtiene de la tabla `plan_table`, este se compara con el rango de cantidad de registros, cuyos valores son parámetros que están almacenados en la tabla de estándares, de este resultado se obtiene un valor de costo propuesto que es el costo máximo permitido, esto significa que el costo calculado por el plan de ejecución no debe de ser mayor al costo propuesto.

4. RESULTADOS

4.1 AUTOMATIZACIÓN DE LOS PROCESOS DE REVISIÓN.

Con la implementación del sistema se automatizaron las tareas de la revisión de código fuente para el cumplimiento de estándares y políticas de desarrollo de software, eliminando la carga de trabajo operativo que estas generan. Sin

esta automatización se necesita por lo menos una persona dedicada tiempo completo para la ejecución de estas tareas, ahora este tiempo se puede utilizar para otro tipo de funciones o actividades permitiendo incrementar la productividad del equipo. Actualmente con las revisiones manuales no se logra cumplir con todas las revisiones para los 60 requerimientos recibidos mensualmente, pero utilizando el sistema se puede lograr el 100% de las revisiones.

4.2 REDUCCIÓN DEL TIEMPO DE ENTREGA DE LA INFORMACIÓN.

El módulo de Revisión de Rendimiento ayuda a controlar el uso de recursos de cada programa antes de ser puestos en producción con el propósito de mantener y mejorar los tiempos de entrega de información. Este módulo también evita la acumulación de problemas de rendimiento provocada por los programas que se colocan producción.

4.3 CONTROL DEL PROCESO DE PASO A PRODUCCIÓN.

Al incorporar el proceso de paso a producción en el sistema se logra garantizar que todos los programas puestos en producción cumplen con todos los requisitos de calidad establecidos.

5. CONCLUSIONES

Se logró automatizar la revisión del código informático y de rendimiento para los programas desarrollados en PL/SQL, lo cual reduce el tiempo de ejecución del proceso y establece el 100% del cumplimiento del proceso.

Se logra reducir la incidencia de los problemas conocidos una vez que estos entren en producción, convirtiendo el proceso en una detección proactiva, esto gracias a la restricción de colocar los programas en producción solo cuando estos cumplen con la implementación de las mejores prácticas establecidas.

Al lograr ejecutar el 100% de las revisiones a través de la automatización de los procesos de revisión se logra incrementar la calidad de los desarrollos con respecto a la continuidad y el tiempo de entrega de la información.

REFERENCIAS

Condori , N. (2002). *Modelo de agregación basado en un sistema neurodifuso para un proceso de evaluación de calidad de software.*

Aho, A. V. (2008). *Compiladores principios, tecnicas y herraminetas.* Mexico: Pearson.

Antonio Calero, P. C. (2009). Vision Innovadorade la calidad del producto de software. *Revista Espanola de innovacion, calidad e ingenieria de software*, Vol.5 , No. 2, Pag. 49-56.

Dorado, J., & Sanz, L. (2000). *Medición para la Gestión de la Ingeniería de Software.*

Fillottrani, P. (2007). *Calidad en el Desarrollo de Software.* Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur.

García, C. A. (23 de julio de 2008). Herramientas de rendimiento. Espana.

Lorentz, D. (August 2010). *Oracle Database SQL Language Reference, 11g Release 1 (11.1).* Oracle.

Navarro, J. (2009). Optimizacion de Oracle SQL.

Pressman. (1992). *Software Engineering, a practitioner's approach (Third edition).* . McGraw-Hill.

Rovedo, P. (Martes de 3 de 2009). Clustering Factor. La Plata, Argentina.

Vanegas, C. A. (2006). *Compiladores: un enfoque.* Bogota.

Zamuriano, R. (oct. 2010). PROCESO DE INSPECCIÓN DE SOFTWARE ASEGURAMIENTO DE LA CALIDAD. *Journal Boliviano de Ciencias*, vol.7, no.21, Pag.10-16.

Authorization and Disclaimer

Authors authorize LACCEI to publish the paper in the conference proceedings. Neither LACCEI nor the editors are responsible either for the content or for the implications of what is expressed in the paper.