**Instituto Tecnológico de Costa Rica**

**Escuela de Ingeniería en Electrónica**


# Componentes Intel de Costa Rica
# PROTAF (Product Test Analysis Feedback)

**Informe de Proyecto de Graduación para optar por el Grado de Bachiller en Ingeniería Electrónica**

**Pedro Huguet Vaughan**

**Cartago, Junio del 2002**

# I. DEDICATORIA

A mis padres, por todo su apoyo y comprensión durante todos estos años de arduo trabajo.

A mi novia, con quien deseo pasar el resto de mi vida, por su amor y apoyo incondicional a esta causa.

# II. AGRADECIMIENTOS

A todas las personas en Componentes Intel Costa Rica que me ayudaron con la realización exitosa de este proyecto, entre ellos: Enrique Acuña, Humberto Ramírez, Allan Hernández, Carlos Ureña, Roman Sancho, Leonardo Calvo, Jorge Aguilar, Harold Campos, Luis Diego Rojas, Max Libby y a todos aquellos cuyos nombres no recuerdo pero que sin embargo pusieron su granito de arena en este proyecto.

A los ingenieros Pedro Murillo y Julio Córdoba por sus distintos aportes a este proyecto, y a los profesores que más me influenciaron en mi carrera Carlos Badilla y Arnoldo Rojas.

# III. ÍNDICE

# IV.ÍNDICE DE FIGURAS Y TABLAS

# CAPÍTULO 1 INTRODUCCIÓN

## 1.1 Descripción de la empresa

### 1.1.1 Descripción general

La corporación Intel®, así como su subsidiaria Componentes Intel® de Costa Rica S.A., tienen como misión hacer un excelente trabajo para sus clientes, empleados y accionistas, siendo el principal suplidor de plataformas computacionales para la economía mundial de Internet.

Sus valores principales son: la orientación al cliente, la orientación a resultados, la toma de riesgos, un gran lugar para trabajar, la calidad y la disciplina. Estos valores persiguen la obtención de sus objetivos que son: hacer de Intel® la principal plataforma computacional en Internet, crecimiento agresivo de nuevos negocios al mejorar sus capacidades y la obtención de una excelencia operacional.

La corporación Intel fue fundada en 1968 y es el líder mundial en la fabricación y desarrollo de microprocesadores. Actualmente Intel® cuenta con 80 000 empleados en más de 45 países alrededor del mundo. Sus principales consumidores son: distribuidores, creadores de equipo original para sistemas computacionales y periféricos, usuarios de computadoras personales quienes compran las mejoras para sus equipos fabricadas por Intel®, revendedores de equipos de comunicación y redes, así como un amplio rango de creadores de sistemas de transmisión industrial, entre otros.

Específicamente Componentes Intel® de Costa Rica S.A., localizado en la Ribera de Belén, cuenta con aproximadamente 2000 empleados y tiene como principal función el ensamble y prueba de microprocesadores, labor en la que ya cuenta con 5 años de experiencia en nuestro país. Desde sus inicios se ha convertido en el líder en la aplicación de los más altos estándares y políticas de medio ambiente, salud y seguridad ocupacional.

La planta en Costa Rica funciona hoy con niveles de eficiencia y productividad iguales o mayores a otras plantas similares, en donde la calidad del recurso humano ha jugado un papel clave en el éxito. Costa Rica es parte esencial de la estructura mundial de manufactura de Intel®, hasta el punto de ser el responsable de la fabricación de poco menos de un tercio de los procesadores Intel para computadoras personales, que se hacen y se envían directamente a clientes de Intel® desde Costa Rica.

### 1.1.2 Departamento donde se realizó el proyecto

A la hora de realizar pruebas, se deben definir estándares de calidad para determinar cuando un lote es razonablemente satisfactorio. Cuando no lo es, el lote debe ser detenido y probado con mayor rigurosidad. El departamento que define este tipo de estándares, desarrolla los programas de prueba para cada producto y que se encarga de ejecutar pruebas sobre los lotes es denominado Ingeniería de Producto. Este departamento necesita una herramienta que les permita obtener una retroalimentación detallada de los distintos errores que son comunes en la línea de producción. Dicha herramienta se le ha dado el nombre de PROTAF.

El departamento cuenta con los servicios de 60 personas en su mayoría ingenieros eléctricos y/o en electrónica aunque también laboran ingenieros en sistemas. Además cuenta con el apoyo de técnicos que en la actualidad estudian ingeniería. A la cabeza del departamento se encuentran los señores Andrés Salazar y Michael Mahler.

## 1.2 Definición del problema y su importancia

La herramienta que solicita este departamento es denominada PROTAF (Product Testing Analysis Feedback). Las fases finales del proceso de producción de los microprocesadores Intel® consisten en una etapa de ensamble, una de pruebas, y un proceso completo de análisis de los resultados de dichas pruebas. La figura 1.1 muestra el modelo PROTAF, el cual es la base de funcionamiento del departamento de Ingeniería de Producto. Las pruebas y el análisis generan mucha información que, actualmente, debe ser recolectada de manera manual. El objetivo primordial del sistema es automatizar esta labor, para obtener una retroalimentación de información rápida y confiable.

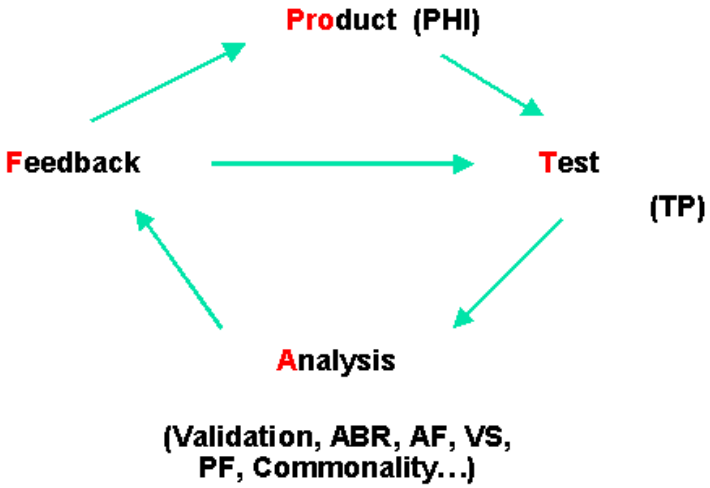



**Figura 1.1** Diagrama del modelo PROTAF

Los principales problemas que se pueden encontrar son la heterogeneidad de información y además, por la complejidad de los sistemas involucrados, la falta de consistencia en dicha información (algunos archivos de los que se extrae información contienen sintaxis difusas). Aunado a estos problemas, se presenta una amplia diversidad de plataformas en las que se encuentran sistemas basados en UNIX y Microsoft, además de una gran cantidad de clientes.

No solamente el departamento Ingeniería de Producto está interesado en este proyecto. También el departamento de calidad está interesado en información de alto nivel (datos estadísticos y mediciones paramétricas), que puede ser obtenida del repositorio de información. Actualmente, existen varios clientes de ambos departamentos, y durante el transcurso del proyecto se planea aumentar este número.

Como se tiene una enorme red de computadoras en Intel, el proyecto debe minimizar el impacto en los sistemas actuales, en cuanto a desempeño y nivel de procesamiento necesario. Además, el proceso de extracción de información que se ejecute directamente en los equipos del piso, debe ser lo más eficiente posible para así no causar saturación de procesamiento de los equipos y por tanto minimizar cualquier retraso en la producción del producto. Los estudios realizados muestran que el retraso máximo permitido en los equipos es de 0,5s.

En general, se desea extraer información de pruebas de distintas fuentes, consolidarla y, a partir de esto, generar reportes y consultas que permitan mejorar la productividad del sitio. La productividad será mejorada, dado que la información resultante permitirá detectar errores que ocurren tanto dentro como fuera de la planta, o sea en las plantas de fabricación de los componentes de los microprocesadores, permitiendo así modificar y mejorar los procesos existentes.

Esta información está centrada en la entidad LF (Límite de Fallas). Cuando un lote tiene cierta cantidad de fallas de un mismo tipo, se pone en espera y se prueba nuevamente por el departamento de ingeniería. Para cada tipo de falla, existe un límite de fallas, si la cantidad de fallas en el lote es inferior o superior a este límite (dependiendo del tipo de fallo) el mismo es puesto en espera. Un lote puede levantar varios LFs. Indistintamente del resultado del LF, las unidades dañadas se eliminan del proceso de producción siguiente para evitar que lleguen al cliente.

En la segunda etapa de PROTAF, que es la correspondiente a este proyecto de graduación, se comenzará con el análisis de la etapa de validación de

lotes (ver figura 1.1). Dicho análisis se centrará principalmente en el seguimiento de cada lote a través de todos los pasos en la llave de validación y los resultados que en estos pasos se obtengan. El principal problema para esta etapa está en que la información que se necesita para un control completo de las validaciones de lotes está diseminada en varios servidores y varios tipos de repositorios de datos. Principalmente se encuentran estructuras de directorios UNIX con archivos texto, aunque también se tienen bases Access y SQL Server.

## 1.3 Objetivos

### 1.3.1 Objetivo general

Automatizar el proceso de recolección y análisis de la información de validación de lotes mediante el uso de rutinas de software, para así reducir el tiempo de análisis y por tanto mejorar el rendimiento en la producción.

### 1.3.2 Objetivos específicos

1. Conocer el proceso de fabricación de microprocesadores para entender su relación con la validación de lotes.

2. Conocer el proceso de ensamblaje microprocesadores para entender su relación con la validación de lotes.

3. Diseñar una interfase de usuario, de forma tal que fomente su utilización.

4. Implementar a nueva interfase en el servidor de la aplicación.

5. Realizar pruebas con usuarios para corroborar la información que actualmente se encuentra disponible en la base de datos.

6. Evaluar los resultados obtenidos de las pruebas.

7. Determinar las distintas fuentes de información disponibles respecto a la validación de lotes.

8. Determinar los formatos de dichas fuentes de información.

9. Especificar la mejor forma de recolectar la información en los distintos formatos.

10. Diseñar las rutinas que obtengan la mayor cantidad de información de las fuentes disponibles de manera independiente.

11. Ejecutar pruebas a las rutinas de recolección antes descritas.

12. Evaluar los resultados de las pruebas a las rutinas de recolección.

13. Diseñar la(s) rutina(s) que permitan unir la información e incluirla en la base de datos SQL actual obtenida anteriormente de manera independiente.

14. Ejecutar pruebas a la(s) rutina(s) de unión.

15. Evaluar los resultados de las pruebas a la(s) rutina(s) de unión.

16. Determinar los análisis requeridos con la información obtenida.

17. Diseñar la interfase de usuario final que permite realizar los análisis antes descritos.

18. Ejecutar pruebas con la interfase de usuario final.

19. Evaluar las pruebas de la interfase.

20. Preparar a los usuarios para la utilización del nuevo PROTAF.

21. Instalar la nueva versión del programa en el servidor para comenzar con los protocolos de pruebas.

22. Ejecutar pruebas al sistema.

23. Evaluar los resultados de las pruebas al sistema.

24. Preparar la documentación para la transferencia del conocimiento tecnológico adquirido durante la ejecución del proyecto.

# CAPÍTULO 2 ANTECEDENTES

## 2.1 Estudio del problema a resolver

Dado su gran tamaño e implicaciones, PROTAF se dividió en varias etapas desde su conceptualización, para ser realizado por varios practicantes en un plazo de aproximadamente 2 años (siempre y cuando no se reestablezcan sus objetivos). Durante los meses de Julio a Diciembre del 2001 el estudiante Arnoldo Muller Molina de la carrera de Ingeniería en Computación del ITCR realizó la primera etapa del proyecto. Esta consistió en desarrollar los algoritmos en Perl, ASP y Visual Basic de forma tal que se pudiera realizar la extracción de la mayor cantidad de información de los archivos texto que generan los sistemas de pruebas en el piso de producción de la planta, con respecto a las pruebas de los lotes en la etapa TEST (ver figura 1.1).

Es importante recordar que, por ciertas normativas de confidencialidad, algunos de los nombres reales de los elementos han sido reemplazados y algunos elementos reales mencionados no son explicados en detalle.

Al finalizar la primera etapa del proyecto (Jul-Dic 2001) el sistema PROTAF estaba diseñado como se detalla a continuación: Las fuentes de información de PROTAF, se encuentran distribuidas en los sistemas UNIX; Existe un tipo de archivo primario que se llamará “X” y otro tipo de archivos a los que referencia “X”, que se llamarán “Y”; Al módulo que interactúa con esta arquitectura, se le denomina subsistema de caché. La segunda arquitectura está basada en sistemas Microsoft, y se le denomina el repositorio de información.

El área de sistemas UNIX se desarrolló en Perl. Se consideró la posibilidad de utilizar otros lenguajes en esta plataforma, pero dado que la manipulación de hileras es una actividad fundamental en este módulo, y Perl está diseñado para este tipo de operaciones, el lenguaje óptimo para este desarrollo fue Perl.

El área del repositorio de información consiste en una plataforma basada en Windows 2000 con servidor de web IIS y servidor de base de datos SQL Server

2000. La razón de la escogencia de estas plataformas está basada en el hecho de que la plataforma UNIX no poseía licencias de base de datos disponibles (al momento de diseño) y la plataforma Microsoft sí, esto tanto para el sistema operativo, servidor de web y motor de base de datos.

Una vez finalizada la primera etapa del proyecto se diseño e implementó un repositorio capaz de filtrar la información de manera compleja y que además tiene la capacidad de desplegar la información en un ambiente web. Además de esto, se le otorgó al sistema la capacidad de manejar usuarios, el manejo de una tabla de configuración del sistema, el almacenamiento de la información de desempeño que genera el subsistema de caché, y una tabla de mensajes del sistema en donde se puedan detectar los errores que encuentra el “parser” del repositorio de información. En esta tabla de mensajes, se pueden observar mensajes de error internos (para depurar el sistema), y mensajes de notificación, como por ejemplo, cuando un usuario ingresa al sistema. El repositorio de información también es capaz de generar reportes en Excel de cierto conjunto estadístico de la información del reporte.

El modelo del sistema en su primera etapa se observa en la figura 2.1.

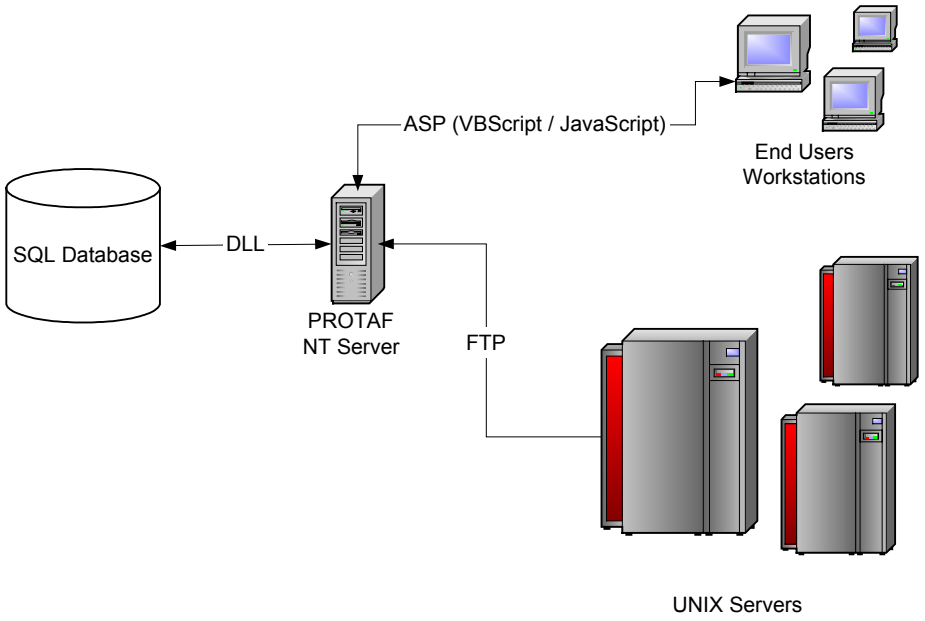



**Figura 2.1** Diagrama del sistema en su primera etapa

Es sobre este modelo que se basa la segunda etapa del sistema. Los primeros objetivos del proyecto se centran en mejorar la interfase que existe entre el servidor PROTAF y los usuarios finales. Para realizar este objetivo, principalmente se debe trabajar sobre el código ASP de la página web del sistema de manera tal que se fomente el uso del sistema como herramienta. Esto se logra diseñando una interfase mas amigable, confiable y eficaz para el usuario final.

Una vez concluidos los primeros objetivos anteriormente descritos, se logra corroborar la información en la base de datos para luego continuar con los siguientes objetivos. Estos se basan en obtener la información de las validaciones de lotes que fallaron las pruebas. A continuación se detalla cada una de las validaciones que debe de incluir el sistema una vez finalizada la segunda etapa.

### 2.1.1 Validaciones de producción

Estas validaciones son aquellas que se ejecutan como parte del proceso normal de producción. Como se explicó anteriormente, cuando un lote no cumple con los límites establecidos de fallas este debe ser probado nuevamente. En este caso, se realiza la validación del lote para comprobar que las fallas en realidad hayan sido causadas por las unidades y no por problemas con el equipo de prueba. Si se comprueba que las fallas son causadas por las unidades, entonces el lote se desecha. Si se comprueba que las fallas no son causadas por las unidades, ya sean errores del módulo de prueba, del manejador de los chips o del mismo operario, el lote se reintegra al proceso normal de producción. Las validaciones de producción siguen el mismo sistema que las pruebas de producción. Su única diferencia es que los resultados de las validaciones son guardados en rutas diferentes dentro de la jerarquía UNIX.

En la mayoría de los casos, las validaciones se realizan bajo las condiciones utilizadas en las pruebas de producción. Sin embargo, cuando existe una falla funcional en pruebas estructurales, la validación se realiza simulando las condiciones con que se prueban las unidades cuando son producidas en la fábrica. Para ello se utiliza un programa distinto que recalcula los límites de falla. Es por ello, que para estas validaciones se diseñaron dos programas: PQ.cgi y VFF.cgi. Siendo el último el programa que ejecuta.

### 2.1.2 Validaciones VFF

Es la prueba que se realiza a todas las unidades que fallan por pruebas funcionales en los módulos de prueba estructurales. El VFF trata de recrear las condiciones en las que las unidades fueron probadas en fábrica, para encontrar disparidades en las correlaciones entre ambas pruebas (Ensamblaje - Fábrica). El departamento de Q&R define los valores permitidos de unidades que presentan dicha disparidad en la correlación. VFF.cgi hace referencia a estas tablas para obtener así los valores actualizados.

### 2.1.3 Validaciones VS

Los experimentos de VS aclaran si las fallas que están en la etapa de QI son realmente inducidas por los hornos o son creadas en el proceso de ensamblaje de los microprocesadores. Por ello se realiza un QI en limpio, donde las unidades anteriormente han pasado por una prueba previa al horno y después se envían a la prueba. Por último se envían nuevamente a QI. La muestra la define el departamento de calidad, y ellos dicen cuando el experimento puede ser eliminado del proceso normal de producción para un producto dado.

### 2.1.4 Validaciones PF

Esta prueba se hace para un producto cuando este comienza su vida de producción y básicamente se realiza para cumplir con los estándares de Intel. El monitoreo tiene como intención eliminar las pruebas en frío que se realizan cuando un producto es inmaduro, esto es, cuando se está comenzando con la producción del mismo. La Eliminación de Prueba en Frío (EPF) es aquella que se realiza una vez que se han producido más de una cierta cantidad de unidades a través de PF, un grupo a nivel mundial decide si el PF se sigue corriendo o no para dicho producto.

### 2.1.5 Validaciones FC

Las FC's son las unidades que fallan el proceso de muestra de calidad. Intel separa del proceso normal de producción cierta cantidad de lotes que van listos para el cliente y ejecuta una prueba extra conocida como Comprobación de Calidad Final (CCF) o prueba de calidad final. En dicha prueba se espera que no falle ninguna unidad ya que estas unidades ya han sido probadas y están listas para enviar al cliente. Si alguna unidad falla se convierte en una FC. Básicamente este es un indicador de la salud y estabilidad del producto y de los programas de prueba utilizados en producción.

## 2.2 Requerimientos de la empresa

Es el objetivo final de la empresa tener un sistema estadístico con base de datos que funcione en un servidor de bajo nivel, cuyo propósito es realizar en forma automática la mayor cantidad de tareas del departamento de ingeniería de producto.

El sistema debe tener la menor cantidad de requerimientos para las computadoras que utiliza el usuario final, de modo que su implementación a nivel corporativo (tanto nacional como internacional) sea lo más sencilla posible para su aplicación, tanto en horas hombre como en presupuesto. Además, debe ser accesible utilizando la redes LAN y WAN de la empresa. El ingreso a la interfase del sistema debe contar con acceso de seguridad dado que los datos que se manejan en el sistema son en extremo confidenciales.

De manera específica, esta segunda etapa del sistema debe ser capaz de obtener la información de validaciones de los lotes que fallaron las pruebas de producción y desplegar los resultados. Esto lo puede realizar en tiempo real o obteniendo los datos de antemano y guardándolos en la base de datos SQL con que se cuenta. Al desplegar los datos, el sistema debe estar en la capacidad de recalcular los límites de falla para determinar si la prueba es exitosa, o sea se tiene una falla inválida o no; en este caso se tiene una falla válida. Además es necesario que despliegue los resultados detallados de la prueba para que el usuario tenga la posibilidad de realizar un análisis más profundo de los datos.

Es importante que todo aquel software que se utilice en el sistema debe contar con las licencias apropiadas. Es por ello que se prefiere la utilización de software que cuente con licencia corporativa, como es el caso con casi todo el software Microsoft para oficina y desarrollo. Todo aquel software para el cual se necesite obtener licencias, debe ser indispensable y se debe de corroborar la necesidad de realizar la inversión con datos viables.

## 2.3 Solución propuesta

### 2.3.1 Interfase de usuario final

Es necesario mejorar la interfase del usuario final del programa diseñada en la primera etapa del proyecto, dado que en la actualidad su uso es relativamente poco debido a la falta de entusiasmo de los usuarios. Es necesario realizar una serie de reuniones con la mayor cantidad de usuarios del sistema para evaluar los cambios que se consideren necesarios.

### 2.3.2 Pruebas de validez y correcciones

Al aumentar el nivel de uso del sistema se realizaran las pruebas de validez de la información utilizando la ayuda de los mismos usuarios para corroborar los resultados obtenidos a finales de la etapa anterior del proyecto en pruebas con ambiente controlado.

Durante las pruebas se harán los cambios necesarios, tanto en interfase como en código interno para asegurar una interacción usuario-sistema de alta productividad, así como para también resolver cualquier error que pueda existir cuando se extrae la información mediante el subsistema de caché al repositorio de datos.

### 2.3.3 Validación de lotes

Una vez que se ha mejorado la interfase y mientras se realizan las pruebas y validaciones correspondientes de la información que se obtiene a partir del subsistema de caché, se implementará el siguiente paso en el desarrollo de PROTAF. Esto incluirá la recolección y análisis de las validaciones que se le aplican a los lotes en piso cuando estos son clasificados como no aptos, con sus respectivos reportes y tablas para que los usuarios del departamento de Ingeniería de Producto tengan una herramienta más para realizar sus labores.

El sistema estará en la capacidad de analizar la información recolectada y además permitirá a los usuarios seguir paso a paso las validaciones que se le hicieron al lote, así como el resultado de dichas validaciones.

### 2.3.4 Pruebas finales

En la última etapa del proyecto se realizarán las pruebas en ambiente controlado del sistema para la validación de la información, antes de su instalación final en el servidor dedicado. Una vez realizada la instalación se procederá con las pruebas del sistema en caliente, comparando la información del sistema con la que se obtiene de otras fuentes de información utilizadas en la actualidad. Las pruebas en caliente serán documentadas para una posterior aprobación para su uso como herramienta de producción. Sin embargo este objetivo está fuera del alcance de este proyecto de graduación.

# CAPÍTULO 3 METODOLOGÍA

1. Conocer el proceso de fabricación de microprocesadores para entender su relación con la validación de lotes.

    a. Tomar los cursos interactivos de que dispone la empresa en la Intranet (5 días)

    b. Asistir a cursos magistrales (3 días)

2. Conocer el proceso de ensamblaje microprocesadores para entender su relación con la validación de lotes.

    a. Tomar los cursos interactivos de que dispone la empresa en la Intranet (5 días)

    b. Asistir a cursos magistrales (3 días)

    c. Realizar tour guiado del piso de producción. (1 día)

    d. Programar reuniones con encargados de piso para obtener una explicación más detallada del proceso paso a paso. (4 días)

3. Diseñar una nueva interfase de usuario, de forma tal que fomente su utilización.

    a. Programar reuniones con los usuarios actuales del sistema para determinar fallas, observaciones y sugerencias. (4 días)

    b. Escribir el código del sitio web que utiliza el sistema como interfase. (10 días)

4. Implementar a nueva interfase en el servidor de la aplicación.

    a. Diseñar un protocolo de pruebas para la interfase. (1 día)

    b. Ejecutar y evaluar los resultados de las pruebas. (3 días)

c. En caso de que los resultados sean positivos instalar la nueva interfase en el servidor. (2 días)

5. Realizar pruebas con usuarios para corroborar la información que actualmente se encuentra disponible en la base de datos.

   a. Programar reunión con los usuarios del sistema para explicar el protocolo de pruebas en caliente del sistema. (4 días)

   b. Recibir observaciones y sugerencias de los usuarios para corregir y mejorar la interfase y/o el sistema de caché. (5 días)

6. Evaluar los resultados obtenidos de las pruebas.

   a. Determinar si el nivel de fallas observadas en el sistema es aceptable para continuar con el proyecto. (2 días)

7. Determinar las distintas fuentes de información disponibles.

   a. Investigar los diferentes requerimientos de la empresa con respecto a la localización de la información de validación. (3 días)

   b. Investigar y estudiar las distintas herramientas, tanto hardware como software, que se utilizan para la validación de lotes. (5 días)

   c. Programar reuniones con los encargados y/o supervisores de las distintas etapas de validación. (4 días)

8. Determinar los formatos de dichas fuentes de información.

   a. Investigar los diferentes requerimientos de la empresa con respecto al formato de la información de validación. (3 días)

   b. Investigar las distintas fuentes de información obtenidas anteriormente para determinar el formato de cada una de ellas. (3 días)

9. Especificar la mejor forma de recolectar la información en los distintos formatos.

    a. Evaluar las fuentes de información para determinar la conectividad con que se cuenta. (2 días)

    b. Investigar los tipos de licencia con que cuenta la empresa para la conectividad de datos. (3 días)

    c. Seleccionar la conectividad que tenga la mejor relación gastos versus nivel de procesamiento necesario. (2 día)

10. Diseñar las rutinas que obtengan la mayor cantidad de información de las fuentes disponibles de manera independiente.

    a. Evaluar el formato de las fuentes de información individualmente. (2 días)

    b. Evaluar el formato necesario para la exportación de la información a la base de datos del repositorio de información. (3 días)

    c. Diseñar el algoritmo de barrido y filtrado de los archivos de información (5 días)

    d. Escribir las rutinas de recolección. (10 días)

11. Ejecutar pruebas a las rutinas de recolección antes descritas.

    a. Diseñar un protocolo de pruebas para las rutinas. (1 día)

    b. Generar copias de trabajo de las fuentes de información para no dañar información sensitiva. (1 día)

    c. Ejecutar el protocolo de pruebas. (3 días)

12. Evaluar los resultados de las pruebas a las rutinas de recolección.

a. Comparar la información obtenida con la propuesta. (1 día)

b. Determinar si las rutinas cumplen con los requerimientos establecidos. (1 día)

13. Diseñar la(s) rutina(s) que permita(n) unir la información obtenida anteriormente de manera independiente e incluirla en la base de datos SQL actual.

    a. Evaluar el formato de las nuevas fuentes de información intermedias. (2 días)

    b. Evaluar el formato, la disposición y diseño de las tablas que contendrán la información en la base de datos del repositorio de información. (3 días)

    c. Escribir las rutinas de unión. (10 días)

14. Ejecutar pruebas a la(s) rutina(s) de unión.

    a. Diseñar un protocolo de pruebas para las rutinas. (1 día)

    b. Generar copias del repositorio de datos para no dañar información sensitiva. (1 día)

    c. Ejecutar el protocolo de pruebas. (3 días)

15. Evaluar los resultados de las pruebas a la(s) rutina(s) de unión.

    a. Comparar la información obtenida con la propuesta. (1 día)

    b. Determinar si las rutinas cumplen con los requerimientos establecidos. (1 día)

16. Determinar los análisis que se requieren en base a la información obtenida.

a. Programar reuniones con los encargados y/o supervisores de validar los lotes. (4 días)

b. Programar reuniones con las personas encargadas del análisis de fallas. (4 días)

c. Programar reuniones con otros departamentos, en especial el departamento de calidad, para determinar sus requerimientos. (5 días)

17. Diseñar la interfase de usuario final que permite realizar los análisis antes descritos.

    a. Programar reuniones con los usuarios del sistema para determinar la mejor manera de presentar la información. (3 días)

    b. Diseñar las páginas web para el despliegue de la información. (5 días)

    c. Escribir el código fuente del sitio web. (10 días)

18. Ejecutar pruebas con la interfase de usuario final.

    a. Diseñar un protocolo de pruebas para la interfase de usuario. (1 día)

    b. Generar copias del repositorio de datos para no dañar información sensitiva. (1 día)

    c. Ejecutar el protocolo de pruebas. (3 días)

19. Evaluar las pruebas de la interfase.

    a. Comparar el funcionamiento de la interfase obtenida con la propuesta. (1 día)

    b. Determinar si la interfase cumple con los requerimientos establecidos. (1 día)

20. Preparar a los usuarios para la utilización del nuevo PROTAF

    a. Diseñar y escribir los manuales de usuario final, técnico y de desarrollo del sistema. (5 días)

    b. Diseñar y escribir las presentaciones para usuario final y para administradores del sistema. (3 días)

21. Instalar la nueva versión del programa en el servidor para comenzar con los protocolos de pruebas.

    a. Obtener aprobación para instalar la nueva versión del sistema en los servidores web y UNIX. (3 días)

    b. Determinar el protocolo de migración al nuevo sistema para evitar la pérdida de información. (1 día)

    c. Realizar respaldos de la información del repositorio de datos del sistema actual. (1 día)

    d. Instalar la nueva versión de PROTAF para su uso. (2 días)

22. Ejecutar pruebas al sistema en caliente.

    a. Diseñar el protocolo de pruebas finales del sistema. (1 día)

    b. Ejecutar el protocolo de pruebas. (7 días)

23. Evaluar los resultados de las pruebas al sistema en caliente.

    a. Comparar los resultados de funcionamiento e información obtenida con la propuesta. (1 día)

    b. Determinar si el sistema cumple con los requerimientos establecidos. (3 días)

24. Preparar la documentación para la transferencia del conocimiento tecnológico adquirido durante la ejecución del proyecto. (3 días)

# CAPÍTULO 4 SOFTWARE DEL SISTEMA

Una vez concluido el proyecto, el modelo que se observa en la figura 2.1 se modificó para mejorar la efectividad del sistema. De esta forma, se redujeron la capacidad de procesamiento y capacidad en disco requeridas para servidor NT y con esto el presupuesto para su compra. Además, la carga de usuarios se divide entre el servidor NT, la base de datos SQL y los servidores UNIX de producción. El nuevo modelo del sistema se muestra en la figura 4.1.

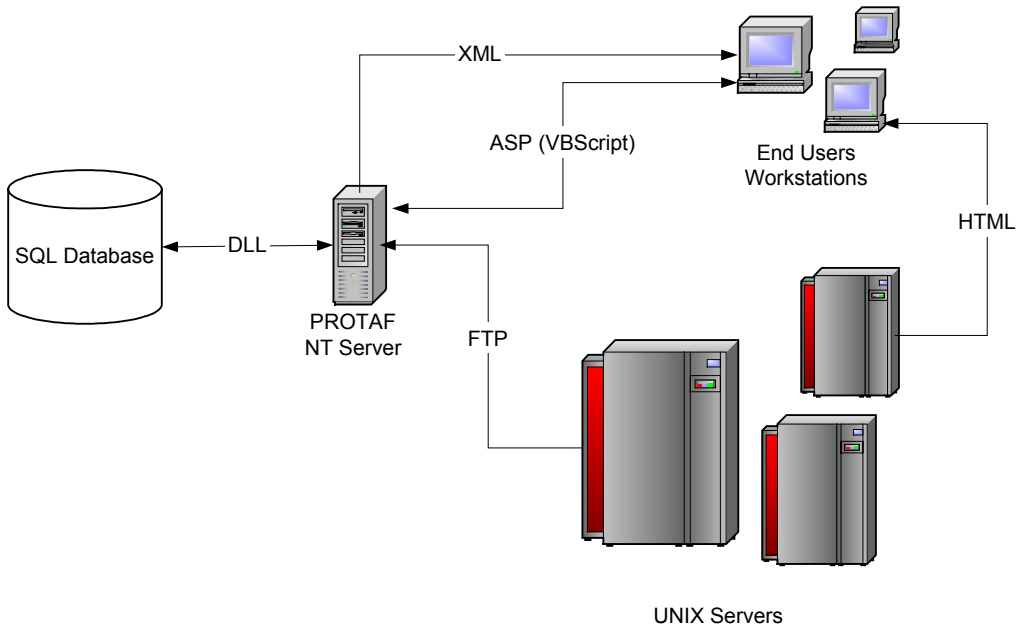



**Figura 4.1** Diagrama del sistema en su segunda etapa

También en la figura **4.1** se puede observar que el código de páginas web se consolidó para utilizar un solo lenguaje. Esto simplifica su mantenimiento y el entrenamiento de próximos desarrolladores y administradores del sistema. En los apartados siguientes se procede a definir y explicar cada una de las interfases y códigos diseñados en el sistema.

## 4.1 Base de datos

Como se mencionó en el capítulo 2 apartado 2.1, la base de datos o repositorio de información del sistema está basado en SQL Server. El diseño de la base de datos que desarrolló el compañero Arnoldo Muller en el último semestre del 2001 sufrió pocos cambios en esta segunda etapa del proyecto. Su definición se detalla en la figura **4.2**.

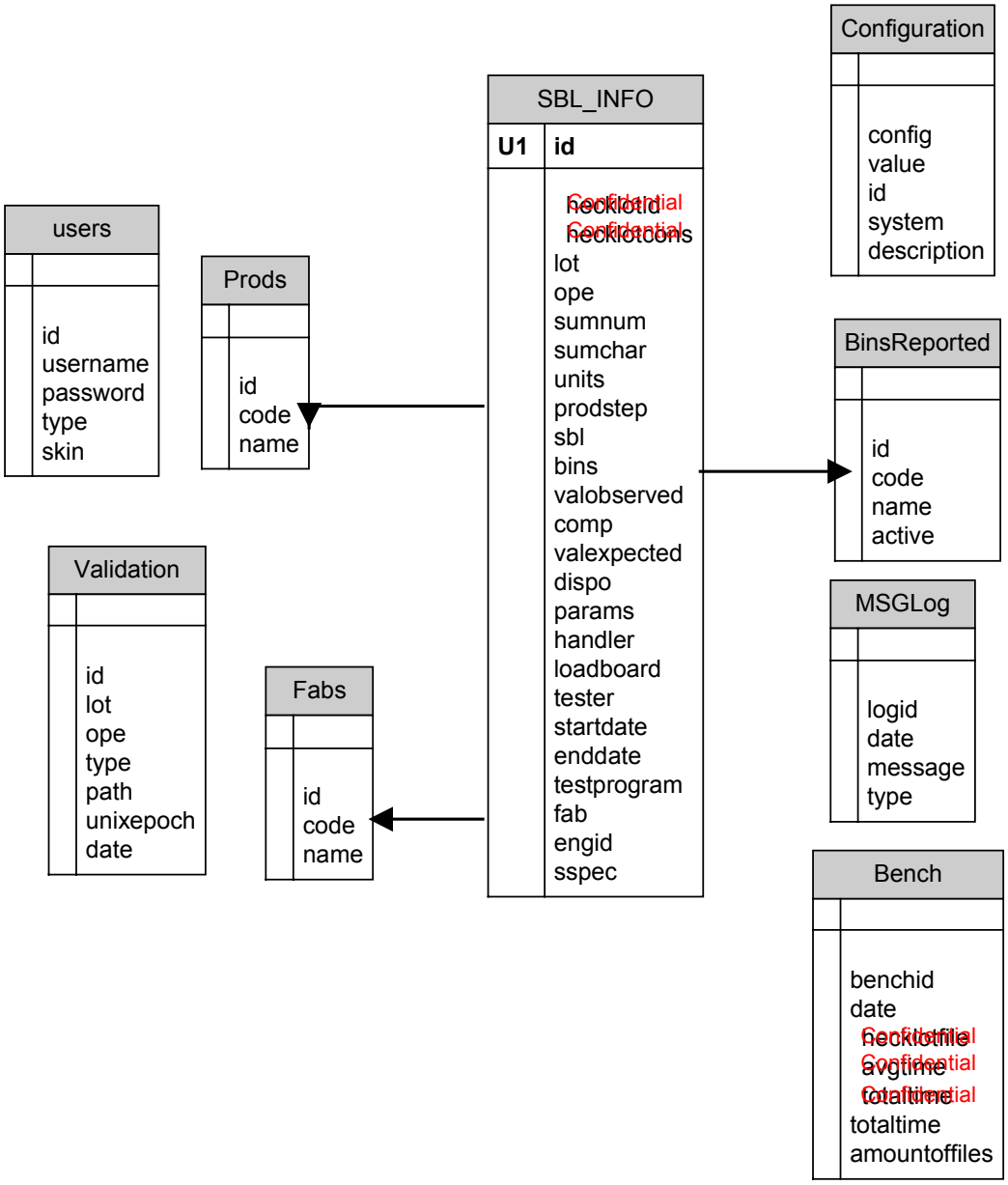



**Figura 4.2** Diseño de la base de datos SQL

Dado que las validaciones de los lotes son guardadas en los servidores UNIX durante un tiempo considerable y que el acceso a éstas no afecta de ninguna manera las computadoras de producción, no fue necesario diseñar los algoritmos para subir la información a la base de datos, en su lugar la información se obtiene en tiempo real. Esta es la principal razón por lo que la base de datos no sufrió cambios de consideración. Los únicos cambios fueron meramente para mejorar el funcionamiento existente del sistema, así como para incluir otros métodos con los que este no contaba.

Sin embargo, el DLL de acceso sí sufrió varios cambios dado que se incluyeron varios procedimientos con que no se contaban, entre ellos:

a. Utilidades de conversión entre distintos formatos de datos (SQL, XML, ...)

b. Utilidades para la creación de consultas SQL

c. Conexión remota con los servidores de producción para obtener información en tiempo real de los lotes.

d. Procedimientos internos de menor importancia

A continuación se ahonda un poco en los distintos procedimientos que se crearon.

### 4.1.1 Conversión de formatos

Estos procedimientos fueron creados principalmente por el hecho de que el lenguaje de programación utilizado en las páginas web, VBScript, no cuenta con convertidores de datos adecuados. Esto es debido a que dicho lenguaje no tiene tipos de datos. A modo de ejemplo se tiene el siguiente comando:

```
FormatXMLDate = format(strDate, "YYYY-MM-DDThh:mm:ss")
```

### 4.1.2 Consultas SQL

#### 4.1.2.1 Consulta de fecha

Este procedimiento recibe el nombre de un campo en la tabla de información y una tira con la consulta de fecha y devuelve una tira con la consulta SQL lista para utilizar en una consulta mayor en la base de datos SQL. El diagrama de flujo de este procedimiento se detalla a continuación.

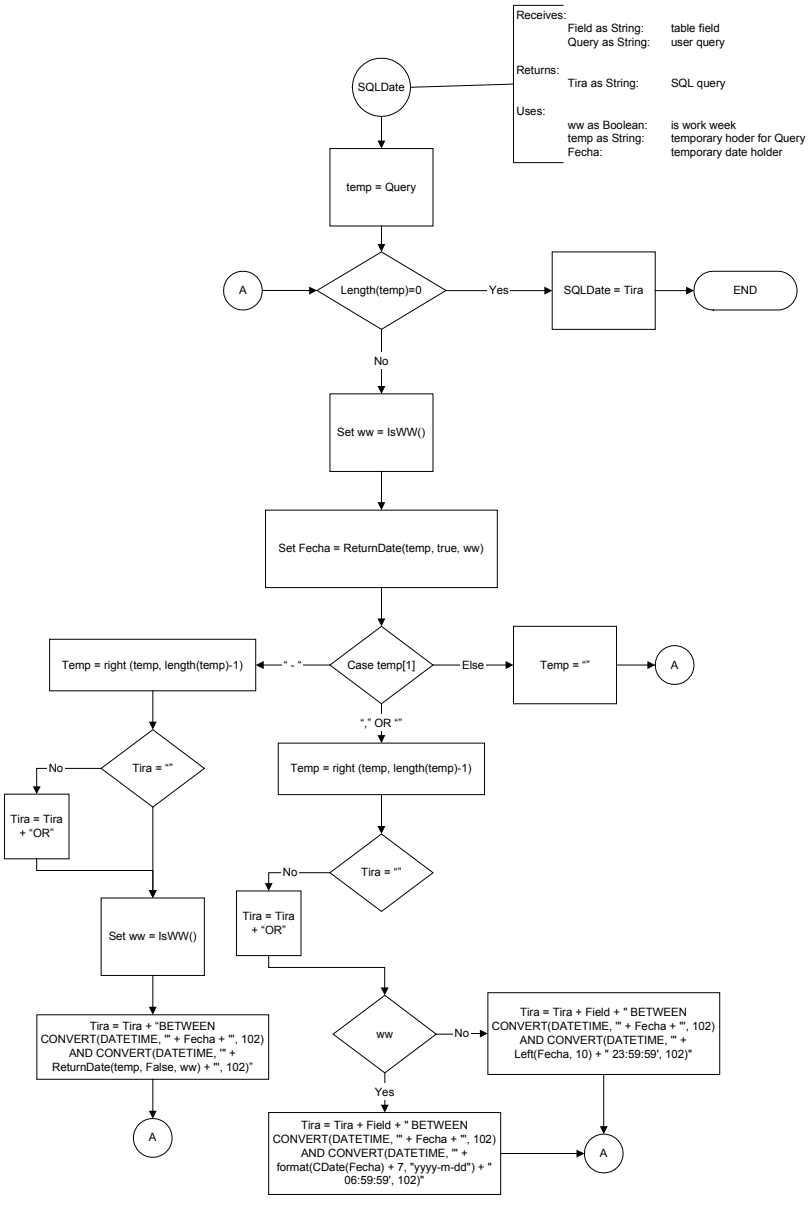



**Figura 4.3** Diagrama de flujo de SQLDate

### 4.1.2.2 Consulta AND / OR

Esta consulta tiene como propósito convertir las consultas provenientes de las páginas web a consultas SQL sintácticamente correctas. El diagrama de flujo se representa a continuación.

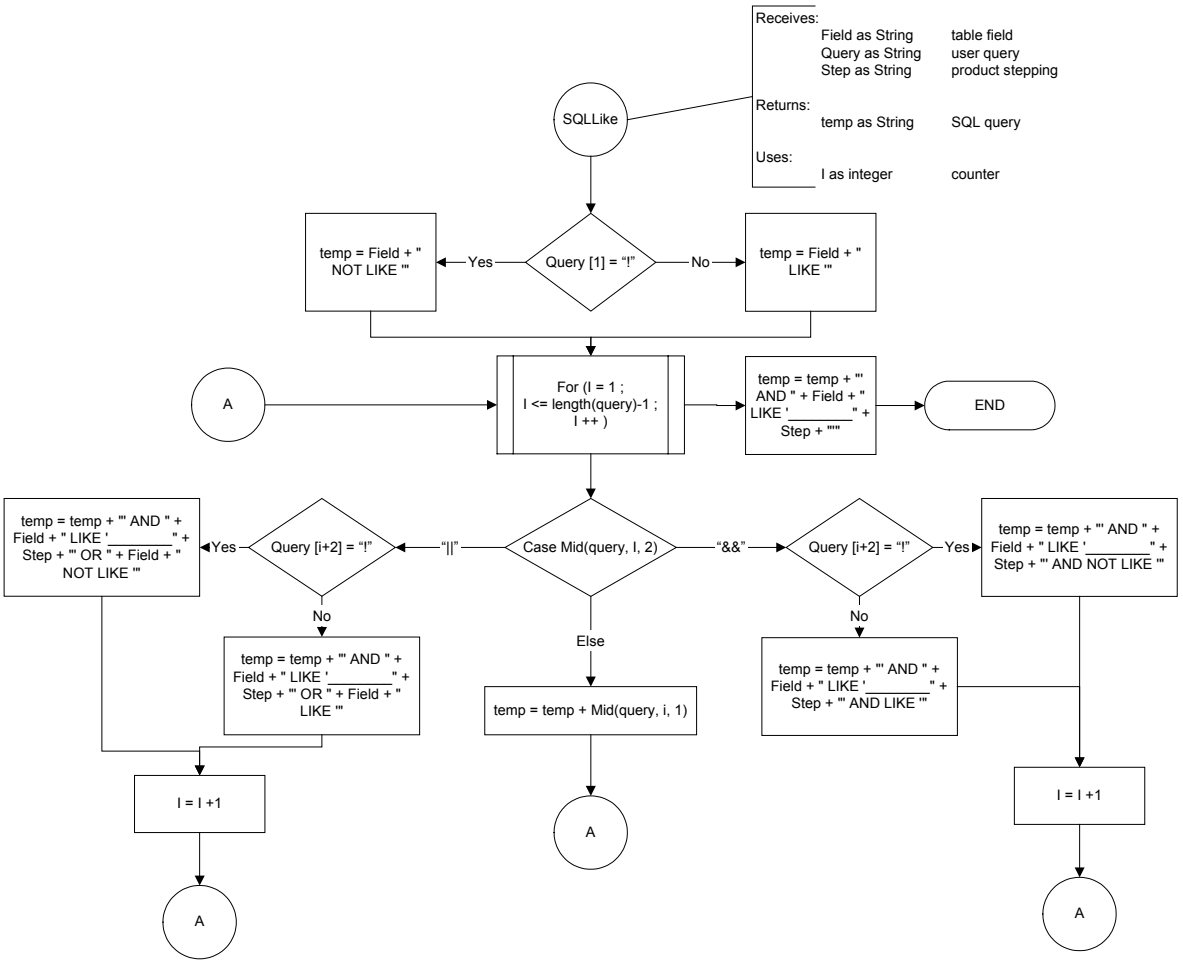



**Figura 4.4** Diagrama de flujo de SQLLike

### 4.1.3 Conexión remota a los servidores de producción

Para realizar esta conexión se utilizó un componente DLL con que cuenta Intel para conexión entre el lenguaje C++ y los servidores de producción. Debido a esto, fue necesario primero diseñar una interfase entre C++ y Visual Basic de forma tal que permitiera trabajar directamente con el componente DLL.

Tanto los procedimientos del DLL como los procedimientos de interfase entre lenguajes están catalogados como confidenciales y por tanto no pueden ser discutidos en mayor grado.

Es importante destacar que este función que se incluyó en el sistema no estaba incluida en los objetivos originales del proyecto. Sin embargo, se consideró de gran importancia incluir esta funcionalidad al sistema para mejorar la interactividad y la cantidad de información con que cuenta el usuario para la toma de decisiones.

## 4.2 Código en UNIX

El código en UNIX es la base del funcionamiento de PROTAF. Este código es el encargado de obtener la información de los servidores de producción y agregarla periódicamente a la base de datos. Además, a petición del usuario, se obtiene la información de validación de lote en tiempo real y se despliega en pantalla. El modelo utilizado para el subsistema de cache en UNIX se muestra en la siguiente figura.

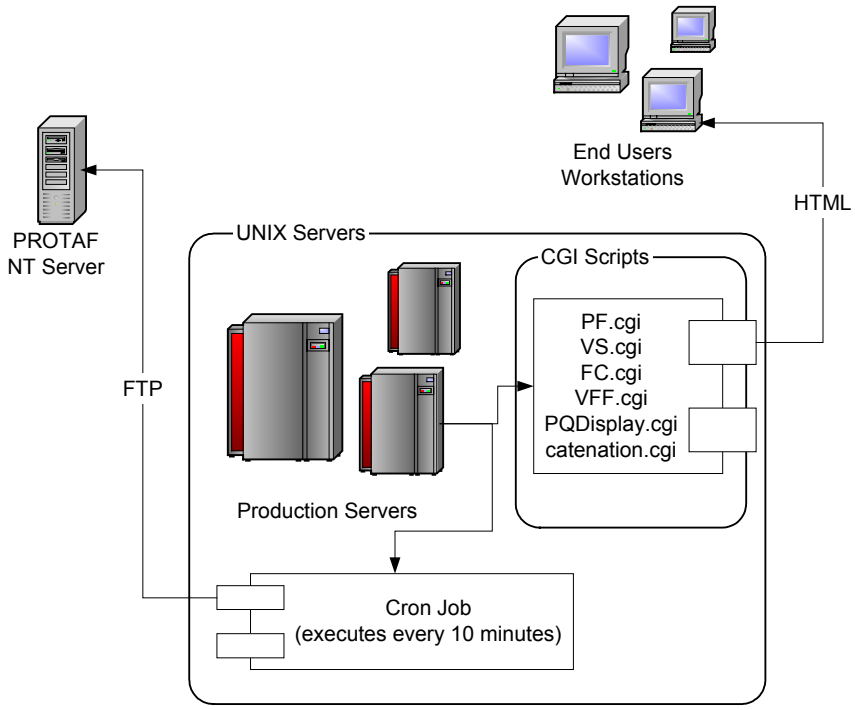



**Figura 4.5** Modelo del sistema en UNIX

En esta segunda etapa del proyecto, se realizaron varios cambios al sistema en UNIX, principalmente se agregaron los programas CGI (Common Gateway Interface) para obtener la información de validación de lotes. Sin embargo, se realizaron también cambios menores a los programas ya escritos en la primera etapa del proyecto. La mayoría de estos cambios fueron pequeñas correcciones al sistema original por errores encontrados y otros se realizaron para mejorar el funcionamiento. A continuación se detalla cada uno de los programas escritos para la nueva versión del sistema en UNIX.

### 4.2.1 Ejecutar.perl

En su primera etapa, el sistema no era capaz de saber si existía alguna otra instancia del sistema ejecutando. Por ello, no era factible utilizar “cron jobs” o trabajos de cronometro para ejecutar el actualizador de la base de datos periódicamente. Uno de los primeros cambios que se realizó al sistema fue entonces el de asegurarse que no existiera otra instancia antes de ejecutar. Esto se logró implementando un programa llamado *Ejecutar.perl* que revisa un archivo para comprobar si la variable *Ejecutando* se encuentra activa o no. En caso negativo, la activa y luego ejecuta el actualizador. Si la variable está activa, envía un mensaje indicando el problema y detiene la ejecución. El diagrama de bloques se observa en la figura 4.6 y el código fuente se incluye en el apéndice 8.2.1.

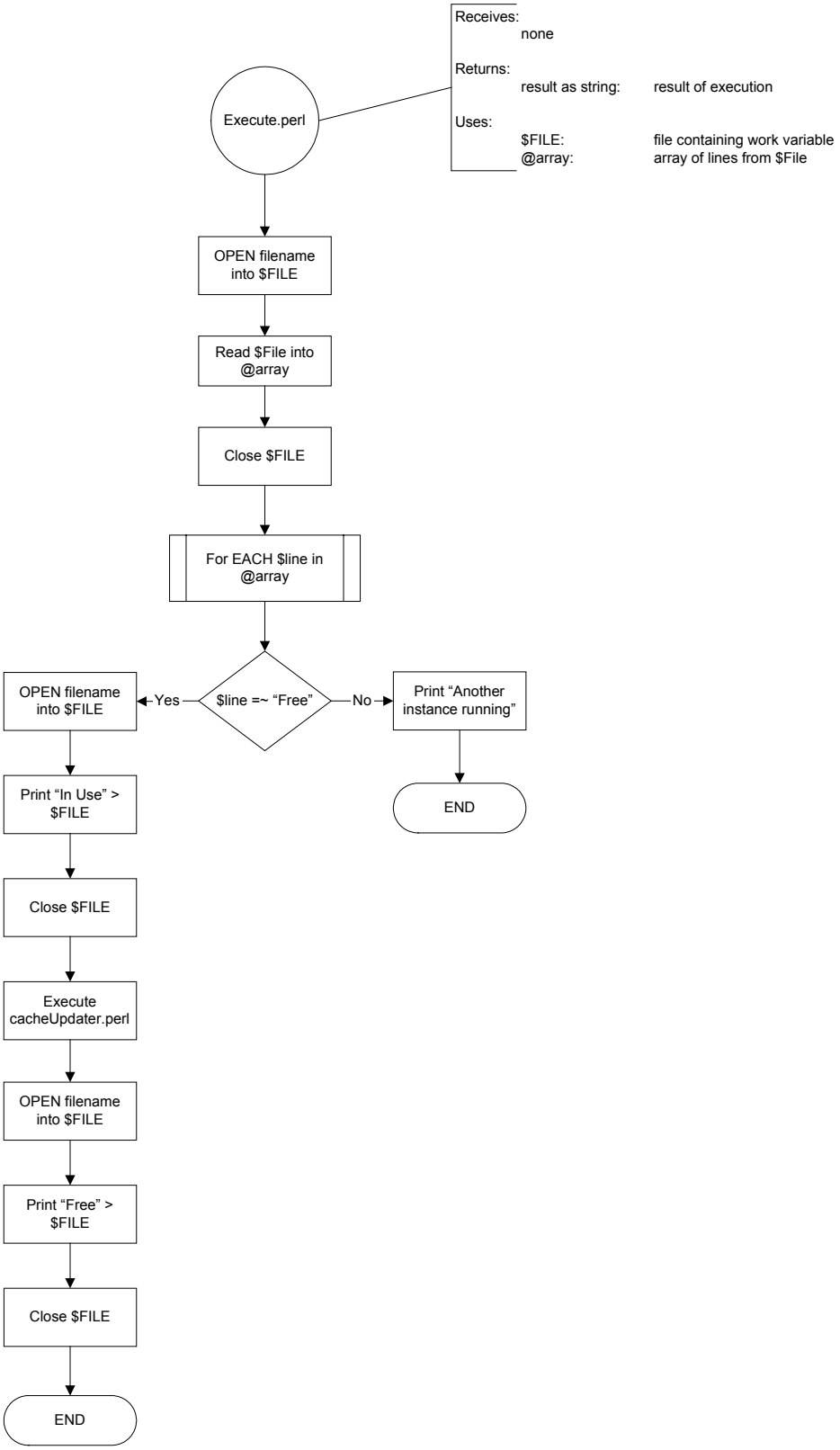



**Figura 4.6** Diagrama de flujo de Ejecutar.perl

### 4.2.2 PQDisplay.cgi

Este programa se encarga de desplegar un resumen, el resultado y el detalle de la validación para los lotes validados en producción, excepto aquellos que se validan utilizando VFF. PQDisplay.cgi consulta un código preexistente para obtener así el detalle de las pruebas de validación unidad por unidad. El programa se encarga de resumir la información y recalcular el nuevo porcentaje de fallas para compararlo con el límite aceptable y por tanto obtener el resultado de la validación. Al final presenta el detalle de la prueba para que el usuario tenga la posibilidad de realizar un análisis mas a fondo de la información de que dispone. En la figura 4.7, se presenta el diagrama de flujo del programa y en la figura 4.8 el resultado que obtiene el usuario en pantalla. Los códigos fuentes del programa se pueden ver en el apéndice 8.2.2.

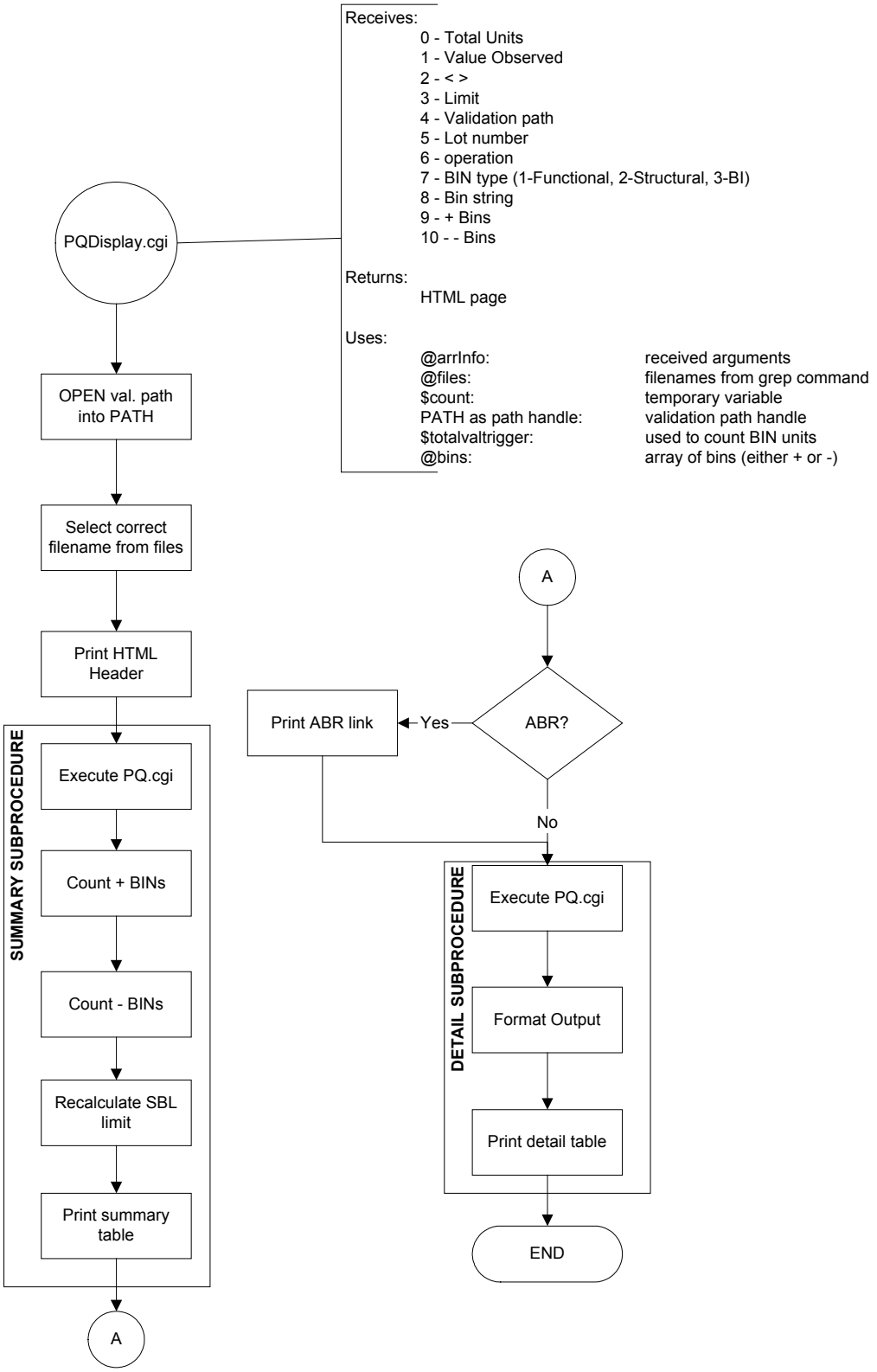



**Figura 4.7** Diagrama de flujo de PQDisplay.cgi

# XS Data Output

XS owner: Jose Chico

## Summary:

**Validation of Bin:** YY

| | Production | Validation |
|---|---|---|
| **Total Units** | 690 | 34 |
| **Trigger % (Units)** | 1.014 (7) | 0.870 (6) |
| **Limit %** | Confidential | Confidential |
| **SBL Result** | NO PASS | VALID LF |

**Validation path:** $valpath

**LYA link:** LYA

## Details:

E4216A837 7044/1A **Prod:** Confidential **Prog:** Confidential **Tester:**RTST412

LOT NUMBER: Confidential

| NUM | PART_ID | BIN |
|---|---|---|
| 0001 | Confidential_Confidential | Confidential |
| 0002 | Confidential_Confidential | Confidential |
| 0003 | Confidential Confidential | Confidential |
| 0004 | Confidential Confidential | Confidential |

**Figura 4.8** Resultado de PQDisplay.cgi

### 4.2.3 VFF.cgi

Como se mencionó en un apartado anterior, cuando se tiene una falla funcional en una prueba estructural, entonces se necesita ejecutar a validación que simula las condiciones en fábrica. Este programa ejecuta el código preexistente que simula las condiciones requeridas y despliega el resultado en pantalla. Al igual que en el apartado anterior, las siguientes dos figuras muestran el diagrama de flujo y resultado del programa, respectivamente. Los códigos fuentes se detallan en el apéndice 8.2.3.

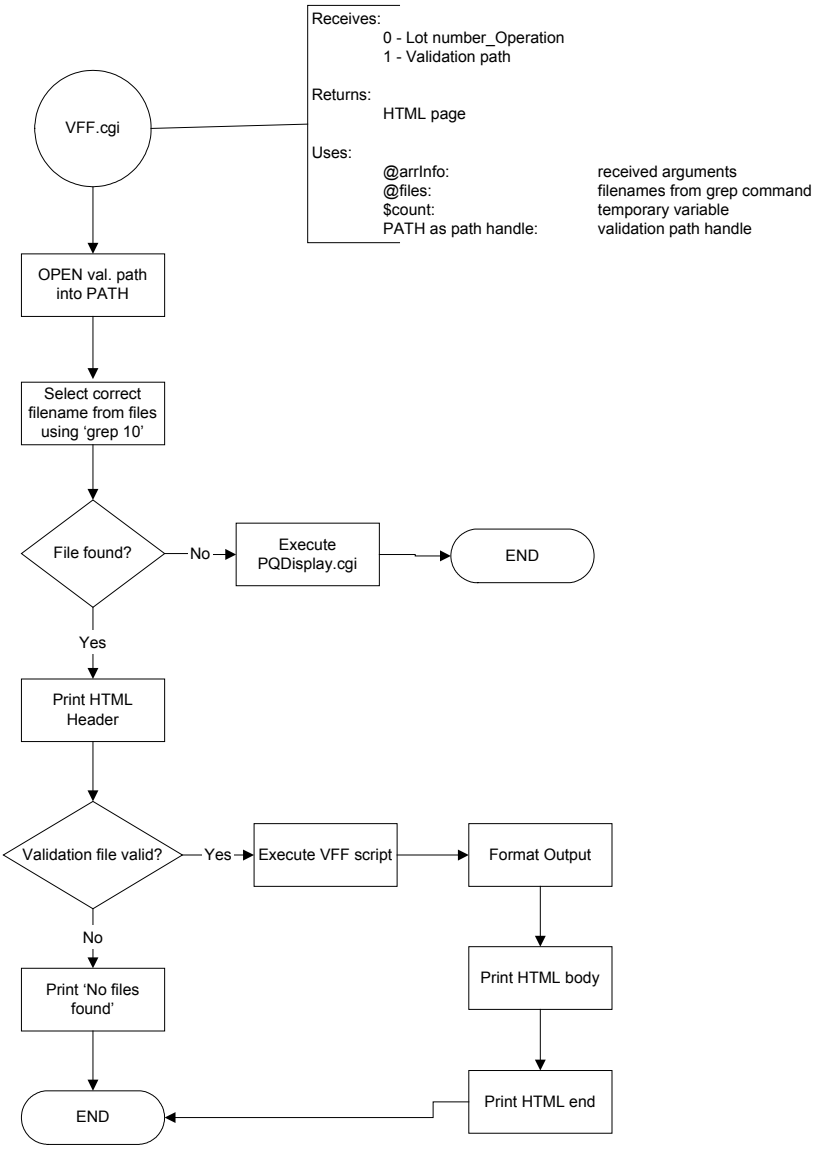



**Figura 4.9** Diagrama de flujo de VFF.cgi

# VFF Data Output

VFF owner: Isela Marin

```
========================================================================
SortLikeTape SBL Calculation Check
      Revision 3.3
========================================================================
Processing ......
Prod directory Found ....
Prod directory Found ....
Prod directory Found ....
Prod directory Found ....
Engr directory Found ....
Engr directory Found ....
Engr directory Found ....
Product is SV RK8EDZJV
*********************************************
***   SortLikeTape SBL Check Revision 3.3   ***
***    date: Tue Apr 30 09:59:05 GMT-0600 2002   ***
*********************************************
Tempr:87.0  System:ZTST501
PBIC summary files :  1A 2A
Tempr:0.0  System:ZTST401
SLT summary files : summary path
Softbin to exclude count:0  SBIN: BINS
+--------------------------------------+
|SLT SBL Limit                  =     4|
|                                      |
|Total SLT Good units           =    13|
|Total SLT Functional rejects   =     3|
|Total SPBIC Good units         =   286|
|Total SPBIC Funcional rejects  =    14|
+--------------------------------------+
+-----------------------------------+
|       PASS SLT Limit!!            |
+-----------------------------------+
```

Production paths

Disposition: Move the lot as normal.

**Figura 4.10** Resultado de VFF.cgi

### 4.2.4 VS.cgi

El diagrama de flujo del procedimiento para validaciones de stress se detalla en la figura 4.11 y un ejemplo de la información que despliega en pantalla se observa en la figura 4.12. El código fuente se incluye en el apéndice 8.2.4.

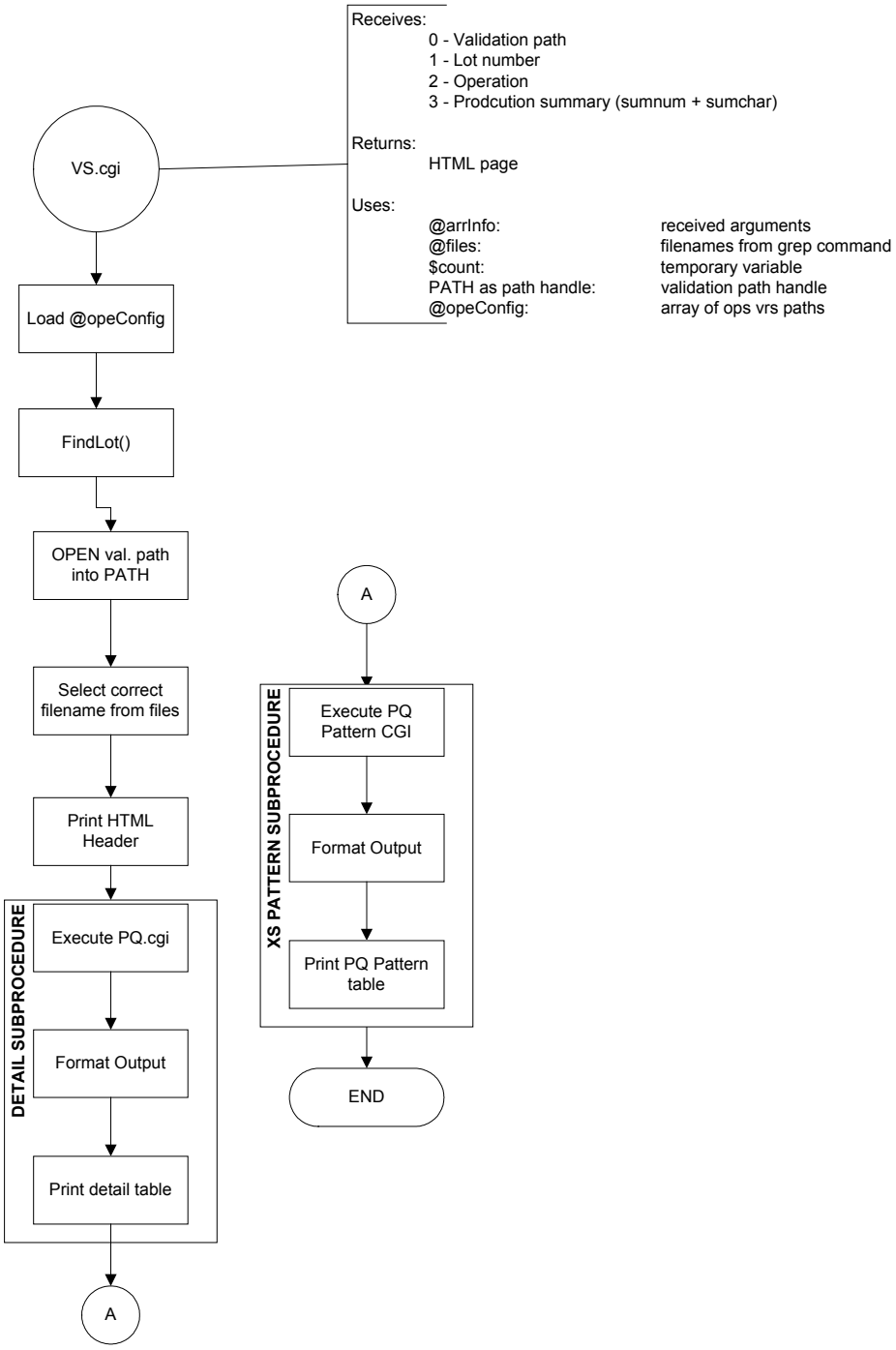



**Figura 4.11** Diagrama de flujo de VS.cgi

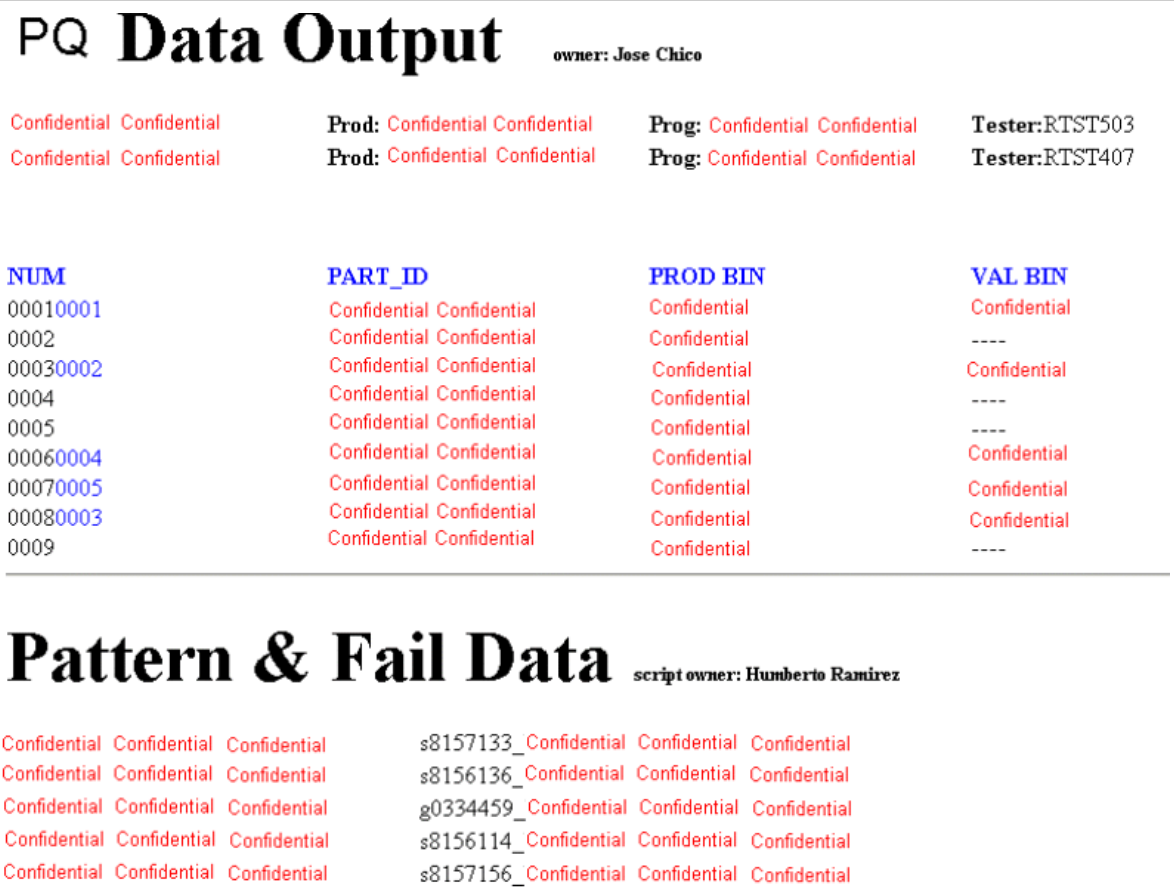



**Figura 4.12** Resultado de VS.cgi

### 4.2.5 PF.cgi

Al igual que con los códigos antes descritos, la validación o prueba en frío utiliza código preexistente para desplegar la información detallada de la validación. En este caso, la información viene en el formato de columnas. Cada una de las columnas es una prueba a distinta temperatura. Como se observa en la figura 4.14, el título de cada columna detalla la temperatura de la prueba. En la figura 4.13 se observa el diagrama de flujo del código. El código fuente se incluye en el apéndice 8.2.5.

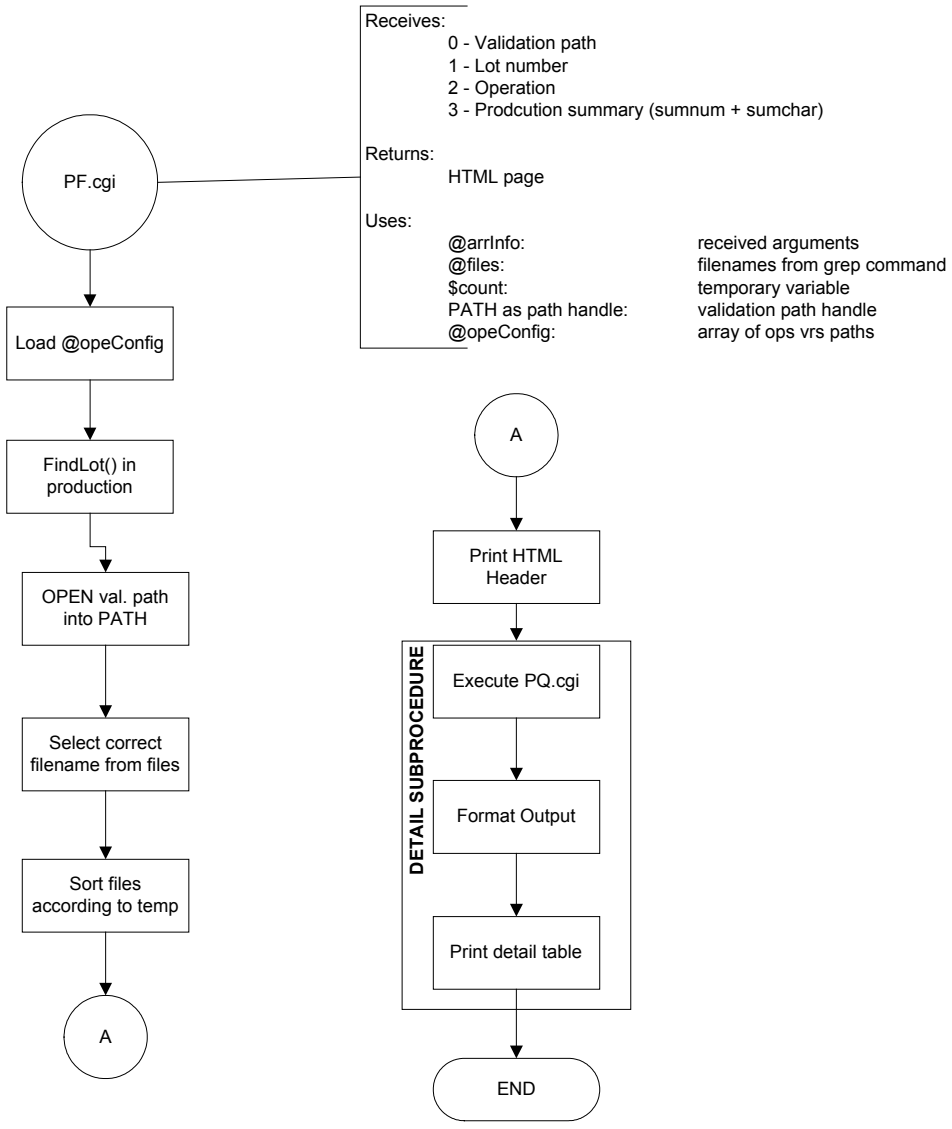
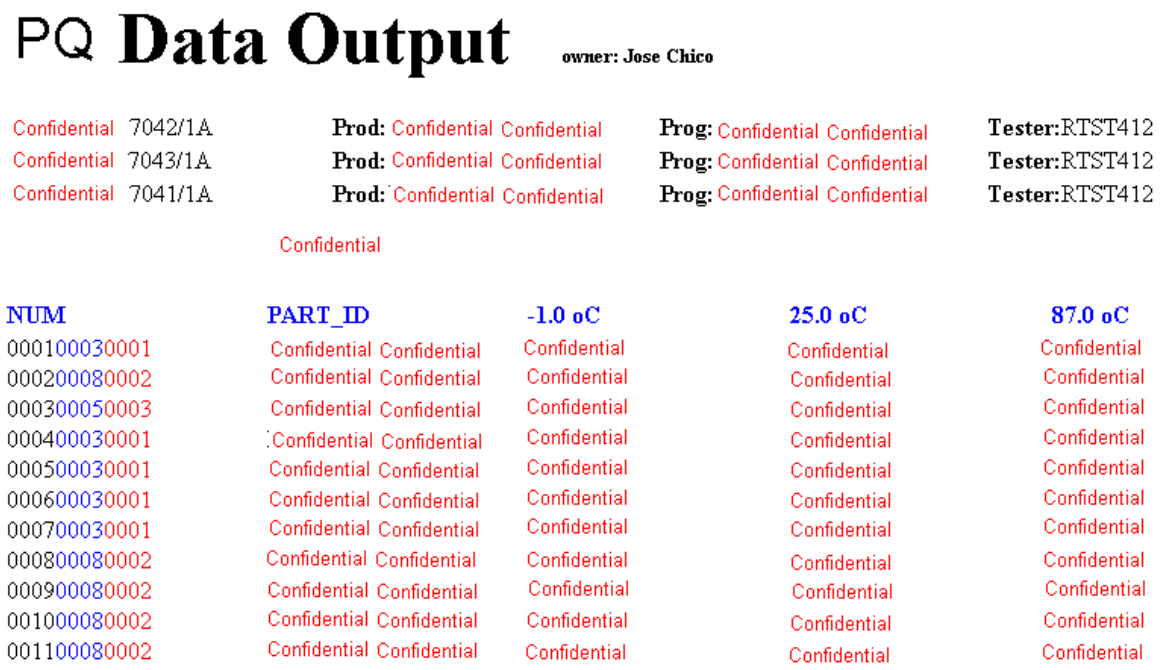



**Figura 4.13** Diagrama de flujo de PF.cgi


PQ Data Output owner: Jose Chico

Confidential 7042/1A Prod: Confidential Confidential Prog: Confidential Confidential Tester:RTST412
Confidential 7043/1A Prod: Confidential Confidential Prog: Confidential Confidential Tester:RTST412
Confidential 7041/1A Prod: Confidential Confidential Prog: Confidential Confidential Tester:RTST412

Confidential

| NUM | PART_ID | -1.0 oC | 25.0 oC | 87.0 oC |
|---|---|---|---|---|
| 000100030001 | Confidential Confidential | Confidential | Confidential | Confidential |
| 000200080002 | Confidential Confidential | Confidential | Confidential | Confidential |
| 000300050003 | Confidential Confidential | Confidential | Confidential | Confidential |
| 000400030001 | Confidential Confidential | Confidential | Confidential | Confidential |
| 000500030001 | Confidential Confidential | Confidential | Confidential | Confidential |
| 000600030001 | Confidential Confidential | Confidential | Confidential | Confidential |
| 000700030001 | Confidential Confidential | Confidential | Confidential | Confidential |
| 000800080002 | Confidential Confidential | Confidential | Confidential | Confidential |
| 000900080002 | Confidential Confidential | Confidential | Confidential | Confidential |
| 001000080002 | Confidential Confidential | Confidential | Confidential | Confidential |
| 001100080002 | Confidential Confidential | Confidential | Confidential | Confidential |

**Figura 4.14** Resultado de PF.cgi

### 4.2.6 FC.cgi y catenation.cgi

La ventana del resultado de la validación de falla de calidad se divide en dos, como se observa en al figura 4.16. La ventana superior muestra el detalle de la validación en su primera parte y luego se despliega el listado de los archivos que existen en el directorio de validación en UNIX. Cuando el usuario hace clic sobre alguno de los archivos en la ventana superior, estos se despliegan en la ventana inferior utilizando el código *catenation.cgi*. Este código utiliza la función *cat* de UNIX para desplegar el contenido de los archivos. El diagrama de flujo de *FC.cgi* se observa en la figura 4.15. Ambos códigos fuentes se incluyen en los apéndices 8.2.6 y 8.2.7 respectivamente.

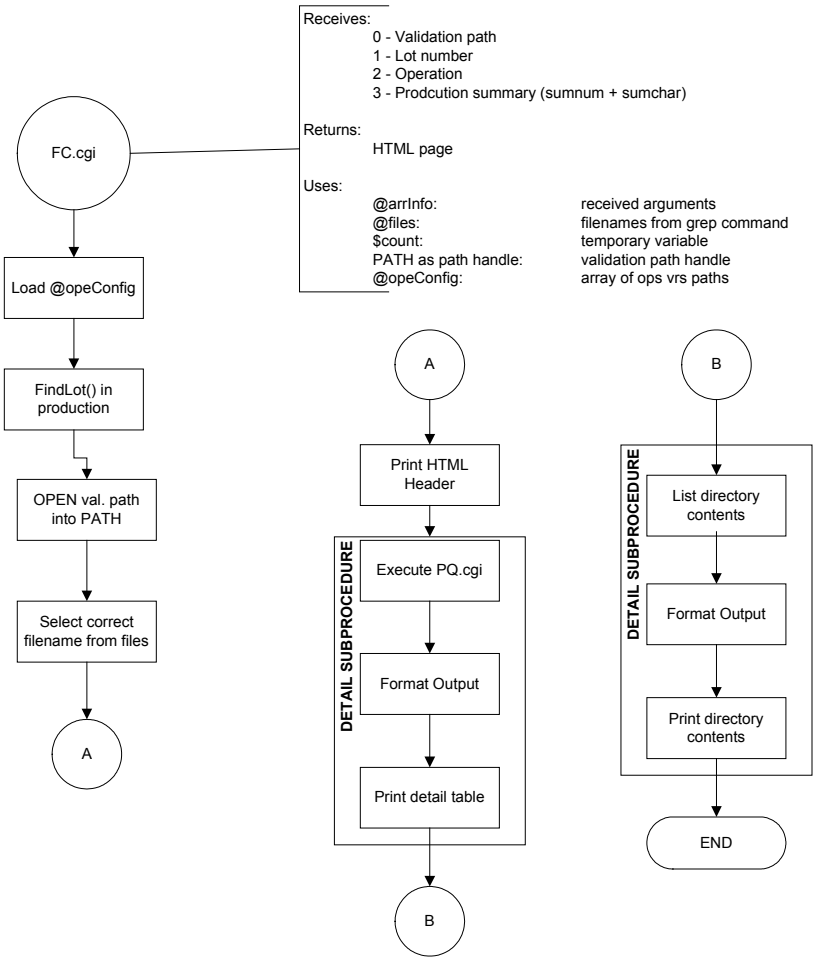



**Figura 4.15** Diagrama de flujo de FC.cgi

0008 E1293251_818_+3_-4 B9215
0008 Confidential Confidential Confidential
0009 Confidential Confidential Confidential
0009 Confidential Confidential Confidential
0010 Confidential Confidential Confidential
0010 Confidential Confidential Confidential

Confidential

---

# Validation directory contents

**Click on any file to see contents**

Oct 25 2001 - 1A

Oct 25 2001 - Shmoo

Oct 25 2001 - Shmoo2

---

**Content for the file:**
Confidential Confidential Confidential Confidential Confidential Confidential Confidential val/Shmoo

```
-----------------------------------------------------------------------------------------
|Title : NULL Date: Thu Oct 25 16:06:31 |
|Test : bf61_1450_maxvcc_012424_1 |
|Timing : nrz_bfd61 |
|Pattern list : bf_io_at_speed_012424_list |
|Exec patterns: ALL ENABLED PATTERNS IN THE LIST |
-----------------------------------------------------------------------------------------
|X axis title: PSBPer (11.00n - 40.00n )S | PINS: NULL Step: 500.000p |
|Y axis title: vcc_spec_wmt (1.10 - 1.80 )V | PINS: NULL Step: 50.000m |
-----------------------------------------------------------------------------------------
```

**Figura 4.16** Resultado de FC.cgi

## 4.3 Páginas web (ASP)

El código de páginas web se desarrollo utilizando Microsoft FrontPage para el diseño gráfico y Microsoft Development Enviroment para el código web. Las páginas web forman parte de la interfase de usuario final que es la única expresión del sistema que logran observar los usuarios. Para utilizar esta interfase se tienen los siguientes requerimientos del usuario:

a. Internet Explorer 4.0 (5.0 o superior recomendado)

b. 'Office Web Components' o 'Office 2000' (este último incluye los OWC)

Es necesario que ambos programas estén instalados en todas las estaciones de trabajo que los usuarios vayan a utilizar. Dado que Intel cuenta con licencia corporativa para ambos programas, estos requerimientos no influyen en gastos para el presupuesto del proyecto.

La figura **4.17** muestra el menú principal de la interfase. Desde el menú se puede acceder el 'LF Information' (información de pruebas), 'Query by Lots' (consulta mediante lotes), 'Lots on Hold Report' (reporte de lotes detenidos) y 'System Configuration' (configuración del sistema). En apartados siguientes se detalla cada módulo del sistema.


**Figura 4.17** Menú principal de la interfase de usuario

### 4.3.1 Módulo de información de pruebas

El módulo de información de LFs permite mediante un filtro realizar una consulta a la base de datos acerca de los resultados de las pruebas de los lotes en el piso de producción. Los filtros accesibles al usuario se observan en la siguiente figura.

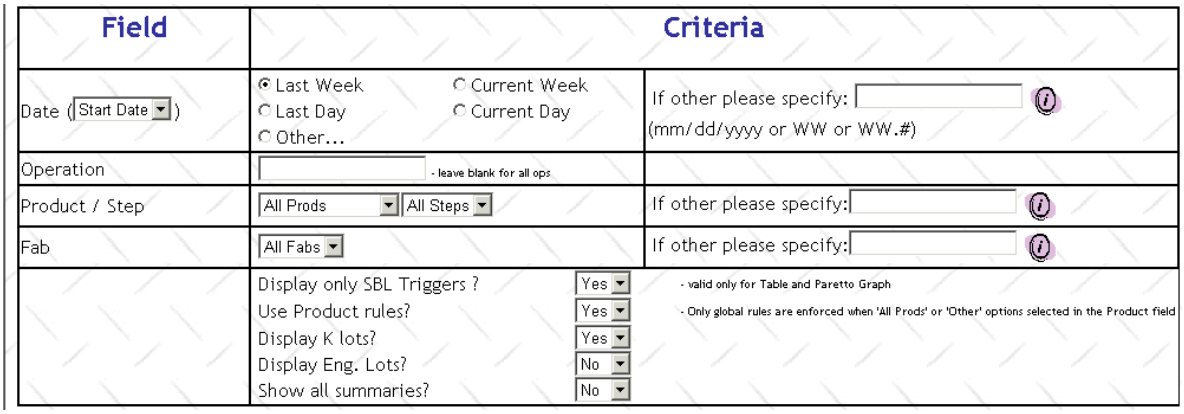
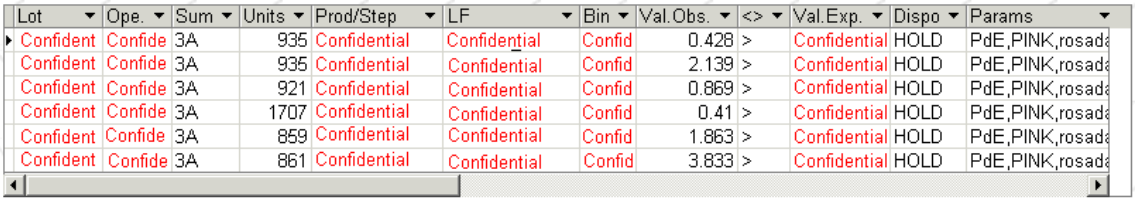



**Figura 4.18** Filtro para la información de pruebas

Entre los filtros se encuentra el filtro de fecha, el filtro de operación, el de producto, el de fábrica y algunos más detallados como solo mostrar fallas, utilizar reglas de productos, desplegar lotes K, desplegar lotes de ingeniería y mostrar todos los sumarios. Los filtros de fecha, operación, producto y fábrica le permiten al usuario seleccionar el filtro desde una lista predefinida o digitar por sí mismo el filtro a utilizar.

Una vez filtrada la información, el resultado se despliega en una tabla pivote que le permite al usuario interactuar con esta para refinar aún más el filtro de información, o copiar los resultados a una hoja Excel por ejemplo. Un ejemplo puede ser visto en la figura **4.19** a continuación.



| Lot | Ope. | Sum | Units | Prod/Step | LF | Bin | Val.Obs. | <> | Val.Exp. | Dispo | Params |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Confident | Confide | 3A | 935 | Confidential | Confidential | Confid | 0.428 | > | Confidential | HOLD | PdE,PINK,rosad |
| Confident | Confide | 3A | 935 | Confidential | Confidential | Confid | 2.139 | > | Confidential | HOLD | PdE,PINK,rosad |
| Confident | Confide | 3A | 921 | Confidential | Confidential | Confid | 0.869 | > | Confidential | HOLD | PdE,PINK,rosad |
| Confident | Confide | 3A | 1707 | Confidential | Confidential | Confid | 0.41 | > | Confidential | HOLD | PdE,PINK,rosad |
| Confident | Confide | 3A | 859 | Confidential | Confidential | Confid | 1.863 | > | Confidential | HOLD | PdE,PINK,rosad |
| Confident | Confide | 3A | 861 | Confidential | Confidential | Confid | 3.833 | > | Confidential | HOLD | PdE,PINK,rosad |

**Figura 4.19** Tabla pivote que despliega la información de pruebas

Alguna de la información que aparece en la tabla es la siguiente:

a. número de lote
b. operación de prueba
c. sumario de prueba
d. unidades del lote
e. identificador de producto
f. falla detectada
g. código de la falla detectada
h. valor observado
i. comparador
j. valor límite
k. disposición
l. parámetros de la disposición
m. fecha de prueba
n. fábrica
o. validación del lote

Al pie de la tabla pivote se tienen los botones de acción que se muestran en la figura **4.20**. Estos botones se incluyen en prácticamente todas las páginas que despliegan información en el sistema de forma tal que el usuario tiene la información que necesita a mano sin necesidad de navegar a otro lugar del sitio para obtenerla.

**Figura 4.20** Botones de acción

El primer botón le permite al usuario ver el resultado en detalle de la prueba realizada en producción. El segundo, despliega la disposición del lote, en otras palabras el paso a seguir una vez realizada la prueba de producción. El siguiente botón despliega, si está disponible, la información de la validación del lote. Esto incluye el resumen de la validación, el resultado y el detalle de esta. El último botón despliega el estatus en tiempo real del lote en el piso de producción.

### 4.3.2 Módulo de consulta mediante lotes

La página principal del módulo de consulta mediante lotes es también un filtro. Este permite filtrar la información cruda de pruebas y validaciones para su despliegue en otra tabla pivote. El filtro, que se muestra en la figura **4.21**, permite filtrar la información mediante la utilización de una lista de lotes o usando un filtro de fecha.

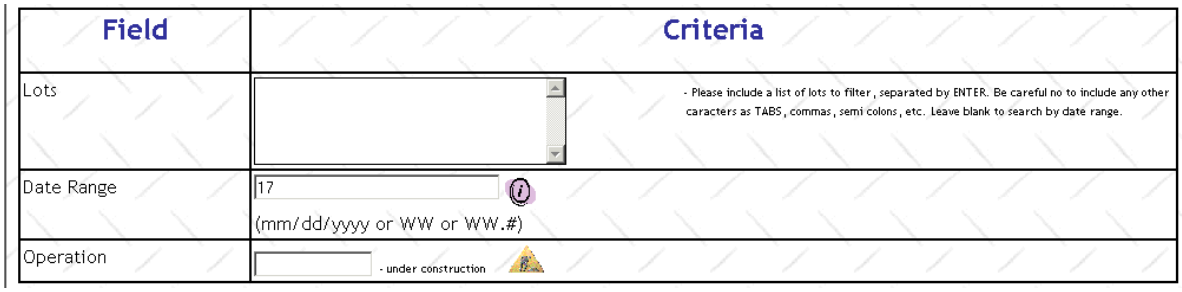



**Figura 4.21** Filtro de consulta mediante lotes

Una vez seleccionado el filtro a utilizar, se despliega la información que se encuentra en la base de datos en una tabla pivote para su posterior procesamiento y análisis por parte del usuario final.

### 4.3.3 Módulo del reporte de lotes en espera

Así como se mencionó anteriormente para la conexión al los servidores de producción, este módulo no está incluido en los objetivos originales del proyecto. Sin embargo, la importancia del reporte de lotes en espera para el departamento, y toda la fábrica en general, causó que a la implementación de este módulo se le diera carácter de muy importante. Además, su implementación trajo consigo una mayor utilización del sistema por parte de los usuarios finales, algo que había sido en cierto grado de gran dificultad debido al poco interés que algunos de ellos mostraban por cambiar los antiguos métodos manuales por el nuevo sistema. A continuación se detalla el módulo y su funcionamiento.

Cuando un lote es probado y falla por alguna razón este lote se detiene hasta que sea validado. Una vez validado, y dependiendo del resultado, el lote puede continuar con el proceso normal o ser desechado. Este módulo le permite a

los ingenieros del departamento realizar una consulta en tiempo real de cuáles lotes están en espera y la razón. Además despliega el resultado, si está disponible, de la validación del lote.

Los lotes en espera pueden ser filtrados por planta y/o operación, como lo muestra la siguiente figura:

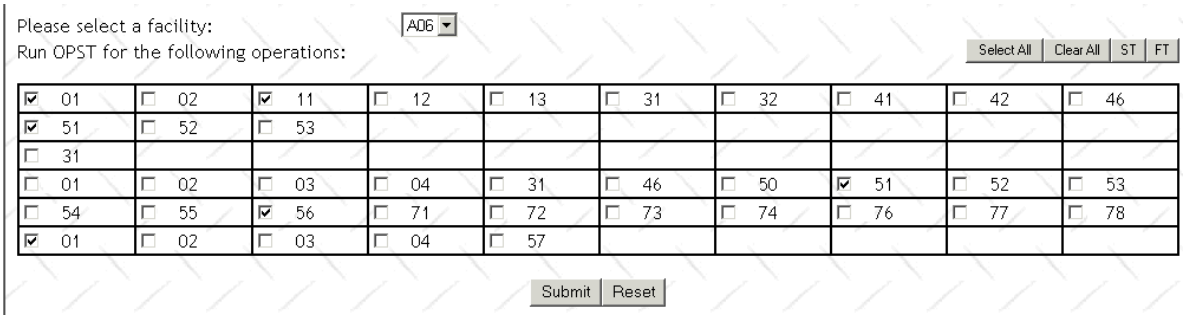



**Figura 4.22** Filtro de lotes detenidos

El detalle de la información que se despliega no puede ser presentada en el presente informe, debido a medidas de confidencialidad. Sin embargo, la información que se presenta es la siguiente:

a. categoría de la detención
b. nombre del encargado
c. número de lote
d. código del producto
e. operación actual
f. estatus de la detención
g. resultados de la última prueba de producción
h. validación del lote
i. días detenido

### 4.3.4 Módulo de configuración del sistema

Dado que el sistema debe ser fácilmente proliferable a otras plantas de la compañía en otros países, desde su primera etapa se incluyó un módulo de configuración del sistema a nivel NT, así como existe uno a nivel UNIX. Este último se comentó anteriormente.

En este módulo se puede configurar el nivel de acceso de los usuarios del sistema. Todo aquel usuario que no se encuentra configurado en la base de datos del sistema tendrá acceso como 'invitado'. Entre otras cosas, se puede configurar los filtros predefinidos de productos y fábricas, así como las reglas de cada producto o las reglas globales. La página de configuración se observa en la figura **4.23**.

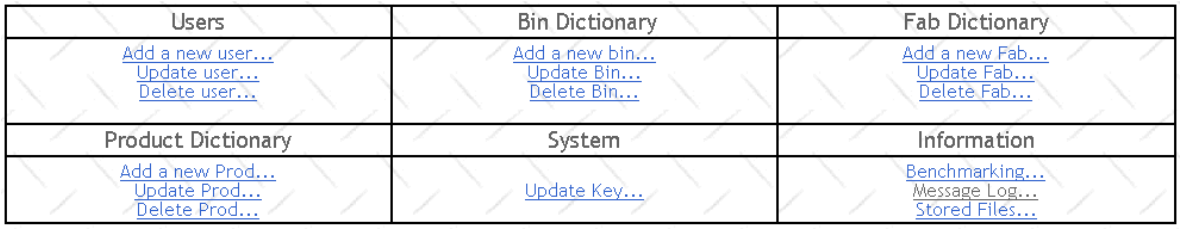



**Figura 4.23** Página de configuración del sistema

También, este módulo permite la fácil configuración de las distintas direcciones del repositorio de datos SQL, o las páginas CGI que se especificaron en apartados anteriores. De esta forma, el sistema se puede instalar en cualquier otro sitio de la compañía con sólo configurar el sistema desde este módulo. En la esquina inferior derecha de la figura **4.23** se observa el acceso a distintas informaciones de la operación del sistema como lo son los tiempos de lectura de los archivos de producción, los tiempos de carga de la información entre otros, así como el listado de mensajes del sistema donde se pueden observar mensajes de información, como por ejemplo cuando un usuario ingresa al sistema, hasta mensajes fatales, como cuando existe algún error en la carga de datos.

Las pantallas específicas de cada configuración no pueden ser mostradas dado que muestran datos confidenciales.

### 4.3.5 Módulo de ayuda

Este módulo no tiene acceso mediante el menú principal pero puede ser consultado prácticamente desde cualquier otro módulo del sistema. Este contiene información importante de uso del sistema así como explicaciones del funcionamiento de las consultas y de los lineamientos que debe seguir el usuario al escribir sus propias consultas. Además, al sistema se le incluyó un ayudante para facilitar su aprendizaje. Un ejemplo de un ayudante puede ser visto en la siguiente figura.

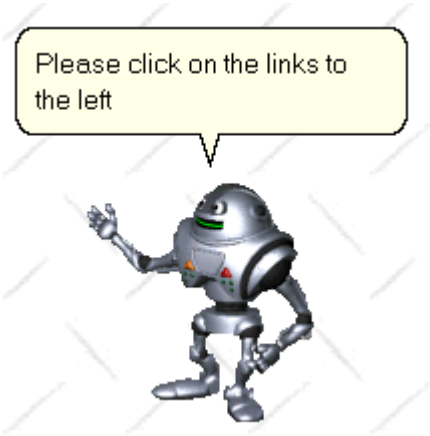



**Figura 4.24** Ayudante del sistema

# CAPÍTULO 5 ANÁLISIS Y RESULTADOS

## 5.1 Explicación del diseño

Como se hizo hincapié en el capítulo anterior, la herramienta de análisis diseñada se divide principalmente en tres partes: la base de datos, el código en UNIX y la interfase con el usuario o servidor NT. A continuación se explica las razones para el diseño de cada una de las partes del sistema.

### 5.1.1 Base de Datos SQL

Aunque la base de datos SQL ya se encontraba implementada en la primera etapa del proyecto, la razón de su escogencia es muy sencilla. Dado que Componentes Intel de Costa Rica cuenta con licencias corporativas de los programas Microsoft, en el afán de reducir costos, se necesitaba seleccionar una base de datos Microsoft. Microsoft Corporation cuenta con dos formatos de base de datos, a saber Access y SQL. Access, incluida en Microsoft Office, es bastante simple y diseñada especialmente para usuarios con poco o ningún conocimiento acerca del diseño de bases de datos o programación de las mismas. Es debido a esto que ésta es inadecuada cuando se necesita manejar mucha información (varios cientos de megabytes o superior) o una cantidad bastante alta de usuarios conectados a la base de datos al mismo tiempo (usualmente superior a 5 o 10 usuarios). En cambio, SQL es una base de datos diseñada para satisfacer las demandas de usuarios más avanzados y por ello su diseño es más estable y de mayor eficiencia. Inclusive, la nueva versión de SQL 2000, utilizada en el proyecto, fue rediseñada para utilizarse de manera confiable en servidores web, donde la cantidad de información y de usuarios son bastante grandes.

Además, esta versión de la base de datos incluye una interfase XML que permite la interacción directa del explorador con la base de datos de una forma rápida pero segura. Esta opción no se utilizó en el actual proyecto, pero se incluye como una recomendación para futuras etapas en apartados posteriores.

El diseño en sí de la base de datos y de la información que esta guarda sufrió pocos cambios, dado que la versión original de la misma (la versión de la

etapa 1 del proyecto), era lo suficientemente robusta para soportar las nuevas funciones que se le incluyeron al sistema.

### 5.1.2 Código Perl

Para la programación en UNIX se utilizó el lenguaje Perl. Este fue seleccionado debido a que, como se explica en apartados anteriores, la información que recopila el subsistema en UNIX está guardada en archivos de texto en estructuras de directorios de UNIX. El lenguaje Perl fue concebido en sus inicios para manejar cadenas de texto y por ello su utilización en esta parte del sistema se decidió desde los comienzos del proyecto.

El lenguaje Perl original es de carácter completamente gratuito, pero cabe recalcar que Componentes Intel de Costa Rica cuenta con las licencias para una de las versiones comerciales de los servicios del lenguaje en UNIX. Por ello, su utilización tampoco conlleva gastos mayores al presupuesto del proyecto.

### 5.1.3 Interfase con el usuario

Es importante destacar que se utilizaron dos versiones diferentes del lenguaje VBScript, una versión ejecuta en el servidor y la segunda en el cliente[1]. Ambos códigos tienen sus beneficios y sus perjuicios y su utilización se hizo según los siguientes lineamientos:

#### Código en el servidor

a. Tiene mayor seguridad dado que ejecuta completamente en el servidor y por tanto no hay posibilidades de que se intercepte.

b. El código escrito para ejecutar en el servidor nunca es enviado al cliente, y por tanto no existe forma de ver el código fuente.

c. Dependiendo de la cantidad de usuarios, este código necesita mucho poder de procesamiento del servidor y por ello se ha de

[1] Clientes son aquellos que se conectan al servidor con una método “REQUEST” solicitando el despliegue de una o varias paginas web.

prever una capacidad de procesamiento mayor en el servidor. Esto conlleva un aumento en el presupuesto para su compra.

**Código en el cliente**

a. El procesamiento que realiza este código puede ser interceptado dado que se ejecuta en el cliente.

b. El código fuente se puede observar en los fuentes de la página web.

c. El poder de procesamiento de este código ejecuta completamente en el cliente. Por ello, la cantidad de usuarios no afecta al servidor dado que por cada usuario existe un nuevo procesador que ejecuta los comandos requeridos por el usuario.

A la hora de implementar el código se intentó llegar a un balance entre varios objetivos principalmente a nivel de procesamiento del servidor versus seguridad del sistema. Al final, toda la interacción que el código necesite realizar con la base de datos se realiza utilizando código en servidor para evitar cualquier problema de seguridad con la base de datos. Aquel código que no accesa la base de datos, se escribió en código cliente para reducir el procesamiento necesario para el servidor y de esta manera poder manejar más usuarios con un servidor de menor costo.

## 5.2 Alcances y limitaciones

Tomando como referencia los objetivos general y específicos planteados para este proyecto en el capítulo 3 del presente documento, es posible ahora realizar un análisis de los alcances, limitaciones y puntos de posible mejoramiento del proyecto desarrollado.

Según lo enumerado en el capítulo 3, y las funciones que el sistema desarrollado es capaz de realizar, es posible concluir que el sistema cumple en forma rigurosa e incluso ha sobrepasado los objetivos planteados al inicio de este proyecto. Por lo tanto, los alcances de este proyecto son los mismos vislumbrados al momento de establecer los objetivos para éste.

El sistema es capaz de dar al usuario una forma rápida y eficaz de manejar la información de las pruebas de los lotes, las validaciones y los lotes en espera. Las tareas del departamento de ingeniería de producto que se determinó debía facilitar el sistema se realizaban, anteriormente, en forma manual y tomaban alrededor de medio día en completar. Utilizando el nuevo sistema es posible realizar estas tareas en menos de 45 minutos. Sin embargo, cabe destacar que siempre hay lugar para mejorar el sistema. En el capítulo siguiente se incluye una lista de recomendaciones para futuras etapas del proyecto.

Los resultados de las validaciones de lotes, principal objetivo del presente proyecto, se obtienen en tiempo real. Esto permite una mayor dinámica en la toma de decisiones dado que se elimina el período de espera que conlleva el cargar dicha información a una base de datos. Esto se logró implementar luego de que se determinó que la información de validaciones de los lotes se guarda en los directorios de producción por un espacio mayor a un año, tiempo suficiente para que dicha información se convierta en obsoleta. Cabe destacar, que a la información de pruebas de los lotes se le determinó y configuró un período de vida en la base de datos de un año. Es por ello que nuevamente las validaciones de más un año de haberse ejecutado no tienen mayor vigencia.

Fuera de los objetivos, se logró aumentar la cantidad de información con que cuenta el usuario utilizando conexiones a los servidores de producción para obtener de estos más información que aquella con que se cuenta en la base de datos. Es importante destacar que dicha información se obtiene en tiempo real, por lo que ayuda en gran manera a la toma de decisiones para los ingenieros y técnicos del departamento en sus actividades diarias.

Como un objetivo no planteado con anterioridad, cabe destacar dentro de los alcances de este proyecto, que a lo largo de su realización se presentó el proyecto en varios de los grupos gobernativos en la estructura de Intel, tanto dentro como fuera de Costa Rica, para su aprobación. Luego de una serie de reuniones y presentaciones a lo largo de varios meses, se obtuvo por parte del grupo más importante en el área de ensamble y pruebas la aprobación para la compra de los servidores necesarios para la implementación del proyecto en cada uno de las 6 fábricas a nivel mundial de Intel. La proliferación de una herramienta de este tipo a nivel mundial desarrollada y soportada desde Costa Rica es una primicia. Por ello, el proyecto y los involucrados han recibido varios reconocimientos por su aporte por parte de los gerentes del departamento y de la fábrica.

# CAPÍTULO 6 CONCLUSIONES Y RECOMENDACIONES

## 6.1 Conclusiones

a. Se hizo un estudio extensivo de los procesos de prueba y validación que se realizan en el piso de producción en Componentes Intel de CR.

b. Se estudió las tareas más cotidianas de los empleados del departamento de Ingeniería de Producto para contemplar la posibilidad de incluir la mayor cantidad de ellas en el sistema.

c. Tomando como base el sistema diseñado en la primera etapa del proyecto se realizó un esquema del nuevo sistema tomando en consideración aquellas sugerencias y observaciones de los usuarios para su mejoramiento.

d. El sistema diseñado es capaz de facilitar el manejo de gran cantidad de información de las pruebas de producción y las validaciones.

e. También se le incluyeron una serie de herramientas al sistema que no se consideraron en los objetivos del presente informe para aumentar su productividad.

f. La calidad de la información que maneja el sistema y su interfase amigable pero poderosa han sido reconocidas por distintos organismos internos de Intel Corporation. Es por ello que el sistema ha sido aprobado para su instalación y posterior utilización en las fábricas de la compañía a nivel mundial.

## 6.2 Recomendaciones

a. Utilizar XML directamente en SQL para aumentar la velocidad de respuesta y reducir la carga de trabajo sobre el servidor web (DLL).

b. Contactar al dueño del programa que genera las tablas en los servidores de producción para resolver problemas con la integridad de los datos en dichas tablas.

c. Continuar el estudio de otras posibles interacciones con otros sistemas de producción para agregar al potencial de la herramienta brindándole más información a los usuarios.

# CAPÍTULO 7 BIBLIOGRAFÍA

a. Microsoft Press. Visual Basic, Manual del Programador. Estados Unidos, 1997.

b. Microsoft Press. System Administration for Microsoft® SQL Server 7.0. Estados Unidos, diciembre 1998.

c. Microsoft Developer Network website - http://msdn.microsoft.com

d. The PERL CD Bookshelf - http://dpg-or.pdx.intel.com/~jherrick/docs/perlcd/

e. CGI Programming - http://www.oreilly.com/openbook/cgi/

f. The UNIX CD Bookshelf - http://dpg-or.pdx.intel.com/~jherrick/docs/unixcd/

# CAPÍTULO 8 APÉNDICES Y ANEXOS

## 8.1 Glosario y abreviaturas

**ASP** Active Server Protocol – Lenguaje de programación para páginas de Internet dinámicas.

**BIN** Dato utilizado para clasificar los lotes de producto

**IIS** Internet Information Service – Servicio de información de internet que permite el acceso a un servidor utilizando cualquier explorador comercial

**Parser** Algoritmo de autochequeo del repositorio de información que permite la autodetección de errores.

**Perl** Lenguaje de programación que cuenta con la facultad de ser utilizado tanto en plataformas Unix como Microsoft.

**Q&R** Quality and Reliability – Departamento de Calidad y Confiabilidad

**LF** Límite de Falla – Límite máximo permitido de ocurrencias por lote.

**SQL** Structured Query Language – Uno de los más robustos lenguajes de bases de datos en el mercado

**CBT** Computer Based Training – curso basado en computadora

**OWC** Office Web Components – componentes para web de Microsoft Office. Estos incluyen el Pivot Table, Pivot Graph y el Spreadsheet.

**VBScript** Lenguaje de programación licenciado por Microsoft para su uso con el Internet Explorer 3.0 o mayor en páginas web ASP, htm, html, DHTML y otras.

**JavaScript** Primer lenguaje de programación creado para ejecutarse con páginas ASP, htm y html.

**CGI** Common Gateway Interface – lenguaje de programación que se utiliza en servidores UNIX para la Internet e Intranet.

**VFF** Validación de Falla Funcional – pruebas de validación que se realizan para emular las condiciones de prueba en la fábrica.

**VS** Validación de Stress – pruebas de validación que se realizan para asegurar que las fallas obtenidas en QI son causadas por dicha prueba y no por el proceso de ensamblaje.

**PF** Prueba en Frío – pruebas de validación que se realizan para mantener los estándares de calidad de Intel

**FC** Falla de Calidad – unidad que falla las pruebas de calidad final que se realizan a una parte del producto que se encuentra listo para ser enviado al cliente-.

**QI** Quemado Inicial – proceso de prueba donde se aplican cargas y temperaturas excesivas al producto, para eliminar aquel cuya vida de uso útil es muy baja

**EPF** Eliminación de Prueba en Frío – eliminación de la prueba en frío

**CCF** Comprobación de Calidad Final – comprobación que se realiza a una parte del producto terminado para asegurar la calidad del proceso de ensamble y prueba.

**XML** Extensible Markup Language – lenguaje para sitios en Internet que permite una identificación de la información más flexible y adaptable. Es conocido como extensible dado que no es un lenguaje fijo como lo es el HTML. En cambio, el XML es un metalenguaje (aquel que describe otros lenguajes) que permite diseñar un lenguaje propio para una sin número de tipos diferentes de documentos.

**HTML** Hyper Text Markup Language – lenguaje utilizado como base para los sitios en Internet. Es un lenguaje estandarizado para la comunicación entre los servidores web y los exploradores.

## 8.2 Hoja de información del proyecto

### Información del estudiante

**Nombre:** Pedro Huguet Vaughan
**Cédula:** 1-1066-541 **Carné ITCR:** 9813092
**Dirección:** Lomas de Ayarco Sur
**Teléfonos:** 397-6088, 272-0448 **Fax**: 272-2398
**Email:** huguetpj@yahoo.com

### Información del proyecto

**Nombre del proyecto:** PROTAF (PROduct Test Análisis Feedback)
**Área del proyecto:** Ensamblaje y pruebas de componentes electrónicos
**Profesor Asesor:** Ing. Julio Córdoba
**Horario de Trabajo:** L-V 8:00am-5:00pm **Horario de Clases:** N/A

### Información de la empresa

**Nombre:** Componentes Intel de Costa Rica S.A.
**Zona:** Ribera de Belén, Heredia.
**Dirección:** 600mts Oeste de la Firestone, calle 129.
**Teléfono:** 296-6000 **Fax**: 298-7323
**Actividad Principal:** Ensamblaje y prueba de componentes electrónicos

### Información del encargado en la empresa

**Nombre:** Lic. y MSc Enrique Acuña
**Puesto que ocupa:** Líder de Grupo en el Dept. de Ingeniería de Producto
**Departamento:** Ingeniería de Producto
**Profesión:** Ingeniero Eléctrico **Grado:** Lic. y MSc
**Teléfono:** 298-6532
**Email:** enrique.m.acuna@intel.com

## 8.3 Resúmenes

### 8.3.1 Español

El proyecto se basa principalmente en el seguimiento de cada lote a través de los pasos de producción y en la llave de validación, y los resultados que en estos pasos se obtengan. Siendo el objetivo principal automatizar el proceso de recolección y análisis de la información de lotes mediante el uso de rutinas de software, para así reducir el tiempo de análisis y por tanto mejorar la producción.

La herramienta que se diseño se denomina PROTAF, la cual está dividida en varias etapas, debido a su gran tamaño e implicaciones, para que los practicantes la diseñaran en un plazo de dos años. La empresa tiene como objetivo final, tener un sistema estadístico con base de datos que funcione en un servidor de bajo nivel, con el propósito de realizar en forma automática la mayor cantidad de tareas del departamento de ingeniería de producción.

Lo que se requiere en esta segunda etapa del sistema, es que el sistema sea capaz de obtener la información de validaciones de los lotes que fallaron las pruebas de producción y desplegar los resultados. Para esto, primero se mejoró la interfase de usuario para así aumentar la utilización del sistema y asegurar que la información que se obtiene sea correcta. A continuación se incluyó la extracción y análisis de las validaciones que se realizan a los lotes. Este sistema está capacitado para analizar la información recolectada y permite a los usuarios seguir paso a paso las validaciones que se realizan a los lotes. En la ultima etapa del proyecto se hicieron las pruebas en ambiente, controlado del sistema para la validación de la información, antes de su instalación. El sistema es capaz de darle al usuario en forma rápida y eficaz la información de las pruebas de los lotes, las validaciones y los lotes en espera. De esta forma, se consiguió que las actividades del departamento que antes tomaban al menos medio día en realizar se realicen ahora en menos de 45 minutos.

**Palabras claves:** estadística, Visual Basic, pruebas, validaciones, SQL, UNIX, servidor web, DLL, FTP, HTML

### 8.3.2 English

The project is based on following the lots through out the steps involved in the production line and validation keys and the results that are obtained in these steps. The main objective is to automate the recollection and analysis of the lot information using software routines, and in this way reduce the time involved in analyzing the data and thus improving production.

The tool developed for this purpose is named PROTAF and is divided in several stages because of its large size and implications. This way several interns can develop the tool in a span of 2 years. The company wants, as a final objective, a statistical system based on a database which will work in a low level server with the purpose of automatically executing several tasks from the department of Product Engineering.

In this second stage of the project the system should be able to obtain the validation information from the lots that failed in the production line and display these results. It's important to highlight that all software that will be used in the project has to have a corporate license. First the user interface was improved to increase the usage of the system and thus validate the data that was being obtained for the lots. Next the extraction and analysis of the validation data was included into the system. This way the system is able to analyze the data obtained and allows the users to follow step by step the validations that are performed to the lots. In the last stage of the project the data (both production and validation data) was validated again by the users before installing the final version into the server. At the end of the project the system is capable of giving the test information to the users quickly and efficiently. Once this was accomplished, the activities that usually took half a day to performed are now done in less than 45 minutes.

**Keywords:** statistics, Visual Basic, tests, validations, SQL, UNIX, web server, DLL, FTP, HTML

## 8.4 Códigos Fuentes

### 8.4.1 Ejecutar.perl

```
use lib $workpath;
BEGIN{unshift(@INC, $workpath);}
use General;

use FileHandle;

# start of script
# every 5 minutes, this script will run cacheUpdater.perl
# Pedro Huguet Vaughan 10666377
# Internship project jan-jul 2002

#while (1==1){
$FILE = new FileHandle;
open($FILE," $workpath/Flag.cfg") or die "Flag file not found!!!\n";
@array=<$FILE>;
close $FILE;

foreach $line (@array) {
      if ($line=~"Free") {
            print "Free to run process!\n";
            open($FILE,"> $workpath/Flag.cfg");
            print $FILE "In use\n";
            close $FILE;
            system("$workpath/cacheUpdater.perl");
            open($FILE,"> $workpath/Flag.cfg");
            print $FILE "Free\n";
            close $FILE;
      }
      else {
            if ($line=~"In use") {
                  print   "Another   instance   of   process   is   already
running.\n";
            }
      }
}
print "\n";
```

### 8.4.2 PQDisplay.cgi

```
use General;

# Creator Pedro Huguet Vaughan 10666377
# Internship Project jan-jul 2002

# displays a formatted HTML document comparing prod units vrs val units

BEGIN{

      # codigo para que los errores fatales se vean en el browser
      use CGI::Carp qw(fatalsToBrowser);

}

# first, I have to obtain the message
# then, I will replace the available params
# at the end, I will print the html.

# argument order:
#0 = Total Units
#1 = Value Observed
#2 = COMPARE (> OR <)
#3 = LIMIT
#4 = Validation path
#5 = Lot number
#6 = Operation
#7 = Bin Type (1-Functional, 2-Structural, 3-Burn In)
#8 = Bin String
#9 = +Bin(s): either single bin or 'split' separated bins in combo
#10 = -Bins(s): either single bin or 'split' separated bins in combo

#################
#Constants     #
#################

$tabchr = chr(8);

#################

@arrInfo = split(",",$ARGV[0]);

opendir(PATH,$arrInfo[4]) || die "<p>Cannot opendir $arrInfo[4]</p>";

@files = `grep lcode $arrInfo[4]*`;
$count=@files;

if ($count>0) {
      if ($files[0]=~ /.*:.*/)  {
            @files = split(":",$files[0]);
            $valfile = $files[0];   #Determine validation file, for now we
suppose it to be the first one
      } else {
            @files = grep !/^\./, readdir PATH;
            $valfile = "$arrInfo[4]$files[0]";
      }
}

############################
# Start of HTML Document
############################

print "Content-type: text/html\n\n" ;
```

```
$arrInfo[8] =~ s/\\//g;

print <<EOF ;
<html>
<head><title>PQ Validation</title></head>
<body>
<h1><font size="7">PQ Data Output</font><font size="2">     PQ
owner: Jose Chico</font></h1>
<hr>
<p><b><u><font size="5">Summary:</font></u></b></p>
<p><b>Validation of Bin:</b> $arrInfo[8]</p>
EOF

#print "<p>$valfile</p>";

if ($valfile ne "") {

Summary();

print <<EOF ;
<p><b>Validation path:</b> $arrInfo[4]</p>
EOF

if ($arrInfo[7]==2) { #Structural Bin = display LYA Link
print <<EOF ;
<p>
<FORM
ACTION= $ACTION
METHOD="POST" TARGET="_blank">
<INPUT TYPE="hidden" NAME="filterlot" value="$arrInfo[5]">
<input type="hidden" name="filterfm" value="">
<input type="hidden" name="filtercustomer" value="">
<input type="hidden" name="filterlocn" value="">
<input type="hidden" name="filterfamily" value="">
<input type="hidden" name="filtertech" value="">
<input type="hidden" name="Sdate" value="">
<input type="hidden" name="edate" value="">
<b>LYA link:</b>
<INPUT TYPE="submit" VALUE="LYA">
</p>
EOF
}

print <<EOF ;
<hr>
<p><b><u><font size="5">Details:</font></u></b>
EOF

Detail();

} else {
      print "<p>No valid files found in directory.</p>";
      print "<p>$arrInfo[4]</p>";
}

print <<EOF ;
</body>
</html>
EOF

closedir(PATH);

exit(0);
```

```
##############################
#Start of subroutines
##############################

sub Summary  {
      $totalvaltrigger = 0;
      @bins = split("split",$arrInfo[9]); #Parse passed bins +
      foreach $bin (@bins) {
            @lines = `$PQpath $valfile | grep $bin | cut -c1-1 | wc -l`;
            $totalvaltrigger = $totalvaltrigger + trim($lines[0]);
      }
      @lines = `$PQpath $valfile | cut -f1`;
      $linesLen = @lines;
      $totalvalunits = $lines[$linesLen-1]/1;
      @bins = split("split",$arrInfo[10]);     #Parse passed bins -
      foreach $bin (@bins) {
            @lines = `$PQpath $valfile | grep $bin | cut -c1-1 | wc -l`;
            $totalvaltrigger = $totalvaltrigger - trim($lines[0]);
      }
      @lines = `$PQpath $valfile | wc -l`;
      $totalprodtrigger = sprintf("%.0f",$arrInfo[1]*$arrInfo[0]/100);
      $porcvaltrigger                  =                  sprintf("%.3f",
$totalvaltrigger/$arrInfo[0]*100);
      if ($arrInfo[2]=='gt') {
            $comp='>';
            if ($totalvaltrigger/$arrInfo[0]*100>$arrInfo[3]){
                  if  ($arrInfo[7]==2)  {  #Structural  Bin  =  display  LYA
Link
                        $ValResult    =    '<font    color="#FF0000">VALID
LF</font><input type="hidden" name="Result" value="%%%VALID LF&&&">';
                  } else {
                        $ValResult  =  '<font  color="#FF0000">SEND  LOT  TO
REQI</font><input  type="hidden"  name="Result"  value="%%%SEND  LOT  TO
REQI&&&">';
                  }
            }
            else {
                  $ValResult     =     '<font     color="#0000FF">INVALID
LF</font><input type="hidden" name="Result" value="%%%INVALID LF&&&">';
            }
      }
      else {
            $comp='<';
            if ($totalvaltrigger/$arrInfo[0]*100<$arrInfo[3]){
                  if  ($arrInfo[7]==2)  {  #Structural  Bin  =  display  LYA
Link
                        $ValResult    =    '<font    color="#FF0000">VALID
LF</font><input type="hidden" name="Result" value="%%%VALID LF&&&">';
                  } else {
                        $ValResult  =  '<font  color="#FF0000">SEND  LOT  TO
REQI</font><input  type="hidden"  name="Result"  value="%%%SEND  LOT  TO
REQI&&&">';
                  }
            }
            else {
                  $ValResult     =     '<font     color="#0000FF">INVALID
LF</font><input type="hidden" name="Result" value="%%%INVALID LF&&&">';
            }
      }
      print <<EOF ;
<p align="center">
<table border="1" cellpadding="0" cellspacing="0" width="60%">
  <tr>
    <td width="20%"> </td>
    <td width="20%">
```

```
      <p align="center"><b>Production</b></td>
    <td width="20%">
      <p align="center"><b>Validation</b></td>
  </tr>
  <tr>
    <td width="20%"> <b>Total Units</b></td>
    <td width="20%"><p align="center">$arrInfo[0]</p></td>
    <td width="20%"><p align="center">$totalvalunits</p></td>
  </tr>
  <tr>
    <td width="20%"> <b>Trigger % (Units)</b></td>
    <td             width="20%"><p             align="center">$arrInfo[1]
($totalprodtrigger)</p></td>
    <td           width="20%"><p           align="center">$porcvaltrigger
($totalvaltrigger)</p></td>
  </tr>
  <tr>
    <td width="33%"> <b>Limit %</b></td>
    <td width="33%"><p align="center">$comp $arrInfo[3]</p></td>
    <td width="34%"><p align="center">$comp $arrInfo[3]</p></td>
  </tr>
    <tr>
    <td width="33%"> <b>LF Result</b></td>
    <td     width="33%"><p     align="center"><font     color="#FF0000">NO
PASS</font></p></td>
    <td width="34%"><p align="center">$ValResult</p></td>
  </tr>
</table>
</p>
EOF
}

sub Detail {
      @lines = `$PQpath $valfile`;
      #chomp(@lines);
      print    '<table    border="0"    cellpadding="0"    cellspacing="1"
width="100%">';
      for ($i=0;$i<=3;$i++){
            $line = $lines[$i];
            $line =~ s/$tabchr//g;
            $temp = chr(27);
            $line =~ s/$temp\[33;44m/<font color="#FF0000">/g;
            $line =~ s/$temp\[37;40m/<font color="#0000FF">/g;
            $line =~ s/$temp\[37;41m/<font color="#0000FF">/g;
            $line =~ s/$temp\[0m/<\/font>/g;
            $line =~ s/Prod:/<\/td><td width="25%"><b>Prod:<\/b>/g;
            $line =~ s/Prog:/<\/td><td width="25%"><b>Prog:<\/b>/g;
            $line =~ s/Tester:/<\/td><td width="25%"><b>Tester:<\/b>/g;
            $line =~ s/--//g;
            print '<tr><td width="25%">'.$line.'</td></tr>';
      }
      print    '<table    border="0"    cellpadding="0"    cellspacing="1"
width="60%">';
      print '<tr>';
      print                   '<td                   width="20%"><b><font
color="#0000FF">NUM</font></b></td>';
      print                   '<td                   width="20%"><b><font
color="#0000FF">PART_ID</font></b></td>';
      print                   '<td                   width="20%"><b><font
color="#0000FF">BIN</font></b></td>';
      print '</tr>';
      $linenum = 1;
      foreach $line (@lines) {
            print '<tr>';
            if ($linenum>=6){
```

```
                  @arrLine = split(" ",$line);
                  for ($i=0;$i<=2;$i++){
                        print '<td width="20%">';
                        $arrLine[$i] =~ s/$tabchr//g;
                        print $arrLine[$i];
                        print '</td>';
                  }
            }
            print '</tr>';
            $linenum++;
      }
      print '</table>';
}

sub trim {
    my($var) = @_;
    $var =~  s/^\s+//;
    $var =~    s/\s+$//;
    $var;
}
```

### 8.4.3 VFF.cgi

```
use General;

# Creator Pedro Huguet Vaughan 10666377
# Internship Project jan-jul 2002

# displays a formatted HTML document comparing prod units vrs val units

BEGIN{

      # codigo para que los errores fatales se vean en el browser
      use CGI::Carp qw(fatalsToBrowser);

}

# first, I have to obtain the message
# then, I will replace the available params
# at the end, I will print the html.

# argument order:
#0 = Lot_Operation
#1 = Val. Path

################
#Constants     #
################

$tabchr = chr(8);

################

@arrInfo = split(",",$ARGV[0]);

opendir(PATH,$arrInfo[1]) || die "<p>Cannot opendir $arrInfo[4]</p>";

@files = `grep lcode_7044 $arrInfo[1]*`;
$count=@files;

if ($count>0) { #If found VFF val then execute VFF, else execute normal
PQ

      #Execute VFF

      if ($files[0]=~ /.*:.*/)  {
            @files = split(":",$files[0]);
            $valfile = $files[0];
      } else {
            @files = grep !/^\./, readdir PATH;
            $valfile = "$arrInfo[1]$files[0]";
      }

############################
# Start of HTML Document
############################

print "Content-type: text/html\n\n" ;

print <<EOF ;
<html>
<head><title>VFF Validation</title></head>
<body>
<h1><font  size="7">VFF  Data  Output</font><font  size="2">      
VFF owner: Isela Marin</font></h1>
```

```
<p></p>
EOF

#print "<p>$valfile</p>";

if ($valfile ne "") {

@lines = `$WVFFpath $arrInfo[0] $valfile`;
foreach $line (@lines) {
      $line =~ s/ / /g;
      if ($line=~ /.*Disposition.*/) {
            print '<p><font   color="#FF0000"
size=5><b>'.$line.'</b></font></p><input    type="hidden"    name="Result"
value="%%%INVALID LF&&&">';
      } else {
            print  '<p  style="word-spacing:  0;  text-indent:  0;  line-
height: 100%; margin:
            0"><kbd>'.$line.'</kbd></p>';
      }
}


} else {
      print "<p>No files found in directory.</p>";
      print "<p>$arrInfo[1]</p>";
}

print <<EOF ;
</body>
</html>
EOF

} else {
      print `PQDisplay.cgi @ARGV`;
}

closedir(PATH);

exit(0);
```

### 8.4.4 VS.cgi

```
use General;

# Creator Pedro Huguet Vaughan 10666377
# Internship Project jan-jul 2002

# displays a formatted HTML document comparing prod units vrs val units

BEGIN{

      # codigo para que los errores fatales se vean en el browser
      use CGI::Carp qw(fatalsToBrowser);

}

# first, I have to obtain the message
# then, I will replace the available params
# at the end, I will print the html.

# argument order:
#0 = Validation path
#1 = Lot number
#2 = Operation
#3 = Production summary (Sumnum+sumchar)

################
#Constants     #
################

$tabchr = chr(8);

################

@arrInfo = split(",",$ARGV[0]);
loadOpeConfig();
$found = 0;
$prodlot = findlot();

opendir(PATH,$arrInfo[0]) || die "<p>Cannot opendir $arrInfo[0]</p>";

@files = `grep lcode $arrInfo[0]*`;
$count=@files;

if ($count>0) {
      if ($files[0]=~ /.*:.*/)  {
            @files = split(":",$files[0]);
            $valfile = $files[0];   #Determine validation file, for now we
suppose it to be the first one
      } else {
            @files = grep !/^\./, readdir PATH;
            $valfile = "$files[0]";
      }
}

############################
# Start of HTML Document
############################

print <<EOF ;
Content-type: text/html\n\n
<HTML>
<head><title>IME Validation</title></head>
<BODY>
```

```
EOF

#print "<p>$prodlot$arrInfo[3]</p>\n";
#print "<p>$arrInfo[0]$valfile</p>\n";

print '<h1><font size="7">PQ Data Output</font><font size="2">  
  PQ owner: Jose Chico</font></h1>'."\n";

Detail();

print '<hr>';
print '<h1><font size="7">Pattern & Fail Data</font><font size="2">
    script owner: Humberto Ramirez</font></h1>'."\n";

PQPat();

print <<EOF ;
</body>
</html>
EOF

exit(0);

##############################
#Start of subroutines
##############################

sub findlot {
      if(exists($opeConfig{$arrInfo[2]})){ # if operation exists, we need
to return the path for it
            $tempstr = $opeConfig{$arrInfo[2]};
            @paths = split(",", $tempstr);
            $max = @paths; # amount of paths
            $cx = 0;
            while($cx < $max){
                  $temp = $paths[$cx];
                  $temp = trim($temp);
                  $res =  $temp . "/" . $arrInfo[1] . "_" . $arrInfo[2] .
"/";
                  if(-e $res){ # if the directory exists, then
                        $found = 5; # it's just a positive number
                        last;
                  }
                  $cx++;
            }
            if($found == 5){
                  $res;
            }
            else{
                  "-2"; # if there's no asociated directory (someone
erased the dir)
            }
      } else {
            "-1";
      }
}

sub trim {
    my($var) = @_;
    $var =~  s/^\s+//;
    $var =~    s/\s+$//;
    $var;
}

##############################
```

```
# loads in the global variables
################################
sub loadOpeConfig {
      open(CONFOPE, $operationsConfigFile);
      my $ope;
      my $paths;

      while(<CONFOPE>)
      {
            chomp($_);
            $_ = trim($_);


            if(! defined($_) ){
                next;
            }
            elsif($_ eq "")
            {
                  next;
            }
            elsif($_ eq "\n"){
                  next;
            }
            elsif(/\s*#/)  #  if  there's  a  comment  in  the  very  first
section of the text, we skip the line
            {
                  next;

            }
            else{ ($ope, $paths) = split(":",$_);
                  $ope = trim($ope);
                  $paths = trim($paths);
                  $opeConfig{$ope} = $paths;
            }
      }

        close(CONFOPE);
}

####################
# Displays PQ data
####################
sub Detail {
      my $bold = 0;
      if ($found == 5) {
            $firstline = 8;
            @lines = `$PQpath $prodlot$arrInfo[3] $arrInfo[0]$valfile`;
      } else {
            $firstline = 6;
            @lines = `$PQpath $arrInfo[0]$valfile`;
      }
      #chomp(@lines);
      print    '<table    border="0"    cellpadding="0"    cellspacing="1"
width="100%">'."\n";
      for ($i=0;$i<=4;$i++){
            $line = $lines[$i];
            $line =~ s/$tabchr//g;
            $temp = chr(27);
            $line =~ s/$temp\[33;44m/<font color="#FF0000">/g;
            $line =~ s/$temp\[37;40m/<font color="#0000FF">/g;
            $line =~ s/$temp\[37;41m/<font color="#0000FF">/g;
            $line =~ s/$temp\[0m/<\/font>/g;
            $line =~ s/Prod:/<\/td><td width="25%"><b>Prod:<\/b>/g;
            $line =~ s/Prog:/<\/td><td width="25%"><b>Prog:<\/b>/g;
            $line =~ s/Tester:/<\/td><td width="25%"><b>Tester:<\/b>/g;
```

```
            $line =~ s/--//g;
            print '<tr><td width="25%">'.$line.'</td></tr>'."\n";
      }
      print '</table>'."\n";
      print '<p> </p>'."\n";
      print    '<table    border="0"    cellpadding="0"    cellspacing="1"
width="100%">'."\n";
      print '<tr>'."\n";
      print                    '<td                    width="25%"><b><font
color="#0000FF">NUM</font></b></td>'."\n";
      print                    '<td                    width="25%"><b><font
color="#0000FF">PART_ID</font></b></td>'."\n";
      if ($found == 5) {
            print    '<td    width="25%"><b><font    color="#0000FF">PROD
BIN</font></b></td>'."\n";
      }
      print      '<td      width="25%"><b><font      color="#0000FF">VAL
BIN</font></b></td>'."\n";
      print '</tr>'."\n";
      $linenum = 1;
      foreach $line (@lines) {
            if ($linenum>=$firstline){
                  print '<tr>'."\n";
                  @arrLine = split(" ",$line);
                  foreach $info (@arrLine){
                        $info =~ s/$tabchr//g;
                        $temp = chr(27);
                        $info =~ s/$temp\[31m/<font color="#FF0000">/g;
                        $info =~ s/$temp\[33;44m/<font color="#FF0000">/g;
                        $info =~ s/$temp\[37;40m/<font color="#0000FF">/g;
                        $info =~ s/$temp\[37;41m/<font color="#0000FF">/g;
                        $info =~ s/$temp\[0m/<\/font>/g;
                        $info =~ s/$tabchr//g;
                        if ($info ne "</font>") {
                              if ($info ne "###") {
                                    print '<td width="25%">';
                                    if ($bold == 1) {
                                          print                     '<font
color="#FF0000"><b>';
                                          $bold = 0;
                                    }
                                    print $info;
                                    print '</td>';
                              } else {
                                    $bold = 1;
                              }
                        }
                  }
                  print '</tr>'."\n";
            }
            $linenum++;
      }
      print '</table>'."\n";
}

####################
# Displays PQ Pattern data
####################
sub PQPat {
      print    '<table    border="0"    cellpadding="1"    cellspacing="1"
width="100%">'."\n";
      @lines = `$PQPatpath $arrInfo[0]$valfile`;
      foreach $line (@lines) {
            print '<tr>';
            @arrline = split(" ",$line);
```

```
            if ($arrline[1] > 799 ) {
                  foreach $info (@arrline) {
                        print '<td width="15%">';
                        print $info;
                        print '</td>';
                  }
                  print '</tr>';
            }
      }
}
```

### 8.4.5 PF.cgi

```
use General;

# Creator Pedro Huguet Vaughan 10666377
# Internship Project jan-jul 2002

# displays a formatted HTML document comparing prod units vrs val units

BEGIN{

      # codigo para que los errores fatales se vean en el browser
      use CGI::Carp qw(fatalsToBrowser);

}

# first, I have to obtain the message
# then, I will replace the available params
# at the end, I will print the html.

# argument order:
#0 = Validation path
#1 = Lot number
#2 = Operation
#3 = Production summary (Sumnum+sumchar)

################
#Constants     #
################

$tabchr = chr(8);

################

@arrInfo = split(",",$ARGV[0]);
loadOpeConfig();
$found = 0;
$prodlot = findlot();

opendir(PATH,$arrInfo[0]) || die "<p>Cannot opendir $arrInfo[0]</p>";

@files = `grep tempr $arrInfo[0]*`;
$count=@files;

if ($count>0) {
      if ($files[0]=~ /.*:.*/)  {
            for ($i=0;$i<=$count-1;$i++) {
                  @info = split(":",$files[$i]);
                  @data = split("_",$info[1]);
                  $TestFile{$info[0]} = $data[2];
            }
            $valfile = "";
            $count = 0; #count the number of val files
            foreach $temp (sort { $TestFile{$a} cmp $TestFile{$b} } keys
%TestFile) {
                  $valfile = $valfile . " " . $temp;
                  $count++;
            }
      } else {
            @files = grep !/^\./, readdir PATH;
            $valfile = "$arrInfo[0]$files[0]";
      }
}
```

```
############################
# Start of HTML Document
############################

print <<EOF ;
Content-type: text/html\n\n
<HTML>
<head><title>CSM Validation</title></head>
<BODY>
EOF

#print "<p>$prodlot$arrInfo[3]</p>\n";
#print "<p>$valfile</p>\n";

print  '<h1><font  size="7">PQ  Data  Output</font><font  size="2">   
  PQ owner: Jose Chico</font></h1>'."\n";

Detail();

print <<EOF ;
</body>
</html>
EOF

exit(0);

############################
#Start of subroutines
############################

sub findlot {
      if(exists($opeConfig{$arrInfo[2]})){ # if operation exists, we need
to return the path for it
            $tempstr = $opeConfig{$arrInfo[2]};
            @paths = split(",", $tempstr);
            $max = @paths; # amount of paths
            $cx = 0;
            while($cx < $max){
                  $temp = $paths[$cx];
                  $temp = trim($temp);
                  $res =  $temp . "/" . $arrInfo[1] . "_" . $arrInfo[2] .
"/";
                  if(-e $res){ # if the directory exists, then
                        $found = 5; # it's just a positive number
                        last;
                  }
                  $cx++;
            }
            if($found == 5){
                  $res;
            }
            else{
                  "-2";  #  if  there's  no  asociated  directory  (someone
erased the dir)
                  }
      } else {
            "-1";
      }
}

sub trim {
    my($var) = @_;
    $var =~  s/^\s+//;
    $var =~    s/\s+$//;
    $var;
```

```
}

################################
# loads in the global variables
################################
sub loadOpeConfig {
      open(CONFOPE, $operationsConfigFile);
      my $ope;
      my $paths;

      while(<CONFOPE>)
      {
            chomp($_);
            $_ = trim($_);


            if(! defined($_) ){
                next;
            }
            elsif($_ eq "")
            {
                  next;
            }
            elsif($_ eq "\n"){
                  next;
            }
            elsif(/\s*#/)  #  if  there's  a  comment  in  the  very  first
section of the text, we skip the line
            {
                  #print "coment found: $_ \n" ;
                  next;

            }
            else{ # here, we upload the info to the operation config hash
                  ($ope, $paths) = split(":",$_);
                  #print "Ope found: $ope, with paths: $paths \n";
                  $ope = trim($ope);
                  $paths = trim($paths);
                  #if(! (-e $paths)){
                  #     print ":-|Error: path not found: $paths\n";
                  #}
                  $opeConfig{$ope} = $paths;
                  # we now have the operation and the configuration


            }
      }

        close(CONFOPE);
}

####################
# Displays PQ data
####################
sub Detail {
      my $bold = 0;
      if (($found == 5) && ($arrInfo[3]!="")) {
            #print $count;
            $firstline = ($count + 1) * 2 + 4;
            #print "$PQpath $prodlot$arrInfo[3] $valfile";
            @lines = `$PQpath $prodlot$arrInfo[3] $valfile`;
      } else {
            #print "$PQpath $valfile\n";
            @lines = `$PQpath $valfile`;
            $firstline = $count * 2 + 4;
```

```
	}

	#chomp(@lines);
	print	'<table	border="0"	cellpadding="0"	cellspacing="1"
width="100%">'."\n";
	for ($i=0;$i<=$firstline-4;$i++){
		$line = $lines[$i];
		$line =~ s/$tabchr//g;
		$temp = chr(27);
		$line =~ s/$temp\[33;44m/<font color="#FF0000">/g;
		$line =~ s/$temp\[37;40m/<font color="#0000FF">/g;
		$line =~ s/$temp\[37;41m/<font color="#0000FF">/g;
		$line =~ s/$temp\[0m/<\/font>/g;
		$line =~ s/Prod:/<\/td><td width="25%"><b>Prod:<\/b>/g;
		$line =~ s/Prog:/<\/td><td width="25%"><b>Prog:<\/b>/g;
		$line =~ s/Tester:/<\/td><td width="25%"><b>Tester:<\/b>/g;
		$line =~ s/--//g;
		print '<tr><td width="25%">'.$line.'</td></tr>'."\n";
	}
	print '</table>'."\n";
	print '<p> </p>'."\n";
	print	'<table	border="0"	cellpadding="0"	cellspacing="1"
width="100%">'."\n";
	print '<tr>'."\n";
	print			'<td			width="15%"><b><font
color="#0000FF">NUM</font></b></td>'."\n";
	print			'<td			width="15%"><b><font
color="#0000FF">PART_ID</font></b></td>'."\n";
	if (($found == 5) && ($arrInfo[3]!="")) {
		print	'<td	width="15%"><b><font	color="#0000FF">PROD
BIN</font></b></td>'."\n";
	}
	foreach $temp (sort { $TestFile{$a} cmp $TestFile{$b} } keys
%TestFile) {
		print		'<td		width="15%"><b><font
color="#0000FF">'.$TestFile{$temp}.' oC</font></b></td>'."\n";
	}
	print '</tr>'."\n";
	$linenum = 1;
	foreach $line (@lines) {
		if ($linenum>=$firstline){
			print '<tr>'."\n";
			@arrLine = split(" ",$line);
			foreach $info (@arrLine){
				$info =~ s/$tabchr//g;
				$temp = chr(27);
				$info =~ s/$temp\[31m/<font color="#FF0000">/g;
				$info =~ s/$temp\[30;47m/<font color="#FF0000">/g;
				$info =~ s/$temp\[33;44m/<font color="#FF0000">/g;
				$info =~ s/$temp\[37;40m/<font color="#0000FF">/g;
				$info =~ s/$temp\[37;41m/<font color="#0000FF">/g;
				$info =~ s/$temp\[0m/<\/font>/g;
				$info =~ s/$tabchr//g;
				if	(($info	ne	"</font>")	&&	($info	ne
'</font><font color="#FF0000">')) {
					if ($info ne "###") {
						print '<td width="15%">';
						if ($bold == 1) {
							print			'<font
color="#FF0000"><b>';
							$bold = 0;
						}
						print $info;
						print '</td>';
					} else {
```

```
                                        $bold = 1;
                                    }
                                }
                            }
                            print '</tr>'."\n";
                        }
                        $linenum++;
                }
                print '</table>'."\n";
}
```

### 8.4.6 FC.cgi

```
use General;

# Creator Pedro Huguet Vaughan 10666377
# Internship Project jan-jul 2002

# displays a formatted HTML document comparing prod units vrs val units

BEGIN{

      # code so that fatal errors can be shown in browser
      use CGI::Carp qw(fatalsToBrowser);

}

# first, I have to obtain the message
# then, I will replace the available params
# at the end, I will print the html.

# argument order:
#0 = Validation path
#1 = Lot number
#2 = Operation
#3 = Production summary (Sumnum+sumchar)

################
#Constants     #
################

$tabchr = chr(8);

################

@arrInfo = split(",",$ARGV[0]);
loadOpeConfig();
$found = 0;
$prodlot = findlot();

opendir(PATH,$arrInfo[0]) || die "<p>Cannot opendir $arrInfo[0]</p>";

@files = `grep lcode $arrInfo[0]*`;
$count=@files;

if ($count>0) {
      if ($files[0]=~ /.*:.*/)  {
            $valfile = "";
            for ($i=0;$i<=$count-1;$i++) {
                  @info = split(":",$files[$i]);
                  $valfile = $valfile . " " . $info[0]
            }
      } else {
            @files = grep !/^\./, readdir PATH;
            $valfile = "$arrInfo[0]$files[0]";
      }
}

############################
# Start of HTML Document
############################

print <<EOF ;
Content-type: text/html\n\n
<HTML>
```

```
<head><title>ITR Validation</title></head>
<BODY>
EOF

#print "<p>prod $prodlot$arrInfo[3]</p>\n";
#print "<p>val $valfile</p>\n";
#print "<p>count $count</p>\n";

print  '<h1><font  size="7">PQ  Data  Output</font><font  size="2">   
  XS owner: Jose Chico</font></h1>'."\n";

Detail();

print <<EOF ;
<p> </p>
<hr>
<h1>Validation directory contents</h1>
<p><b>Click on any file to see contents</b><p>
EOF

ListDir();

print <<EOF ;
</body>
</html>
EOF

exit(0);

##############################
#Start of subroutines
##############################

sub findlot {
      if(exists($opeConfig{$arrInfo[2]})){ # if operation exists, we need
to return the path for it
            $tempstr = $opeConfig{$arrInfo[2]};
            @paths = split(",", $tempstr);
            $max = @paths; # amount of paths
            $cx = 0;
            while($cx < $max){
                  $temp = $paths[$cx];
                  $temp = trim($temp);
                  $res =  $temp . "/" . $arrInfo[1] . "_" . $arrInfo[2] .
"/";
                  if(-e $res){ # if the directory exists, then
                        $found = 5; # it's just a positive number
                        last;
                  }
                  $cx++;
            }
            if($found == 5){
                  $res;
            }
            else{
                  "-2";  #  if  there's  no  asociated  directory  (someone
erased the dir)
            }
      } else {
            "-1";
      }
}

################################
################################
```

```
sub trim {
    my($var) = @_;
    $var =~  s/^\s+//;
    $var =~    s/\s+$//;
    $var;
}

################################
# loads in the global variables
################################
sub loadOpeConfig {
      open(CONFOPE, $operationsConfigFile);
      my $ope;
      my $paths;

      while(<CONFOPE>)
      {
            chomp($_);
            $_ = trim($_);


            if(! defined($_) ){
                next;
            }
            elsif($_ eq "")
            {
                  next;
            }
            elsif($_ eq "\n"){
                  next;
            }
            elsif(/\s*#/)  #  if  there's  a  comment  in  the  very  first
section of the text, we skip the line
            {
                  #print "coment found: $_ \n" ;
                  next;

            }
            else{ # here, we upload the info to the operation config hash
                  ($ope, $paths) = split(":",$_);
                  #print "Ope found: $ope, with paths: $paths \n";
                  $ope = trim($ope);
                  $paths = trim($paths);
                  #if(! (-e $paths)){
                  #     print ":-|Error: path not found: $paths\n";
                  #}
                  $opeConfig{$ope} = $paths;
                  # we now have the operation and the configuration


            }
      }

        close(CONFOPE);
}

####################
# Displays PQ data
####################
sub Detail {
      my $bold = 0;
      if (($found == 5) && ($arrInfo[3]!="")) {
            $firstline = ($count + 1) * 2 + 4;
            #print "$PQpath $prodlot$arrInfo[3] $valfile";
            @lines = `$PQpath $prodlot$arrInfo[3] $valfile`;
```

```
      } else {
            #print "$PQpath $valfile\n";
            @lines = `$PQpath $valfile`;
            $firstline = $count * 2 + 4;
      }

      #chomp(@lines);
      print    '<table    border="0"    cellpadding="0"    cellspacing="1"
width="100%">'."\n";
      for ($i=0;$i<=$firstline-4;$i++){
            $line = $lines[$i];
            $line =~ s/$tabchr//g;
            $temp = chr(27);
            $line =~ s/$temp\[33;44m/<font color="#FF0000">/g;
            $line =~ s/$temp\[37;40m/<font color="#0000FF">/g;
            $line =~ s/$temp\[37;41m/<font color="#0000FF">/g;
            $line =~ s/$temp\[0m/<\/font>/g;
            $line =~ s/Prod:/<\/td><td width="25%"><b>Prod:<\/b>/g;
            $line =~ s/Prog:/<\/td><td width="25%"><b>Prog:<\/b>/g;
            $line =~ s/Tester:/<\/td><td width="25%"><b>Tester:<\/b>/g;
            $line =~ s/--//g;
            print '<tr><td width="25%">'.$line.'</td></tr>'."\n";
      }
      print '</table>'."\n";
      print '<p> </p>'."\n";
      print    '<table    border="0"    cellpadding="0"    cellspacing="1"
width="100%">'."\n";
      print '<tr>'."\n";
      print                    '<td                    width="15%"><b><font
color="#0000FF">NUM</font></b></td>'."\n";
      print                    '<td                    width="15%"><b><font
color="#0000FF">PART_ID</font></b></td>'."\n";
      if (($found == 5) && ($arrInfo[3]!=""))  {
            print    '<td    width="15%"><b><font    color="#0000FF">PROD
BIN</font></b></td>'."\n";
      }
      for ($i=0;$i<=$count-1;$i++) {
            print    '<td    width="15%"><b><font    color="#0000FF">VAL
BIN</font></b></td>'."\n";
      }
      print '</tr>'."\n";
      $linenum = 1;
      foreach $line (@lines) {
            if ($linenum>=$firstline){
                  print '<tr>'."\n";
                  @arrLine = split(" ",$line);
                  foreach $info (@arrLine){
                        $info =~ s/$tabchr//g;
                        $temp = chr(27);
                        $info =~ s/$temp\[31m/<font color="#FF0000">/g;
                        $info =~ s/$temp\[30;47m/<font color="#FF0000">/g;
                        $info =~ s/$temp\[33;44m/<font color="#FF0000">/g;
                        $info =~ s/$temp\[37;40m/<font color="#0000FF">/g;
                        $info =~ s/$temp\[37;41m/<font color="#0000FF">/g;
                        $info =~ s/$temp\[0m/<\/font>/g;
                        $info =~ s/$tabchr//g;
                        if   (($info   ne   "</font>")   &&   ($info   ne
'</font><font color="#FF0000">')) {
                              if ($info ne "###") {
                                    print '<td width="15%">';
                                    if ($bold == 1) {
                                          print                    '<font
color="#FF0000"><b>';
                                          $bold = 0;
                                    }
```

```
                                    print $info;
                                    print '</td>';
                              } else {
                                    $bold = 1;
                              }
                        }
                  }
                  print '</tr>'."\n";
            }
            $linenum++;
      }
      print '</table>'."\n";
}

####################
# Displays dir list
####################
sub ListDir {
      my @files = split("\n",`ls -l $arrInfo[0]`);
      foreach $file (@files) {
            if ($file =~ /-.*/) {
                  @info = split(" ",$file);
                  print "<p>".$info[4]." ".$info[5]." ".$info[6]." - ";
                  print    '<a    href="cat.cgi?'.$arrInfo[0].$info[7].'"
target="bottom">'.$info[7].'</a></p>';
            }
      }
}
```

### 8.4.7 catenation.cgi

```
use General;

# Creator Pedro Huguet Vaughan 10666377
# Internship Project jan-jul 2002

# displays a formatted HTML document comparing prod units vrs val units

BEGIN{

      # code so that fatal errors can be shown in browser
      use CGI::Carp qw(fatalsToBrowser);

}

# first, I have to obtain the message
# then, I will replace the available params
# at the end, I will print the html.

# argument order:
#0 = Path to display

#################
#Constants     #
#################

$tabchr = chr(8);

#################

@arrInfo = split(",",$ARGV[0]);

@file = split("\n",`cat $arrInfo[0]`);

#############################
# Start of HTML Document
#############################

print <<EOF ;
Content-type: text/html\n\n
<HTML>
<BODY>
EOF

print  '<p  style="word-spacing:  0;  text-indent:  0;  line-height:  100%;
margin: 0"><b>Content for the file:
</b></p>'.$arrInfo[0].'</p>'."\n";
print '<font face="Courier">';
foreach $line (@file) {
      print  '<p  style="word-spacing:  0;  text-indent:  0;  line-height:
100%; margin: 0">'.$line.'</p>'
}

print <<EOF ;
</font>
</body>
</html>
EOF

exit(0);

###############################
###############################
```

```
sub trim {
    my($var) = @_;
    $var =~  s/^\s+//;
    $var =~    s/\s+$//;
    $var;
}
```