Título: “Mecanismo para el control de la integridad de las bibliotecas dinámicas del sistema SCADA Guardián del Alba.”

Trabajo de Diploma para optar por el título de Ingeniera en Ciencias Informáticas

Autor:

Liuba Sosa Fridrij

Tutores:

Ing. José Manuel Batista Viltre

Ing. Lisandra González Serrano

Co Tutores:

Ing. Ernesto Leyva Barrero

Msc. Mileidy García Rodríguez

La Habana

“Año 55 de la Revolución”

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al Centro de Informática Industrial de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____	_____
Firma del autor	Firma del tutor
Liuba Sosa Fridrij	**Ing. José Manuel Batista Viltre**

Datos de contacto

Ing. José Manuel Batista Viltre

Graduado de Ingeniería en Ciencias Informáticas en la Universidad de las Ciencias Informáticas (UCI). Actualmente se desempeña como Arquitecto de software del SCADA SAINUX con 1 año de experiencia.

Ing. Ernesto Leyva Barrero

Graduado de Ingeniería en Ciencias Informáticas en la Universidad de las Ciencias Informáticas. Actualmente se desempeña como jefe de la línea de HMI Web del Centro de Ingeniería Industrial de la UCI con 8 años de experiencia.

Msc. Mileidy García Rodríguez

Graduada de Licenciada en Educación en la especialidad de Marxismo Leninismo e Historia. MSc. en Estudios Sociales de Ciencia y Tecnología. Actualmente se desempeña como VDIP de la Facultad 5 con 15 años de experiencia.

Ing. Lisandra González Serrano.

Graduada de Ingeniera en Ciencias Informáticas en la Universidad de las Ciencias Informáticas. Actualmente se desempeña como especialista general del departamento Integración y Despliegue del Centro de Informática Industrial (CEDIN) con 3 años de experiencia.

Agradecimientos

Le agradezco este logro alcanzado…

A todas aquellas personas que de una forma u otra hicieron mi vida más fácil en la universidad.

A mis tutores y cotutores, por brindarme siempre su ayuda desinteresada, por las críticas constructivas que hacen de este un mejor trabajo, por hacerme sentir más segura y por confiar en mí.

A José Manuel Batista, que no lo menciono como mi tutor, porque más que eso hoy es mi amigo, es una lástima no haberte conocido antes, eres genial, y como tutor, nadie pudo ser mejor, tu creíste en mi cuando yo pensé que no podría, tu ayuda me permitió llegar hasta donde estoy hoy.

A toda mi familia, que siempre me apoyó en todo momento y me dan constantemente su amor.

A todos todos mis amigos como Ronniel, Fidel, Jorgito, Norberto, Lianny, Joyce, Dayle, Alexis y a las muchachitas del 92108, por soportarme a diario, saben que me tienen en sus vidas para siempre, gracias, porque por ustedes mi corazón ahora es más grande.

A Dayanis, por ser mi amiga y dejarme ser parte de su vida, que aunque su cara siempre diga lo contrario sé que tengo un lugarcito en su corazón.

A Made por ser mi hermanita y preocuparse por mí siempre, por formar parte de mi familia y dejarme ser parte de la suya.

A mi hermano por todo el amor que siempre me dio, que aunque hoy no está aquí, sé que me acompañará toda la vida porque aunque siempre nos fajábamos no podemos vivir el uno sin el otro.

A mi mamita y mi papito por ser lo más grande que tengo en este mundo, gracias a los dos, sin ustedes nada de esto hoy sería posible, por querer cumplir mi sueño de ser como ustedes es que he llegado hasta aquí y tendré que seguir superándome para poder alcanzarlos, por estar siempre para mi, por soportarme aun cuando ni yo misma me soporto, por consentirme siempre y por enseñarme que el amor verdadero si existe, los amo.

Dedicatoria

Les dedico este triunfo a mi mamá, mi papá y mi hermano, por ser las personas más importantes de mi vida, y por las cuales me levanto cada día queriendo ser alguien mejor.

Resumen

El presente trabajo surge por la necesidad de crear un mecanismo capaz de controlar la integridad de las bibliotecas dinámicas de los diferentes módulos del sistema SCADA Guardián del GALBA (GALBA).

Para realizar la propuesta planteada se trazó como objetivo, desarrollar un mecanismo integrado, para los distintos módulos del SCADA Guardián del Alba, que permita comprobar la integridad de las bibliotecas dinámicas.

El mecanismo desarrollado fue validado mediante pruebas al sistema, cumpliéndose satisfactoriamente cada una de las historias de usuario propuestas por el cliente. El uso del mecanismo de control de la integridad de las bibliotecas dinámicas de los módulos del sistema SCADA GALBA permitirá detectar posibles cambios o modificaciones en las bibliotecas utilizadas y recuperar tiempo perdido a la hora de buscar posibles errores en el sistema. Además dotará al equipo de desarrollo del sistema SCADA de un mecanismo sencillo y centralizado que agilizará los procesos de compilación y ejecución del software.

Palabras clave: biblioteca dinámica, biblioteca estática, GALBA, integridad, SCADA.

Índice de contenido

Índice de tablas

Índice de ilustraciones

INTRODUCCIÓN

Históricamente aumentar los niveles de eficiencia, minimizar los costos y optimizar los procesos de producción, ha resultado importante para la industria. En la actualidad uno de los mecanismos que puede dar solución a estas necesidades es la automatización de los distintos procesos que componen la cadena de producción.

Los sistemas de supervisión, adquisición y control de datos, más conocidos por sus siglas en inglés: SCADA (*Supervisory Control and Data Acquisition*) constituyen el núcleo de la mayoría de estos sistemas automatizados. Estos son partes integrales de la mayoría de los ambientes industriales complejos o geográficamente dispersos, ya que pueden recoger la información de una gran cantidad de fuentes rápidamente y presentarla ante un operador.

Un sistema SCADA es una aplicación software especialmente diseñada para que pueda funcionar sobre ordenadores en el control de producción, proporcionando comunicación con los dispositivos de campo y controlando el proceso de forma automática desde la pantalla del operador. Además, provee a diversos usuarios toda la información que se genera en las industrias, proceso productivo, control de calidad, supervisión y mantenimiento.

En la Universidad de la Ciencias Informáticas se desarrollan numerosos proyectos productivos sobre automatización de empresas e instituciones tanto dentro del país como en el exterior. Tal es el caso del proyecto SCADA Guardián del Alba (GALBA), desarrollado por el Centro de Informática Industrial (CEDIN) perteneciente a la Facultad 5, el cual se ha venido desplegando en las instalaciones productivas de la empresa venezolana Petróleos de Venezuela (PDVSA).

El GALBA es un sistema diseñado para operar en tiempo real con una arquitectura distribuida, compuesto por módulos que al ejecutarse posibilitan el funcionamiento del sistema como un todo. La distribución de estos permite obtener configuraciones escalables según los requisitos que presente cada aplicación. El sistema SCADA GALBA utiliza bibliotecas dinámicas que contienen funciones para lenguajes de uso general las cuales facilitan personalizar de manera muy amplia la aplicación que se desea desarrollar.

Una condición importante para el mejor desarrollo y mantenimiento de las bibliotecas dinámicas es que los servicios a programas independientes que estas proporcionan, permitan que el código y los datos se compartan y puedan modificarse de forma modular. Sin embargo realizar cambios de manera incorrecta a estas bibliotecas, puede ocasionar inestabilidad o mal funcionamiento del sistema. Una de las dificultades que afronta actualmente el SCADA GALBA, es no garantizar la integridad de las bibliotecas tanto en tiempo de compilación como de

ejecución ya que las bibliotecas utilizadas pueden ser modificadas por el cliente o por un agente externo sin que el sistema pueda detectar estos cambios. Asimismo ocasionaría el retraso en el mantenimiento del sistema debido a la demora en la busca de falsos errores y provocar el descontento en los usuarios del sistema.

Es por ello que se define como **problema científico**: ¿Cómo controlar la integridad de las bibliotecas dinámicas de los distintos módulos del sistema SCADA GALBA?
Una vez planteado el problema científico, se deriva como **objeto de estudio** los mecanismos para controlar la integridad de los datos.
Se define como **objetivo general** de la investigación, desarrollar un mecanismo integrado, para los distintos módulos del SCADA Guardián del Alba, que permita comprobar la integridad de las bibliotecas dinámicas.
Definiéndose como **campo de acción** los métodos de verificación de la integridad de los datos en Software Libre.
Se concibe como **idea a defender** que**:**

Con la realización de esta investigación se proporcionará un mecanismo que te permita controlar la integridad de las bibliotecas dinámicas, permitiendo de esta manera conocer las causas de fallos del software en cuanto al funcionamiento de dichas bibliotecas.

Para dar solución al objetivo propuesto se definen las siguientes **Tareas de Investigación**:

1. Elaboración del marco teórico de la investigación a partir del estado del arte existente sobre el tema.
2. Análisis y selección de las herramientas y tecnologías a utilizar.
3. Elaboración de la documentación correspondiente al análisis y diseño del mecanismo de verificación de la integridad de las bibliotecas.
4. Obtención del código generado a partir de los requerimientos especificados.
5. Integración del mecanismo de comprobación a los diferentes módulos del SCADA Guardián del Alba.
6. Realización de pruebas al software con el objetivo de garantizar su correcto funcionamiento.

Para el cumplimiento de estos objetivos se utilizan varios **métodos y técnicas en la búsqueda y procesamiento de la información** como son:

Métodos teóricos:

- **Método analítico-sintético:** Se utiliza este método para el estudio de los conceptos empleados en la verificación de la integridad de las bibliotecas de los módulos del SCADA, realizando un análisis de todos los documentos elaborados por desarrolladores, para la extracción y síntesis de los elementos más importantes.
- **Análisis histórico–lógico:** Se utiliza este método para realizar el estudio del estado del arte acerca del tema en cuestión, analizando los antecedentes y las tendencias actuales en cuanto a la evolución y desarrollo de los mecanismos de verificación de integridad de los datos.
- **Modelación:** Utilizado para definir y representar gráficamente las funcionalidades del sistema usando el Lenguaje Unificado de Modelado (UML).

Métodos empíricos:

- **Entrevistas:** Se utilizan para la recolección de la información y el conocimiento existente en los especialistas vinculados a los temas de automatización industrial.
- **Experimentos:** Empleados en la elaboración de prototipos funcionales, con el objetivo de comprobar la efectividad de la implementación de las funcionalidades.

El presente documento está estructurado en cuatro capítulos:

Capítulo 1: Fundamentación teórica. Describe el estado del arte actual. Se tratarán las principales características y conceptos asociados a la integridad de las bibliotecas dinámicas de los cuales dependen los diferentes módulos de los sistemas SCADA.

Capítulo 2: Herramientas y tecnologías actuales. Se explican las metodologías y tecnologías actuales a considerar para realizar la selección de aquellas que se van a utilizar para el desarrollo de la aplicación teniendo en cuenta los requisitos del usuario.

Capítulo 3: Descripción de la solución propuesta. Se describe el flujo actual de eventos, se elabora una propuesta del sistema y se especifican las historias de usuario, con sus correspondientes tareas.

Capítulo 4: Desarrollo y pruebas. Se realizará la implementación de la aplicación y se realizarán las pruebas necesarias al sistema.

Capítulo 1: Fundamentación teórica

1.1 Introducción

En el presente capítulo se hace una introducción a los conceptos fundamentales relacionados con la investigación, se definen las principales características y conceptos asociados a la integridad de las bibliotecas dinámicas de las cuales dependen los diferentes módulos de los sistemas SCADA. Para ello se abordan definiciones generales de estos sistemas.

1.2 Sistemas SCADA

SCADA es un sistema basado en aplicaciones de software diseñado para funcionar sobre computadores en el control de producción, proporcionando comunicación con los dispositivos de campo (*Controladores Lógicos Programables-PLC*, *Unidad Terminal Remota-RTU*) y controlando el proceso de forma automática desde la pantalla del computador. Permitiendo realizar a distancia operaciones de control, supervisión y registro de datos de cualquier proceso industrial. Los sistemas SCADA mejoran la eficacia del proceso de monitoreo y control proporcionando la información oportuna para poder tomar decisiones operacionales apropiadas. De igual forma, ya que cuenta con información (alarmas, históricos, paradas) de primera mano de lo que ocurre u ocurrió en el proceso, permite la integración con otras herramientas del negocio.

Los primeros sistemas SCADA se caracterizaban por ser monolíticos, las generaciones actuales de estos sistemas tienden a ser distribuidos. Un SCADA distribuido es aquel sistema en que sus componentes tanto de hardware como de software se encuentran conectados en una red de área local, cada uno de estos, con una función específica dentro del sistema. Los componentes, mediante la comunicación y coordinación entre sí, permiten que el sistema funcione como una entidad física, cuyo objetivo fundamental es lograr el control y supervisión de los procesos industriales.

1.2.1 Sistema SCADA Guardián del ALBA (GALBA)

El GALBA es un sistema distribuido en módulos que trabajan de manera conjunta posibilitando el funcionamiento del sistema como un todo. Estos módulos se encuentran interconectados a través de un software para la distribución de los servicios en la red, conocido como software de comunicación entre aplicaciones. La distribución de los módulos existentes en el SCADA permite obtener configuraciones escalables en dependencia de los requisitos que presente

cada aplicación. Es un sistema en tiempo real y presenta una arquitectura distribuida. Está dividido en varios subsistemas entre los que se encuentran:

- **Comunicación:** Es la capa de software, que se encarga de la comunicación entre los diferentes módulos que forman parte del sistema. Este módulo tiene como finalidad proporcionar la capa de comunicación de alto nivel, tanto sincrónica, como asincrónica, para la comunicación de todos los módulos que conforman el sistema SCADA.
- **Adquisición:** Es el encargado de la adquisición, recepción, procesamiento y distribución de los datos provenientes del campo.
- **Configuración:** El servicio de configuración está formado por un grupo de componentes cada uno con una tarea específica y una base de datos que contendrá las configuraciones de cada uno de los módulos que conformen el proyecto activo. Es el encargado de almacenar, persistir y suministrar la información base para el funcionamiento de los demás módulos del SCADA.
- **Almacenamiento de datos históricos:** Es el encargado de almacenar la información del sistema para que posteriormente pueda ser empleada, por ejemplo, en generación de reportes, tendencias o en gestión de producción. La base de datos histórica (BDH) contendrá la información persistente de los datos recolectados de los dispositivos.
- **Seguridad:** Proporciona las funcionalidades necesarias para garantizar el trabajo autorizado a usuarios y módulos, además brinda las herramientas necesarias para la protección contra ataques maliciosos o involuntarios al sistema por parte de personas o recursos, tales como fallas de energía, problemas de red o servidores.
- **Visualización o HMI:** Se encarga de representar en un ordenador, los procesos que ocurren en el campo en tiempo real, muestra los componentes implicados, los sensores, las estaciones remotas, y el sistema de comunicación dándole al operador diferentes niveles de control en dependencia de sus niveles de privilegios. Este módulo permite al operador el contacto directo con el sistema y realizar la supervisión y el control del proceso en general.

Cada uno de estos módulos depende para su correcto funcionamiento de bibliotecas dinámicas. Algunos SCADA ofrecen bibliotecas de funciones para lenguajes de uso general

que permiten personalizar de manera muy amplia la aplicación que desee realizarse con dicho sistema.

1.3 Bibliotecas

Según se van desarrollando programas de ordenador, se puede notar que algunas partes del código se utilizan en muchos de ellos. Por ejemplo, si se tienen varios programas que utilizan números complejos o las funciones de suma y resta que son comunes, sería estupendo poder guardar esas funciones en un directorio separado de los programas concretos y tenerlas ya compiladas, de forma que se puedan usar siempre que sea necesario, lo que trae consigo ventajas como son:

- No tener que volver a escribir el código.
- Se ahorrará el tiempo de compilar ese código que ya está compilado.
- Además, se sabe que mientras se desarrolla un programa, se prueba y se corrige, hay que compilarlo muchas veces.
- El código ya compilado estará probado y será fiable.

Para dar cumplimiento a estas ventajas se crean bibliotecas, las cuales están conformadas por una o más funciones que están ya compiladas y preparadas para ser utilizadas en cualquier programa que necesite de ellas.

La mayoría de los sistemas operativos modernos proporcionan bibliotecas que implementan los servicios del sistema. De esta manera, estos servicios se han convertido en una "materia prima" que cualquier aplicación moderna espera que el sistema operativo ofrezca. Como tal, la mayor parte del código utilizado por las aplicaciones modernas se ofrece en estas bibliotecas.

En Linux podemos hacer dos tipos de bibliotecas: estáticas y dinámicas.

Bibliotecas Estáticas. Se enlazan al compilar, es decir, quedan "dentro" del ejecutable final. En Windows tienen la extensión .lib, mientras que en Linux tienen la extensión .a.

Bibliotecas Dinámicas. Se enlazan al ejecutar, es decir, el sistema operativo debe encontrarlas al ejecutar el programa. Si una aplicación se instaló bien, el sistema operativo no debe tener problema para encontrarla. En Windows tienen la extensión .dll y mientras que en Linux tienen la extensión .so.

1.3.1 Bibliotecas estáticas

Una biblioteca estática es una biblioteca que "se copia" en nuestro programa cuando lo compilamos. Una vez que tenemos el ejecutable de nuestro programa, la biblioteca no sirve

para nada (sirve para otros futuros proyectos). Podríamos borrarla y nuestro programa seguiría funcionando, ya que tiene copia de todo lo que necesita. Sólo se copia aquella parte de la biblioteca que se vaya a utilizar. Por ejemplo, si la biblioteca tiene dos funciones y nuestro programa sólo llama a una, sólo se copia esa función. Esto puede ser de mucha importancia en aplicaciones muy grandes, ya que el ejecutable debe ser cargado en memoria de una sola vez. [5]

1.3.2 Bibliotecas dinámicas

Una biblioteca dinámica no se copia en el programa al compilarlo, por lo que el ejecutable será más pequeño, pero si se mueve el ejecutable a otra máquina las bibliotecas tienen que ir con él, ya que cada vez que el código necesite algo de la biblioteca, irá a buscarlo a ésta. Por tanto si el programa es borrado dará un error de que no la encuentra.

Ventajas e inconvenientes de las bibliotecas estáticas:

Ventajas

- El programa después de compilado se puede llevar a otro ordenador sin necesidad de llevarse las bibliotecas.
- El programa después de compilado es, en principio, más rápido en ejecución. Cuando llama a una función de la biblioteca, la tiene en su código y no tiene que ir a leer el fichero de la biblioteca para encontrar la función y ejecutarla.
- Si ocurre algún cambio, a los ejecutables no les afecta.
- Durante su ejecución, las bibliotecas que hubiesen intervenido en su construcción no necesitan estar presentes.

Inconvenientes

- Si se realizan modificaciones en la biblioteca, es necesario recompilar todos los ejecutables que la utilizan.
- El programa después de compilado es más grande, ya que se hace copia de todo lo que necesita.

Ventajas e inconvenientes de las bibliotecas dinámicas:

Ventajas

- Cuenta con un menor tamaño de los ejecutables.
- Los procesos pueden compartir el código de biblioteca.
- Actualización automática de bibliotecas: Uso de versiones.

- Si se realizan modificaciones en la biblioteca para corregir un error, lo hace automáticamente en todos los ejecutables.
- Puede ser cargada bajo demanda en el momento en que se necesita su funcionalidad, e incluso puede ser descargada cuando no resulta necesaria.

Inconvenientes

- Mayor tiempo de ejecución debido a carga y montaje.
- Si ocurre algún cambio en la biblioteca con respecto a algún parámetro o una función, los ejecutables dejarán de funcionar.
- Las bibliotecas deben ser cargadas en su totalidad aunque no solo se utilice una parte de su funcionalidad.
- Dificultad de depuración y mantenimiento de aplicaciones que utilizan bibliotecas dinámicas
- Durante su ejecución, las bibliotecas que hubiesen intervenido en su construcción deben estar en el mismo directorio o en el camino de búsqueda "Path".

1.4 Integridad de ficheros

La integridad de un fichero consiste en averiguar si algún dato del archivo ha variado desde su creación. El archivo puede haber sido modificado por error, por cortes en la comunicación o, en el peor de los casos, porque un atacante haya inyectado código malicioso.

En entornos de código abierto es posible contrastar y compilar el código fuente del programa, revisándolo para comprobar que no contiene código no deseado. De esta forma se puede garantizar que el software no ocasionará ningún daño.

Cuando un desarrollador hace público un programa, suele almacenarlo en un servidor para que sea descargado. Este servidor no siempre está bajo el control del autor del programa o es administrado por la misma persona que ha realizado la herramienta, por tanto, puede ser vulnerable a ataques. Incluso si el programador aloja el software en un servidor administrado por él, es posible que pueda ser comprometido sin su conocimiento y otro atacante tenga la posibilidad de modificar o sustituir el programa que pone a disposición de todos los usuarios.

Cuando un atacante se apodera de un servidor FTP o web y tiene la posibilidad de modificar un programa que más tarde será descargado por otros, puede suponer un riesgo a nivel global si ocurre sobre aplicaciones populares. Este tipo de ataque se ha dado en muchas ocasiones. Sin

las herramientas y precauciones adecuadas, nada garantiza que un programa esté libre de malware, ni siquiera el hecho de descargarlo de una página oficial del fabricante.

En general, los axiomas que hay tener en cuenta siempre que se implemente un control de seguridad o se considere un aspecto de la seguridad, son los siguientes:

- **Confidencialidad.** Nadie debe poder acceder a los datos privados de un usuario, excepto el propio usuario y las personas autorizadas.
- **Disponibilidad.** El sistema y los datos tienen que ser accesibles por los usuarios autorizados en todo momento.
- **Integridad.** Nadie puede cambiar, recortar o falsificar ilegítimamente los datos.

Comprobar la integridad de los ficheros es la tarea que permite saber a ciencia cierta si éstos han sido modificados desde su creación.

1.4.1 Funciones hash criptográficas

Las funciones hash son estructuras de datos muy conocidas que se encuentran estrechamente ligadas con la criptografía en general y la integridad de los datos en particular.

Las funciones hash criptográficas convierten un mensaje de cualquier tamaño en un mensaje de una longitud constante. Lo que se obtiene al aplicar una función hash criptográfica a un mensaje (flujo de datos, o más usualmente, un archivo) se llama resumen criptográfico, huella digital o message digest. Es decir, a partir de un número indeterminado de bits, siempre se obtiene un número constante y diferente que identifica de forma unívoca a ese flujo de datos.

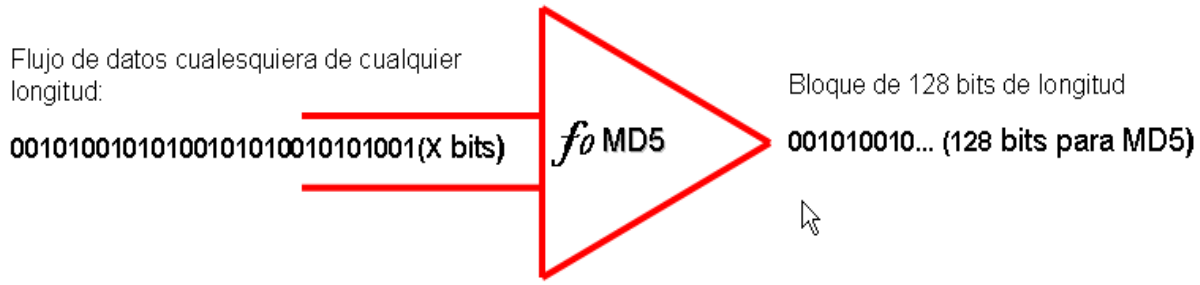

Ilustración 1: Representación de la función hash MD5

En seguridad de la información, se utilizan funciones hash criptográficas en procesos de autenticación, o de comprobación de integridad de datos.

1.4.1.1 Aplicaciones de las funciones hash

Con estas premisas, las aplicaciones de las funciones hash son claras:

1. **Protección de contraseñas.** Estas funciones permiten almacenar un resumen criptográfico, en vez del texto claro de las contraseñas. Así, en lugar de comparar las contraseñas en texto claro, se calcula su hash con una función y se comparan los resultados. De esta manera, el sistema no tiene por qué almacenar el texto claro en ningún momento para comprobar si alguien conoce una contraseña almacenada.

2. **Comprobación de integridad.** Si se calcula un hash de dos flujos de datos y dan un mismo resultado, se puede confirmar que los flujos de datos son idénticos.

3. **Garantizar la integridad de un flujo de datos a diferentes niveles:**

   - Al descargar u obtener un fichero por cualquier medio, se puede comprobar que se trata del original o que no tiene defectos si el proveedor proporciona un resumen criptográfico para comparar. Si hubiese cambiado un solo bit del archivo, el resumen sería muy distinto.

   - Se puede comprobar si se han producido cambios no controlados en los datos de un sistema de almacenamiento, calculando y comprobando de forma periódica los resúmenes criptográficos de los datos.

   - Ahorro de "coste" computacional en criptografía. Por ejemplo en la firma electrónica, firmar un mensaje es computacionalmente costoso. Sin embargo, que un hash represente a ese mensaje resulta mucho más liviano. Por tanto, se suele firmar un hash (que representa unívocamente a un flujo de datos o mensaje) en lugar de al

mensaje en sí.

1.4.1.2 Algoritmos más utilizados

Los algoritmos más utilizados para calcular el hash son:

- **MD5.** Ante la entrada de cualquier flujo de datos, devuelve un bloque de 128 bits. En 2006 se publicó un método capaz de encontrar colisiones en unos minutos y por tanto, aunque muy usado, no se considera totalmente seguro hoy día.
- **SHA256** (y sus sucesores: SHA512, por ejemplo). Ante la entrada de cualquier flujo de datos, devuelve un bloque de 256 bits. Al aumentar los bits de salida (hasta 2256 frente a 2128 del MD5), la posibilidad de colisión es menor y por tanto es más seguro. Se utiliza en los principales protocolos de cifrado: SSL, SSH, PGP o IPSec. Los métodos para encontrar colisiones, aunque existen en SHA, no tienen suficiente potencia como para poder proporcionar un ataque práctico, por tanto se considera relativamente seguro hoy día.

1.4.2 Métodos prácticos de comprobación de integridad

A continuación se muestra una tabla con los métodos de comprobación de la integridad de los archivos

Método	**Ventajas**
Hash	Garantía de que el archivo no ha cambiado
Firma Digital	Garantía de que el archivo no ha cambiado y proviene de una firma digital concreta, aunque nada garantiza que pertenezca a esa persona física determinada
Certificado	Garantía de que el archivo no ha cambiado y proviene de una firma digital concreta, garantizada su identidad por una tercera persona confiable(Autoridad Certificadora)
Virustotal	Ofrece una idea(no definitiva) sobre si el archivo contiene o no malware

Tabla 1: Comparativa de métodos de comprobación de identidad

Hash

Una vez que se conocen las características de las funciones hash, se pueden utilizar como método para garantizar la integridad de los archivos que se descargan. Por ejemplo, un programador que hace público un archivo ejecutable de un programa, puede calcular su hash mediante los algoritmos MD5 o SHA256 y publicarlo al mismo tiempo. De esta forma, se sabe que cualquier otro archivo que no sea exactamente ese que ha publicado el autor, tendrá un resultado hash diferente.

Firma digital

Además de la integridad, gracias a la criptografía simétrica es posible comprobar no sólo que un archivo no ha sido alterado, sino que ha sido realizado por la persona u organización que afirma haberlo creado. Esto se consigue a través de las firmas criptográficas que acompañan a ciertos archivos y, como se ha mencionado, corresponden a la firma del hash del archivo.

Las firmas suelen ser archivos con extensión SIG o ASC que resultan de firmar criptográficamente con la clave privada del autor el hash de un fichero. Si posteriormente se comprueba, a través de la clave pública, que el fichero firmado concuerda con la firma, es que se está ante un fichero realmente creado por quien dice haberlo hecho, y no modificado desde que se firmó. Una vez más, esto no garantiza en ningún modo las intenciones del archivo, sólo su origen.

Certificados

Un certificado consiste en la asociación entre una entidad física y una firma, realizado por una entidad confiable.

Un certificado digital certifica que una firma criptográfica pertenece a una persona, y una entidad lo ha comprobado, es decir, le ha pedido a esa persona sus datos y pruebas fehacientes de que la firma le pertenece.

Virustotal

En cualquier caso, además, existe la posibilidad de analizar concienzudamente el archivo en busca de virus o troyanos.

Es aconsejable utilizar sistemas de análisis múltiple como virustotal.com para asegurar al menos que algunas casas antivirus reconocen o no el programa como peligroso. Virustotal.com ofrece la posibilidad de conocer la opinión de múltiples motores antivirus sobre un programa o archivo concreto.

1.5 Conclusiones

En este capítulo se hizo un estudio de las principales características y conceptos fundamentales utilizados para el desarrollo del mecanismo de verificación de integridad de las bibliotecas dinámicas de los módulos del sistema SCADA GALBA.

Capítulo 2: Herramientas y tecnologías actuales

2.1 Introducción

La calidad de un producto de software es determinada en gran medida por la calidad del proceso utilizado para desarrollarlo y mantenerlo, por lo que se dedica esta sección a la selección de la metodología de desarrollo, lenguaje de programación, herramienta y lenguaje de modelado que se emplean en el desarrollo y documentación de la aplicación.

2.2 Sistema operativo: GNU/Linux

Un sistema operativo puede ser contemplado como una colección organizada de extensiones software del hardware, consistente en rutinas de control que hacen funcionar al computador y proporcionan un entorno para la ejecución de programas.

Estos programas utilizan las facilidades proporcionadas por el sistema operativo para obtener acceso a recursos del sistema informático como el procesador, archivos y dispositivos de entrada/salida (E/S). De esta forma, el Sistema Operativo constituye la base sobre la cual pueden escribirse los programas de aplicación, los cuales invocarían sus servicios por medio de llamadas al sistema. Por otro lado, los usuarios pueden interactuar directamente con él a través de órdenes concretas. En cualquier caso, actúan como interfaz entre los usuarios/aplicaciones y el hardware de un sistema informático.

2.3 Lenguaje de Modelado: UML

Lenguaje Unificado de Modelado (*UML, por sus siglas en inglés, Unified, Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; aun cuando todavía no es un estándar oficial, está respaldado por el OMG (*Object Management Group*). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema, incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

Es importante resaltar que UML es un "lenguaje" para especificar y no para describir métodos o procesos, es decir, es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de software.

2.4 Metodología de Desarrollo

En un proyecto de desarrollo de software la metodología define quién debe hacer qué, cuándo y cómo debe hacerlo. Una metodología es un proceso. No existe una metodología de software universal. Las características de cada proyecto (equipo de desarrollo, recursos) exigen que el proceso sea configurable.

Una de las metodologías de desarrollo de software utilizada en la actualidad es la *Extreme Programing* (XP), centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios.

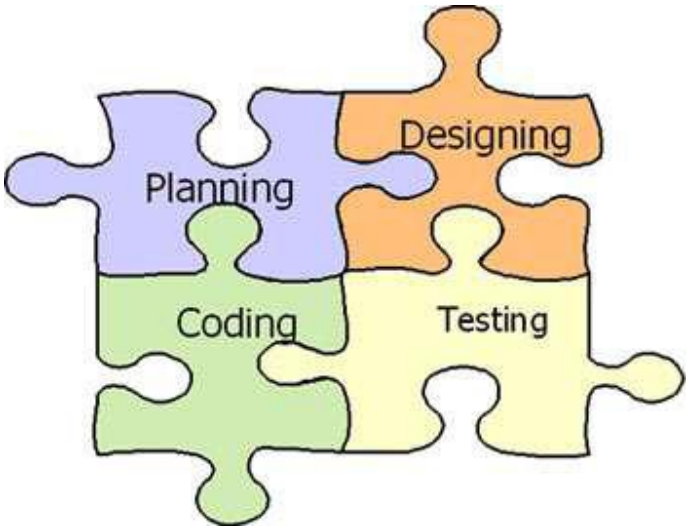

Ilustración 2: Metodología Extreme Programing

Características de la metodología XP:

- *Pruebas Unitarias:* Se basa en las pruebas realizadas a los principales procesos, de tal manera que se puedan hacer pruebas de las fallas que pudieran ocurrir.
- *Refabricación:* Se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- *Programación en pares:* Una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

¿Qué es lo que propone XP?

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantiva del proceso.
- El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.
- El cliente o el usuario se convierten en miembro del equipo.

Lo fundamental en este tipo de metodología es:

- La comunicación, entre los usuarios y los desarrolladores.
- La simplicidad, al desarrollar y codificar los módulos del sistema.
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.

2.5 Lenguaje de programación

Un lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Es utilizado para controlar el comportamiento físico y lógico de una máquina.

2.5.1 Lenguaje C++

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje multiparadigma.

Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos.

Una particularidad del C++ es la posibilidad de redefinir los operadores (sobrecarga de operadores), y de poder crear nuevos tipos que se comporten como tipos fundamentales. C++ permite trabajar tanto a alto como a bajo nivel. A partir de dichas razones y por estudios realizados en el proyecto sobre los diferentes lenguajes de programación posibles a utilizar en el proyecto se seleccionó dicho lenguaje.

2.5.2 Bash

El intérprete Bash es algo más que una simple consola. Es un lenguaje interpretado de programación que ayuda al administrador a realizar la mayor parte de las tareas necesarias, tanto en la automatización como en el arranque del sistema.

El lenguaje usado por Bash está definido por su propio intérprete y combina la sintaxis de otros Shells, como el Korn Shell (*ksh*) o el C Shell (*csh*). Muchos de los comandos que usualmente se usan en la consola también pueden usarse en los scripts, salvo aquellos que pertenecen estrictamente a una distribución en particular.

La sintaxis de órdenes de Bash es un superconjunto de instrucciones basadas en la sintaxis del Intérprete Bourne. La mayoría de los shell scripts Bourne pueden ejecutarse por Bash sin ningún cambio, con la excepción de aquellos guiones del intérprete de órdenes, o consola, Bourne que hacen referencia a variables especiales de Bourne o que utilizan una orden interna de Bourne. Cuando se utiliza como un intérprete de órdenes interactivo, Bash proporciona autocompletado de nombres de programas, nombres de archivos y nombres de variables cuando el usuario pulsa la tecla TAB.

2.6 OpenSSL

OpenSSL es un proyecto de software libre basado en SSLeay, desarrollado por Eric Young y Tim Hudson. Consiste en un robusto paquete de herramientas de administración y bibliotecas relacionadas con la criptografía, que suministran funciones criptográficas a otros paquetes como OpenSSH y navegadores web.

Estas herramientas ayudan al sistema a implementar el Secure Sockets Layer (SSL), así como otros protocolos relacionados con la seguridad, como el Transport Layer Security (TLS). Este paquete de software es importante para cualquiera que esté planeando usar cierto nivel de seguridad en su máquina con un sistema operativo libre basado en GNU/Linux.

El conjunto de herramientas OpenSSL es una característica de FreeBSD que muchos usuarios pasan por alto. OpenSSL ofrece una capa de cifrado de transporte sobre la capa normal de comunicación, permitiendo la combinación con muchas aplicaciones y servicios de red.

Algunos usos de OpenSSL son la validación cifrada de clientes de correo y las transacciones basadas en web como pagos con tarjetas de crédito.

Uno de los usos más comunes de OpenSSL es ofrecer certificados para usar con aplicaciones de software. Estos certificados aseguran que las credenciales de la compañía o individuo son válidos y no son fraudulentos. Si el certificado en cuestión no ha sido verificado por una de las diversas “autoridades certificadoras”, suele generarse una advertencia al respecto. Este

proceso tiene un costo asociado y no es un requisito imprescindible para usar certificados, aunque puede darle un poco de tranquilidad a los usuarios más paranoicos.

2.7 IDE: Eclipse

Un entorno de desarrollo integrado (*Integrated Development Environment o IDE*) es un programa compuesto por una serie de herramientas que utilizan los programadores para desarrollar código. Esta herramienta puede estar pensada para su utilización con un único lenguaje de programación o bien puede dar cabida a varios de estos.

Las herramientas que normalmente componen un entorno de desarrollo integrado son las siguientes: un editor de texto, un compilador, un intérprete, unas herramientas para la automatización, un depurador, un sistema de ayuda para la construcción de interfaces gráficas de usuario y, opcionalmente, un sistema de control de versiones.

Hoy en día los entornos de desarrollo proporcionan un marco de trabajo para la mayoría de los lenguajes de programación existentes en el mercado (*C, C++, C#, Java, Python y Visual Basic*). Además es posible que un mismo entorno de desarrollo tenga la posibilidad de utilizar varios lenguajes de programación, como es el caso de Eclipse.

2.8 Visual Paradigm

Visual Paradigm para UML (*Unified Modeling Language, Lenguaje Unificado de Modelado*), es una herramienta CASE (*Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora*) profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Los sistemas de modelado UML ayudan a una más rápida construcción de aplicaciones de calidad y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, ingeniería inversa, generar código a partir de diagramas, generación de objetos a partir de bases de datos, generación de bases de datos a partir de diagramas de entidad relación y generar documentación, posee licencia gratuita y comercial Permite interoperabilidad con modelos UML2 (*metamodelos UML 2.x para plataforma Eclipse*). Se integra con Visio. Posee una plataforma, llamada SDE, capaz de integrarse con Eclipse, NetBeans, Oracle JDeveloper, JBuilder, IntelliJ IDEA, WebLogicWorkshop, Microsoft Visual Studio.

2.9 Herramienta de auto-construcción.

Las herramientas de auto-construcción (AT) tienen dos objetivos principales, brinda mayor factibilidad para el uso del usuario y para que el proyecto desarrollado sea más portátil, incluso para sistemas en los que nunca se ha probado, instalado o construido su código.

Las AT proporcionan un entorno de construcción que es lo suficientemente flexible como para permitir que el proyecto sea construido con éxito en las futuras versiones o distribuciones, prácticamente sin cambios en los scripts de creación.

El marco GNU autotools se compone de tres paquetes principales, cada uno de los cuales proporcionan y se basan en varios componentes más pequeños. Los tres paquetes principales son autoconf, automake y libtool. Estos paquetes fueron inventados en ese orden, y han evolucionado con el tiempo.

2.9.1 Autoconf

Es un paquete de macros M4 extensible que producen los scripts de shell para configurar automáticamente los paquetes de código fuente. Estos scripts pueden adaptar los paquetes a muchas clases de sistemas tipo UNIX sin la intervención del usuario. Autoconf crea una secuencia de comandos de configuración de un paquete a partir de un archivo de plantilla que muestra las funciones del sistema operativo que el paquete puede utilizar, en forma de las llamadas macro M4.

El autor original fue David MacKenzie, quien inició el proyecto en 1991. Mientras que los scripts de configuración se hacen más largos y más complejos, existían sólo unas pocas variables que deben ser especificadas por el usuario. La mayoría de ellas eran simplemente decisiones que deben tomarse con respecto a los componentes, características y opciones. Con Autoconf, en lugar de modificar, depurar y perder el sueño, los desarrolladores pueden escribir un guión corto meta-archivo, utilizando una macro concisa basada en el lenguaje, y dejarlo generar un script de configuración perfecta.

Una secuencia de comandos de configuración generada es más portátil, más correcta y más fácil de mantener que una versión a mano del código de la misma escritura. Además, detecta a menudo errores de semántica o lógica que el autor habría pasado depurando días.

Autoconf genera scripts de configuración, proporciona un conjunto común de opciones que son importantes para todos los portátiles, gratuito y de código abierto. Genera y configura secuencias de comandos, además también proporciona opciones específicas de cada proyecto. Estos se definen en el archivo de configure.ac para cada proyecto.

Este paquete ofrece varios programas como:

- autoconf
- autoheader
- autom4te
- autoreconf
- autoscan
- autoupdate
- ifnames

2.9.2 Automake

La diferencia más significativa entre un proyecto de software libre exitoso y uno que rara vez recibe una segunda mirada se encuentra en el corazón de los detalles de mantenimiento del proyecto. Los usuarios potenciales se vuelven disgustados con un proyecto bastante fácil, especialmente cuando ciertos trozos de funcionalidad esperada faltan o está escrito incorrectamente. Los usuarios han llegado a esperar algunos estándar objetivos Make. Una marca objetivo es una meta especificada en la línea de comando make:

```
$ Make install
```

En este ejemplo, install es la meta u objetivo. Los objetivos comunes a realizar incluyen all, clean e install. Ninguno de estos son objetivos *reales, u*n objetivo real es un archivo generado por el sistema de construcción. Si se está construyendo un ejecutable llamado doofabble, entonces se esperaba ser capaz de escribir:

```
$ Make doofabble
```

Esto generará un archivo ejecutable llamado real doofabble. Pero especificando objetivos reales en la línea de comando make es un trabajo más de lo necesario. Cada proyecto deberá ser construido de manera diferente.

Automake consiste en convertir una especificación muy simplificada del proceso de construcción de su proyecto en la sintaxis estándar makefile repetitivo que siempre funciona correctamente la primera vez, y proporciona toda la funcionalidad estándar esperada de un proyecto de software libre.

El paquete Automake proporciona las siguientes herramientas en forma de scripts de Perl:

- automake
- aclocal

2.9.3 Libtool

Libtool no solo provee un conjunto de macros de autoconf que ocultan las diferencias de nombres de las bibliotecas en los makefiles, también provee una biblioteca de carga dinámica de funciones que pueden ser añadidas a los programas, permitiendo que se pueda escribir más código dinámico compartido de gestión de objetos en tiempo de ejecución.

El paquete libtool ofrece los siguientes programas, bibliotecas y ficheros de cabecera:

- libtool (programa)
- libtoolize (programa)
- ltdl (bibliotecas estáticas y compartidas)
- ltdl.h (cabecera)

El script de shell libtool es una versión genérica de Libtool diseñado para ser utilizado por los programas de su plataforma.

2.10 Conclusiones

En este capítulo se definieron las herramientas y tecnologías a utilizar, como metodología de desarrollo de software se seleccionó *Extreme Programming* y para realizar el modelado del sistema la herramienta CASE *Visual Paradigm*, apoyándose en el Lenguaje Unificado de Modelado (UML). Se seleccionó *Eclipse* como IDE, *C++* y Bash como lenguajes de programación y *OpenSSL* como biblioteca de encriptación.

Capítulo 3: Descripción de la solución propuesta

3.1 Introducción

La planificación y el diseño del software son etapas importantes en su ciclo de desarrollo. Es clave durante su desarrollo que se logre una buena planificación antes de entrar en el diseño del mismo. Las aplicaciones deben estar soportadas por una correcta selección de su arquitectura para organizar y comprender mejor el sistema. El presente capítulo aborda los temas referentes a cada una de estas etapas.

3.2 Flujo actual de eventos

Una vez concluido el proceso de desarrollo de los módulos del SCADA GALBA, se procede a la compilación y generación de sus correspondientes bibliotecas y ejecutables. Este es un proceso que se lleva a cabo en el departamento de Integración y Despliegue, donde se realiza la integración de los módulos haciendo uso de diversas herramientas de auto-construcción y empaquetado. Todo este proceso requiere que se especifique para cada módulo sus respectivas dependencias, ya sean a bibliotecas propias del GALBA o a alguna biblioteca externa. Para ello se editan los archivos *configure.ac*, que permiten enlazar las bibliotecas instaladas en la computadora con las requeridas por los módulos antes de ser compilado el código fuente. Sin embargo estas comprobaciones solo se realizan teniendo en cuenta el nombre de cada biblioteca, o la ruta especificada para la inclusión de las mismas, sin realizar ningún chequeo de integridad. Por tanto una vez empaquetados los módulos, en el momento de ser ejecutados estos no son capaces de determinar si las bibliotecas con las cuales se está ejecutando son realmente las mismas con las que fue compilado.

3.3 Propuesta de sistema

Para poder proteger la integridad de las bibliotecas desarrolladas y la confianza hacia nuestros desarrolladores, se decide desarrollar un mecanismo de seguridad en tiempo de configuración y compilación de los módulos.

Mediante el mecanismo propuesto se utiliza la biblioteca *libssl y* el algoritmo MD5 que proporciona la herramienta *md5sum, generando un* código hash único para cada fichero. Incluso para un mismo fichero, con solo adicionar un simple espacio, este código cambia completamente, facilitándole al sistema comprobar la integridad de sus bibliotecas, comparando de cada una de ellas, los códigos hash generados en tiempo de ejecución y compilación.

Para dar cumplimiento al mecanismo propuesto se establecen los siguientes requisitos funcionales.

1. Crear una plantilla llamada *check_dependencies.cpp.in* que contenga las funcionalidades de comprobar las dependencias y generar los códigos MD5. Debe garantizarse la ejecución recursiva del chequeo de dependencias, es decir, que las dependencias verifiquen igualmente sus dependencias. Esta funcionalidad solo será aplicada para las aplicaciones y las bibliotecas que dependan de bibliotecas desarrolladas por el equipo, en caso de que la biblioteca no tenga dichas dependencias, la plantilla solo contendrá la función en común definida.
2. Modificar el fichero de configuración de la auto-construcción *configure.ac* perteneciente al módulo o biblioteca que dependa de una biblioteca desarrollada por el equipo de desarrollo.
3. Modificar el fichero *src/Makefile.am* perteneciente al módulo o biblioteca.
4. En la función principal (*main*) de cada aplicación se debe comprobar la correcta ejecución de la función que realiza la comprobación de las dependencias. En caso de ser satisfactorio el resultado, se continúa la ejecución del módulo, en caso contrario, se detiene la ejecución y se informa el problema.

3.3.1 Personal relacionado con el sistema

El personal relacionado con el sistema es el que lleva a cabo todas las operaciones necesarias para la obtención de los resultados de los procesos que se ejecutan en el mismo.

Personal relacionado con el sistema	Descripción
Integrador	Realiza las operaciones necesarias para poder llevar a cabo la auto-construcción del sistema.

Tabla 2 Personal relacionado con el sistema.

3.4 Historias de Usuario

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales.

La historia de usuario debe ser fácil de entender tanto por desarrolladores como clientes; posible de probar, de valor para el cliente y lo suficientemente pequeña para que los

programadores puedan construir un considerable grupo de ellas en cada iteración. En XP la gestión de requerimientos del sistema es extremadamente simple, el cliente describe y prioriza sus necesidades mediante historias de usuario que deben ser descripciones cortas y escritas sin terminología técnica.

Las historias de usuario solamente proporcionarán los detalles sobre la complejidad y cuánto tiempo conllevará la implementación de dicha historia de usuario. El nivel de detalle de las historias de usuario debe ser el máximo posible, permitiendo hacerse una ligera idea de cuánto costará implementar el sistema.

Historias de usuario	
Número: 1	**Nombre Historia de Usuario:** Generar códigos MD5.
Actor: Integrador	**Iteración Asignada:** 1
Prioridad de Negocio: Alta	**Puntos Estimados:** 2
Riesgo en desarrollo: Alto	**Puntos Reales:** 2
Descripción: El sistema podrá generar los códigos MD5 correspondientes a cada una de las bibliotecas desarrolladas.	
Observaciones: Se debe generar en tiempo de compilación y en tiempo de ejecución.	

Tabla 3 HU Generar códigos MD5.

Historias de usuario	
Número: 2	**Nombre Historia de Usuario:** Chequear dependencias.
Actor: Integrador	**Iteración Asignada:** 1
Prioridad de Negocio: Alta	**Puntos Estimados:** 3
Riesgo en desarrollo: Alto	**Puntos Reales:** 3
Descripción: El sistema podrá chequear las dependencias y garantizar la ejecución recursiva del chequeo de las mismas.	
Observaciones:	

Tabla 4 HU Chequear dependencias.

Historias de usuario	
Número: 3	**Nombre Historia de Usuario:** Modificar el fichero de configuración de la auto-construcción.
Actor: Integrador	**Iteración Asignada:** 2

Prioridad de Negocio: Media	**Puntos Estimados:** 1
Riesgo en desarrollo: Medio	**Puntos Reales:** 1
Descripción: El integrador podrá modificar el fichero de configuración de la auto-construcción *configure.ac* perteneciente al módulo o biblioteca que dependa de una biblioteca desarrollada por el equipo de desarrollo.	
Observaciones:	

Tabla 5 HU Modificar el fichero de configuración de la auto-construcción.

Historias de usuario	
Número: 4	**Nombre Historia de Usuario:** Modificar el fichero *src/Makefile.am* perteneciente al módulo o biblioteca.
Actor: Integrador	**Iteración Asignada:** 2
Prioridad de Negocio: Media	**Puntos Estimados:** 2
Riesgo en desarrollo: Medio	**Puntos Reales:** 2
Descripción: El integrador podrá modificar el fichero *src/Makefile.am* perteneciente al módulo o biblioteca.	
Observaciones:	

Tabla 6 HU Modificar el fichero src/Makefile.am perteneciente al módulo o biblioteca.

Historias de usuario	
Número: 5	**Nombre Historia de Usuario:** Requisitos de software
Descripción: Se debe tener instalado una computadora con el sistema operativo GNU/Linux Debian Squeeze con todas las dependencias del scada GALBA.	
Observaciones:	

Tabla 7 HU Requisitos de software

Historias de usuario	
Número: 6	**Nombre Historia de Usuario:** Requisitos de Hardware
Descripción: Para la ejecución del mecanismo se debe tener una computadora con las siguientes prestaciones: Procesador Intel Pentium IV Frecuencia del CPU: 2.4GHz	

Memoria RAM: 1GB
Observaciones:

Tabla 8 HU Requisitos de Hardware

Historias de usuario	
Número: 7	**Nombre Historia de Usuario:** Restricciones en el diseño e implementación.
Descripción: Para el desarrollo de la solución se definen una serie de restricciones: Lenguaje de programación: C++ y Bash Entorno integrado de desarrollo (IDE): Eclipse Juno	
Observaciones:	

Tabla 9 HU Restricciones en el diseño e implementación.

Historias de usuario	
Número: 8	**Nombre Historia de Usuario:** Requisitos de Usabilidad.
Descripción: El sistema podrá ser usado por los integradores y desarrolladores del SCADA GALBA para poder controlar la integridad de sus bibliotecas dinámicas.	
Observaciones:	

Tabla 10 HU Requisitos de usabilidad

Historias de usuario	
Número: 9	**Nombre Historia de Usuario:** Requerimiento de Documentación de Usuario.
Descripción: Se debe garantizar que el sistema cuente con una correcta documentación.	
Observaciones:	

Tabla 11 HU Requerimiento de documentación de usuario

Historias de usuario	
Número: 10	**Nombre Historia de Usuario:**

	Requerimiento asociado al Licenciamiento.
Descripción: Se debe garantizar que el sistema se desarrolle bajo los principios del software libre y por tanto cualquier componente de software que se utilice también lo debe ser.	
Observaciones:	

Tabla 12 HU Requerimiento asociado al Licenciamiento.

3.5 Planificación y entrega

La planificación es una fase corta en la que el cliente, los gerentes y el grupo de desarrolladores acuerdan el orden en que deberán implementarse las historias de usuario y asociadas a éstas, las entregas.

Esta fase consiste en una o varias reuniones grupales de planificación, donde resultado de esta fase es un Plan de Entregas.

Los programadores realizan una estimación del esfuerzo necesario de cada una de las HU en dependencia de la prioridad establecida por el cliente. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente.

3.5.1 Plan de entrega

Una vez que el cliente culmina la elaboración de las HU, se comienza con la creación del Plan de Entregas.

El mismo se hace con la intención de que los programadores obtengan una estimación de dichas historias en cuanto al nivel de detalle, o sea, para fijar el período de tiempo que se puede tardar en la implementación de cada una.

Este plan recoge las historias que serán agrupadas para conformar una entrega, y el orden de las mismas. El plan se realiza en base de las estimaciones de esfuerzos de desarrollo realizadas por los programadores. Las estimaciones de esfuerzos asociadas a la implementación de las historias tendrán como medida el punto. Un punto, se corresponde a una semana de programación ideal. Las historias deben poder ser programadas en un tiempo entre una y tres semanas.

La planificación se puede realizar basándose en el tiempo o en el alcance. La velocidad del proyecto es aprovechada para establecer cuántas historias de usuario se pueden hacer antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias.

Historia de usuario	Final 1ra Iteración 22/03/2012	Final 2da Iteración 12/04/2012
Generar códigos MD5. Chequear dependencias.	5	Finalizado
Modificar el fichero de configuración de la auto-construcción. Modificar el fichero *src/Makefile.am* perteneciente al módulo o biblioteca.	No empezado	3

Tabla 13 Plan de entrega

Historias de usuario	Tiempo estimado	Iteración	Tiempo real
Generar códigos MD5. Chequear dependencias.	5	1	5
Modificar el fichero de configuración de la auto-construcción. Modificar el fichero *src/Makefile.am* perteneciente al módulo o biblioteca.	3	2	3

Tabla 14 Estimación del esfuerzo por historia de usuarios.

3.5.2 Plan de iteraciones

Luego de estimar el esfuerzo con el cual sería desarrollada las historias de usuario y contar con un plan de entrega, se da paso a la planificación de la etapa de implementación del sistema. Los objetivos principales del plan de iteración son dar al equipo un lugar central para la información con respecto a los objetivos de la iteración, un plan detallado con tareas asignadas y resultados de la evaluación. Este artefacto también ayuda al equipo a monitorear el progreso de la iteración.

En este plan se recogen las historias de usuario que sería implementado en cada iteración, de acuerdo al orden preestablecido. De acuerdo con lo antes mencionado se decide realizar el sistema en dos iteraciones, las cuales se especifican a continuación:

Iteración 1

Esta primera iteración tiene como meta implementar las historias de usuarios con prioridad alta para el cliente, las cuales son: generar códigos MD5 y chequear dependencias. Estas historias de usuario constituyen la estructura básica del sistema. Con la finalización de esta iteración se obtendrá la primera entrega del sistema con el propósito de mostrar al cliente lo realizado y recibir retroalimentación del mismo.

Iteración 2

En esta iteración se implementan las historias de usuario que tiene prioridad media para el cliente, las cuales son: modificar el fichero de configuración de la auto-construcción y modificar el fichero *src/Makefile.am* perteneciente al módulo o biblioteca. La versión de prueba referente a esta iteración junto con las implementaciones anteriores, serán mostradas al cliente con el objetivo de realizar cambios en base a la opinión del mismo. Al finalizar esta iteración, luego de que haya pasado satisfactoriamente por la etapa de pruebas se dispondrá de la aplicación con todas las funcionalidades descritas por el cliente.

3.5.3 Plan de duración de las iteraciones

Para una mayor organización del trabajo y como parte del ciclo de vida de un proyecto utilizando la metodología XP, se crea el plan de duración de cada una de las iteraciones, en este caso se hace para el único equipo de desarrollo con el cual se cuenta. Este plan tiene como finalidad reflejar la duración de cada iteración, así como el orden en que serán implementadas las HU en cada una de las mismas, lo que ayuda a obtener una idea aproximada del tiempo que durará la confección del sistema en su totalidad.

Iteraciones	Orden de las HU a implementar	Duración total de las iteraciones
Iteración 1	Generar códigos MD5. Chequear dependencias.	5 semanas
Iteración 2	Modificar el fichero de configuración de la auto-construcción. Modificar el fichero *src/Makefile.am* perteneciente al módulo o biblioteca.	3 semanas

Tabla 15 Plan de duración de las iteraciones

3.5.4 Historias de usuario divididas en tareas

Las historias de usuario están embebidas dentro de tareas de programación, que serán definidas en cada iteración. Estas tareas son escritas en fichas como las historias de usuario, pero en el lenguaje de los programadores. Cada tarea debería ser estimada en un período de uno a tres días de desarrollo.

3.5.4.1 Iteración 1

Historia de usuario	Tareas
Generar códigos MD5.	1. Crear las variables necesarias. 2. Obtener el descriptor del fichero en cuestión. 3. Obtener el tamaño del fichero. 4. Cargar en memoria dicho fichero. 5. Generar código MD5 puro. 6. Convertir el resultado de dicha función a valores alfanuméricos y retornarlo.
Chequear dependencias.	1. Crear un mapa para almacenar el nombre de la biblioteca y el valor del MD5 2. Verificar existencia de la biblioteca. 3. Verificar la existencia de la función definida que chequea las dependencias en la biblioteca. 4. Comprobar la igualdad entre el MD5 generado en tiempo de compilación y el MD5 generado en tiempo de ejecución.

Tabla 16 Tareas abordadas en la primera iteración.

3.5.4.2 Iteración 2

Historia de usuario	Tareas
Modificar el fichero de configuración de la auto-construcción.	1. Almacenar la ruta de las bibliotecas. 2. Almacenar el/los nombre(s) de las bibliotecas de las que depende la aplicación o biblioteca en cuestión.

	3. Adicionar src/check_dependencies.cpp dentro de AC_CONFIG_FILES 3.1 En caso de ser una aplicación adicionar al final del fichero la modificación del archivo config.h en el que se le adicionará la declaración de la función que chequea las dependencias.
Modificar el fichero *src/Makefile.am* perteneciente al módulo o biblioteca.	1. Adicionar al _SOURCES el fichero check_dependencies.cpp. 2. Adicionar una regla para insertar dentro de check_dependencies.cpp el nombre de la biblioteca de la cual depende y su respectivo MD5 antes de generar el fichero check_dependencies.o.

Tabla 17 Tareas abordadas en la segunda iteración

3.6 Conclusiones

A partir de las acciones vinculadas al campo de acción, se definió una propuesta del sistema. Se especificaron los usuarios que estarán relacionados con su utilización y funcionamiento. Se realizó una descripción de las HU precisando por el cliente la prioridad de cada una, definiendo así el orden en el que serán implementadas las mismas. Se cuenta con 4 HU que serán implementadas en dos iteraciones y se plantearon y describieron las tareas de desarrollo para dar cumplimiento a las funcionalidades abordadas por las HU.

Capítulo 4: Implementación y pruebas

4.1 Introducción

En la etapa de implementación de un software, se transforman las clases y objetos en ficheros fuente, binarios y ejecutables. El resultado final, es un sistema que en la etapa de pruebas, se evalúa su calidad y desempeño como producto de software. En esta etapa se detectan y corrigen errores para la posterior aceptación del producto. El presente capítulo aborda los temas relacionados con la implementación y pruebas realizadas al sistema.

4.2 Principales funcionalidades

A continuación se muestran algunos fragmentos de código fundamentales para el funcionamiento del mecanismo de verificación de la integridad de las bibliotecas dinámicas de los diferentes módulos del sistema SCADA GALBA.

4.2.1 Generación de código MD5

En la ilustración 3 se muestra la función que permite la generación del código MD5 de la biblioteca que se pasa por parámetro. La función abre la biblioteca devolviendo un descriptor, luego se obtiene el tamaño y posteriormente se carga en memoria todo el contenido de la biblioteca. Para luego con la funcionalidad que brinda la biblioteca *openssl* generar el código MD5 de dicha biblioteca. Para mayor entendimiento de la función MD5 ver documentación de Openssl.

```
std::string _md5(const char* libso)
{
    unsigned char md5_digest[MD5_DIGEST_LENGTH];
    int file_descript;
    unsigned long file_size;
    char* file_buffer;
    file_descript = open(libso, O_RDONLY);
    if(file_descript < 0) return "";
    struct stat statbuf;
    if(fstat(file_descript, &statbuf) < 0) return "";
    file_size = statbuf.st_size;
    file_buffer = (char*)mmap(0, file_size, PROT_READ, MAP_SHARED, file_descript, 0);
    MD5((unsigned char*) file_buffer, file_size, md5_digest);
    char result[33];
    for(int i=0; i <MD5_DIGEST_LENGTH; i++) {
        sprintf(&result[i*2],"%02x", (unsigned int)md5_digest[i]);
    }
    std::cout<<"RESULT: "<<result<<std::endl;
    return result;
}
```

Ilustración 3 Generación de código MD5

4.2.2 Chequear dependencias

En la ilustración 4 se muestra la función que permite chequear la integridad de las dependencias de una aplicación o de una biblioteca. Se recorre un mapa que contiene las dependencias con el código MD5 que se generó en tiempo de compilación. En primer lugar se verifica la existencia de dicha biblioteca, que exista la funcionalidad de chequear las dependencias para el chequeo recursivo, luego se verifica si la ejecución de dicha función es satisfactoria y por último si coincide el MD5 generado en tiempo de ejecución con el que está almacenado en el mapa.

```
typedef bool(*ptr_check_depends)();
bool check_depends()
{
    std::map<std::string, std::string> deps_map;
    //MAPS
    std::map<std::string, std::string>::iterator it = deps_map.begin();
    for(;it!=deps_map.end();++it)
    {
        void *handler = dlopen(std::string("@libdir@/" + it->first).c_str(),RTLD_LAZY);
        if(handler == NULL)
        {
            return false;
        }
        void *cds = dlsym(handler,"check_depends");
        if(cds == NULL)
        {
            dlclose(handler);
            return false;
        }
        if(!(ptr_check_depends(cds))())
        {
            dlclose(handler);
            cds = NULL;
            return false;
        }
        dlclose(handler);

        if(_md5(std::string("@libdir@/" + it->first).c_str()) != it->second)
        {
            return false;
        }
    }
    return true;
}
```

Ilustración 4 Chequear dependencias.

4.2.3 Insertar en el mapa el nombre de la biblioteca con su correspondiente MD5

En la ilustración 5 se muestra un fragmento del código encargado de insertar en el fichero que contiene las funcionalidades anteriormente descritas, el nombre de sus dependencias con su respectivo código MD5.

```
sim_scada_configuracion_servidor-check_dependencies.o: check_dependencies.cpp
	for item in @confs_cv_cudependecies@;do $(SED) -e "s/MAPS/& \ndeps_map
[\"lib$$item.so\"]=\"`$(MD5SUM) "@confs_cv_depends_dir@/lib$$item.so" |
$(AWK) '{split($$0,cadena," ");print(cadena[1]);}'`\";/g" -i
check_dependencies.cpp; done
	$(CXX) $(DEFS) $(DEFAULT_INCLUDES) $(INCLUDES)
$(sim_scada_configuracion_servidor_CPPFLAGS) $(CPPFLAGS)
$(sim_scada_configuracion_servidor_CXXFLAGS) $(CXXFLAGS) -MT
sim_scada_configuracion_servidor-check_dependencies.o -MD -MP -MF $(DEPDIR)/
sim_scada_configuracion_servidor-check_dependencies.Tpo -c -o
sim_scada_configuracion_servidor-check_dependencies.o `test -f
'check_dependencies.cpp' || echo '$(srcdir)/'`check_dependencies.cpp
	$(am__mv) $(DEPDIR)/sim_scada_configuracion_servidor-
check_dependencies.Tpo $(DEPDIR)/sim_scada_configuracion_servidor-
check_dependencies.Po
```

Ilustración 5: Insertar en el mapa el nombre de la biblioteca con su correspondiente MD5.

4.3 Pasos para la auto-construcción

Para llevar a cabo la ejecución de la auto-construcción del proyecto será necesario seguir los siguientes pasos:

4.3.1 Generar los ficheros necesarios para la auto-construcción del proyecto.

sh autogen.sh --verbose

4.3.2 Comprobar las funcionalidades, bibliotecas y herramientas del sistema, necesarias para la auto-construcción del proyecto. Definir dónde será instalado dicho proyecto luego de ser compilado y la forma en que se construirá el proyecto.

./configure --prefix=$HOME/scadasim --enable-recursive=$PWD --C

4.3.3 Una vez configurado todo el proyecto, se pasa a compilarlo. Durante este proceso se van generando los códigos MD5 de las bibliotecas y se van almacenando.

make

4.3.4 Una vez compilado, se procede a instalar el proyecto.

make install

4.4 Representación gráfica del mecanismo para el control de la integridad de las bibliotecas dinámicas del sistema SCADA Guardián del Alba

En la Ilustración 6 se muestra un diagrama que establece como queda conformado gráficamente el mecanismo para el control de la integridad de las bibliotecas dinámicas del sistema SCADA Guardián del Alba. A continuación se describe la estructura del mismo:

La carpeta principal, llamada Scadasim, es la encargada de contener en su interior cada uno de los módulos establecidos por el cliente con las bibliotecas que utiliza cada una de ellas. Se les establece esta estructura para probar todos los posibles casos existentes de dependencias.

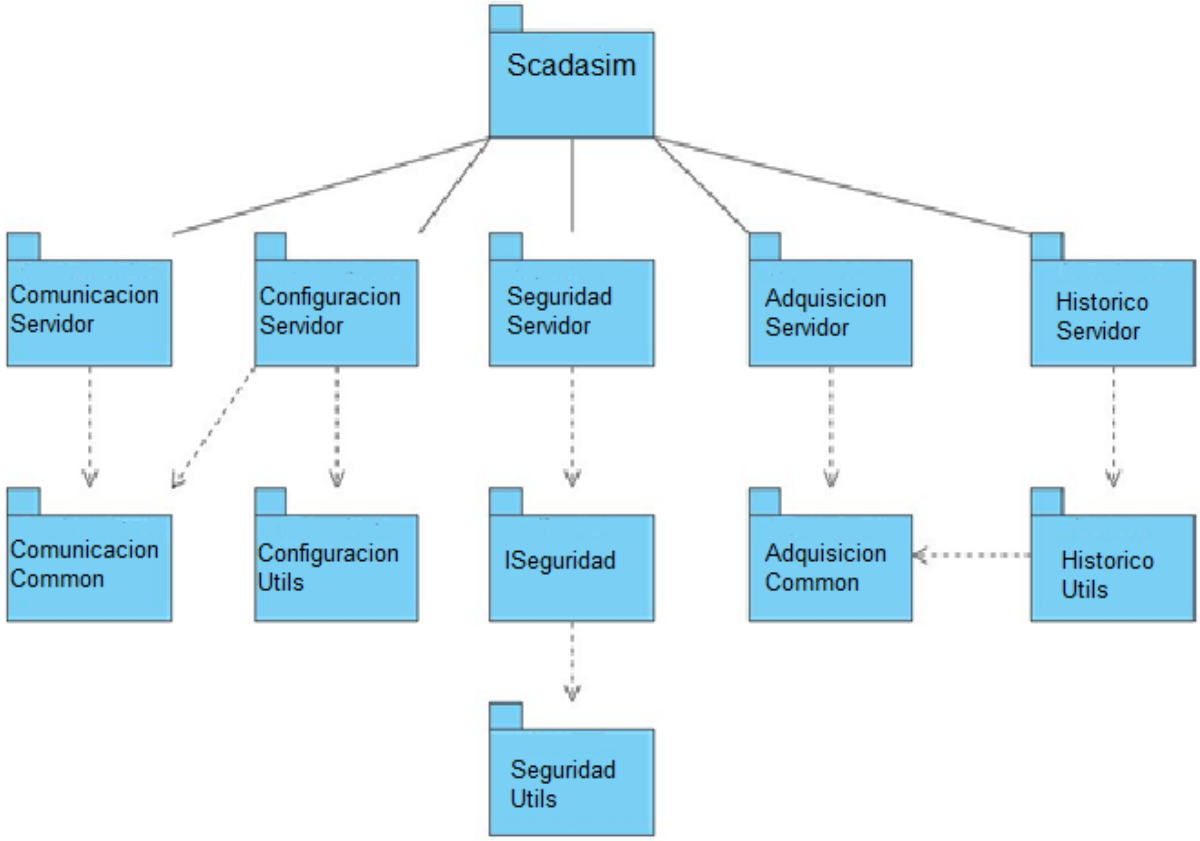

Ilustración 6: Representación gráfica del mecanismo para el control de la integridad de las bibliotecas dinámicas del sistema SCADA Guardián del Alba.

4.4.1 Estructura interna del mecanismo para el control de la integridad de las bibliotecas dinámicas del sistema SCADA Guardián del Alba

En las Ilustraciones 7 y 8 respectivamente se muestran ejemplos de cómo quedarían los ficheros de configuración para la auto-construcción de una aplicación y de una biblioteca. A continuación se describen los ficheros claves que se modifican o agregan al proyecto:

- *check_dependencies.cpp.in*: Plantilla que contiene las funcionalidades para el chequeo de las dependencias, a partir de la cual se genera el fichero fuente *check_dependencies.cpp* que posteriormente es incluido dentro del binario que se genera.
- *src/Makefile.am*: Dentro de este fichero se adiciona la regla encargada de modificar el contenido de la fuente *check_dependencies.cpp* en dónde se inserta dentro de un mapa el nombre de la biblioteca con su respectivo MD5.
- *ComunicacionServidor o ComunicacionCommon/configure.ac*: Dentro de este fichero se guardan, en una variable la ruta de donde serán almacenadas las bibliotecas cuando se estén compilando y en otra variable solo el nombre de las dependencias que hayan sido desarrolladas por el equipo de desarrollo.

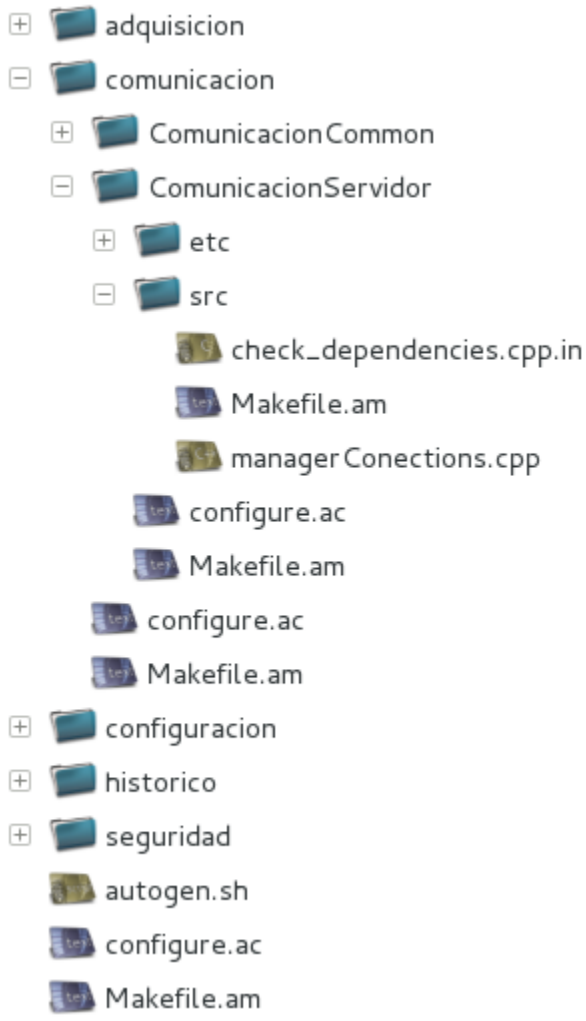

Ilustración 7: Estructura del proyecto para una aplicación.

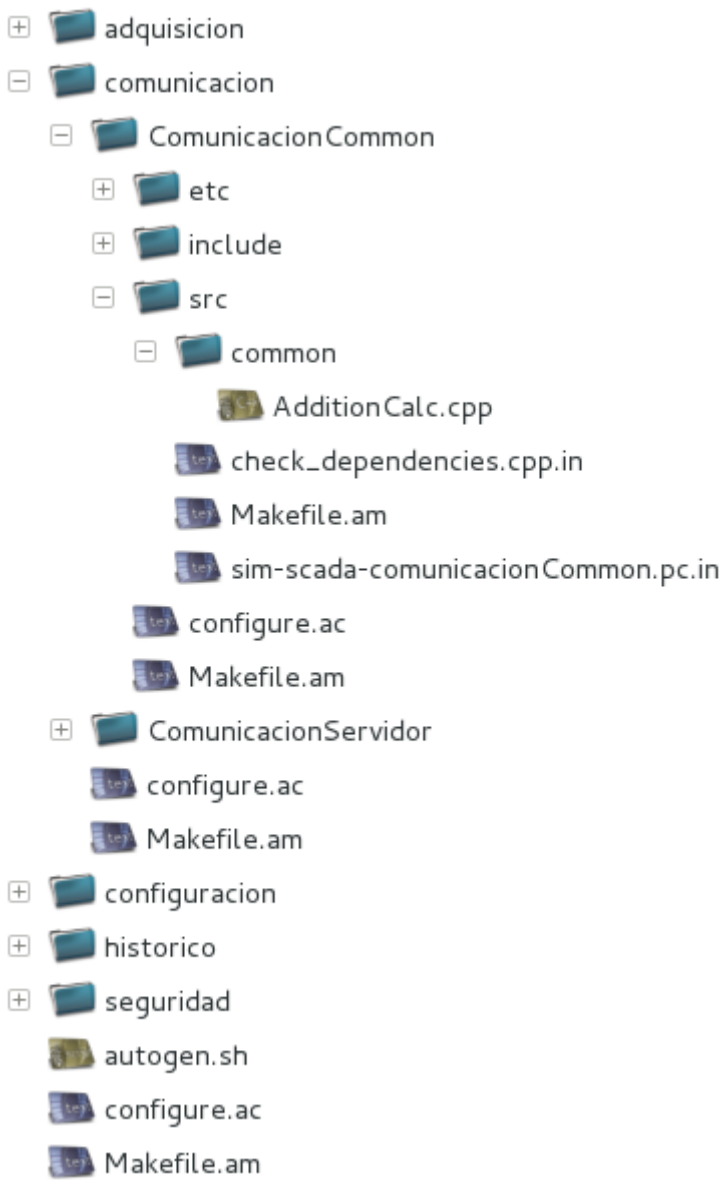

Ilustración 8: Estructura del proyecto para una biblioteca.

4.5 Prueba

Una vez que se ha culminado con la implementación de cualquier sistema, es de vital importancia realizarle un conjunto de pruebas para comprobar su correcto funcionamiento cuando se ponga a disposición de los usuarios finales. Es importante que se pruebe constantemente el software. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección.

La metodología XP divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñada por los programadores, y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñadas por el cliente final.

4.5.1 Pruebas de aceptación

El objetivo de estas pruebas es verificar los requisitos, por este motivo, los propios requisitos del sistema son la principal fuente de información a la hora de construir las pruebas de aceptación. Estas son creadas a partir de las historias de usuario. Durante una iteración la historia de usuario seleccionada en la planificación de iteraciones se convertirá en una prueba de aceptación.

Una historia de usuario puede tener más de una prueba de aceptación, tantas como sean necesarias para garantizar su correcto funcionamiento y no se considera completa hasta que no supera sus pruebas de aceptación. Una prueba de aceptación es como una caja negra. Cada una de ellas representa una salida esperada del sistema. Es responsabilidad del cliente verificar la corrección de las pruebas de aceptación y tomar decisiones acerca de las mismas.

A continuación se muestran las pruebas de aceptación propuestas a realizarse por iteración para una mayor organización.

4.5.1.1 Iteración 1

Caso de prueba de aceptación	
Código: HU1_P1	**Historia de usuario:** 1
Nombre: Generar códigos MD5.	
Descripción: Prueba para la funcionalidad de generar códigos MD5.	
Condiciones de ejecución: Es necesario incluir la cabecera openssl/md5.h. A demás es necesario incluir otras cabeceras del sistema necesarias.	
Entradas/Pasos de ejecución: Pasarle a la función la ruta de un fichero existente.	
Resultado esperado:	

Genera un código hash correspondiente al fichero.
Evaluación de la prueba: Prueba satisfactoria

Tabla 18 Prueba 1 HU Generar códigos MD5.

Caso de prueba de aceptación	
Código: HU1_P2	**Historia de usuario:** 1
Nombre: Generar códigos MD5.	
Descripción: Prueba para la funcionalidad de generar códigos MD5.	
Condiciones de ejecución: Es necesario incluir la cabecera openssl/md5.h. A demás es necesario incluir otras cabeceras del sistema necesarias.	
Entradas/Pasos de ejecución: Pasarle a la función la ruta de un fichero que no existe.	
Resultado esperado: Retorna un valor vacío.	
Evaluación de la prueba: Prueba satisfactoria	

Tabla 19 Prueba 2 HU Generar códigos MD5.

Caso de prueba de aceptación	
Código: HU2_P1	**Historia de usuario:** 2
Nombre: Chequear dependencias.	
Descripción: Prueba para la funcionalidad de chequear dependencias.	
Condiciones de ejecución: Es necesario incluir la cabecera dlfcn.h que nos proporciona las funciones necesarias para acceder a las bibliotecas.	
Entradas/Pasos de ejecución: El mapa debe contener el nombre de al menos una biblioteca existente.	

Dicha biblioteca debe contener una función *check_dependencies* que retorne verdadero. El MD5 almacenado debe coincidir con el MD5 que se genera en tiempo de ejecución.
Resultado esperado: La aplicación se ejecutará.
Evaluación de la prueba: Prueba satisfactoria

Tabla 20 Prueba 1 HU Chequear dependencias.

Caso de prueba de aceptación	
Código: HU2_P2	**Historia de usuario:** 2
Nombre: Chequear dependencias.	
Descripción: Prueba para la funcionalidad de chequear dependencias.	
Condiciones de ejecución: Es necesario incluir la cabecera dlfcn.h que nos proporciona las funciones necesarias para acceder a las bibliotecas.	
Entradas/Pasos de ejecución: Que al menos no se cumpla una de las siguientes condiciones: El mapa debe contener el nombre de al menos una biblioteca existente. Dicha biblioteca debe contener una función *check_dependencies* que retorne verdadero. El MD5 almacenado debe coincidir con el MD5 que se genera en tiempo de ejecución.	
Resultado esperado: La aplicación mostrará un mensaje de error: “Error. Dependencias corruptas o no existen.”	
Evaluación de la prueba: Prueba satisfactoria	

Tabla 21 Prueba 2 HU Chequear dependencias.

Caso de prueba de aceptación	
Código: HU3_P1	**Historia de usuario:** 3
Nombre: Modificar el fichero de configuración de la auto-construcción.	
Descripción: Prueba para la funcionalidad de modificar el fichero de configuración de la auto-	

construcción.
Condiciones de ejecución: El fichero de configuración de la auto-construcción configure.ac que se quiere modificar debe existir.
Entradas/Pasos de ejecución: Almacenar la ruta hacia las bibliotecas. Almacenar el/los nombre(s) de las bibliotecas de las que depende la aplicación o biblioteca en cuestión. Dentro de AC_CONFIG_FILES adicionar *src/check_dependencies.cpp.*
Resultado esperado: El proyecto se compilará sin lanzar ningún error.
Evaluación de la prueba: Prueba satisfactoria

Tabla 22 Prueba 1 HU Modificar el fichero de configuración de la auto-construcción.

Caso de prueba de aceptación	
Código: HU3_P2	**Historia de usuario:** 3
Nombre: Modificar el fichero de configuración de la auto-construcción.	
Descripción: Prueba para la funcionalidad de modificar el fichero de configuración de la auto-construcción.	
Condiciones de ejecución: El fichero de configuración de la auto-construcción configure.ac que se quiere modificar debe existir.	
Entradas/Pasos de ejecución: Dentro de AC_CONFIG_FILES no adicionar *src/check_dependencies.cpp.*	
Resultado esperado: Al estar compilando el proyecto lanzará un error de que no puede encontrar el fichero *src/check_dependencies.cpp.*	
Evaluación de la prueba: Prueba satisfactoria	

Tabla 23 Prueba 2 HU Modificar el fichero de configuración de la auto-construcción.

Caso de prueba de aceptación	
Código: HU4_P1	**Historia de usuario:** 4
Nombre: Modificar el fichero *src/Makefile.am* perteneciente al módulo o biblioteca.	
Descripción: Prueba para la funcionalidad de modificar el fichero *src/Makefile.am* perteneciente al módulo o biblioteca.	
Condiciones de ejecución: El fichero *src/Makefile.am* perteneciente al módulo o biblioteca que se quiere modificar debe existir.	
Entradas/Pasos de ejecución: Eliminar del nombre_aplicación_SOURCES el fichero *check_dependencies.cpp*.	
Resultado esperado: Se compila el proyecto, pero cuando se ejecute la aplicación muestra un mensaje de error.	
Evaluación de la prueba: Prueba satisfactoria	

Tabla 24 Prueba 1 HU Modificar el fichero src/Makefile.am perteneciente al módulo o biblioteca.

4.6 Conclusiones

En este capítulo se registraron las bases necesarias para la implementación de la aplicación. Los diferentes diagramas mostrados en el capítulo contribuyeron a poseer un mejor entendimiento del sistema. Se diseñaron y ejecutaron las pruebas de aceptación que permitieron demostrar que el sistema cumple con las funcionalidades que el cliente definió, concluyendo que la propuesta de solución cumple con los resultados esperados que se trazaron al inicio de la investigación.

Conclusiones

Al término de la investigación, “Mecanismo para el análisis de la integridad de las bibliotecas dinámicas de los módulos del sistema SCADA GALBA”, se llegaron a las siguientes conclusiones:

- Se desarrolló una solución consistente en un mecanismo, para los distintos módulos del SCADA Guardián del Alba, que permite comprobar la integridad de las bibliotecas dinámicas para ser utilizado en el sistema SCADA GALBA.
- El mecanismo permite detectar posibles cambios o modificaciones en las bibliotecas utilizadas y recuperar el tiempo perdido a la hora de buscar posibles errores en el sistema.

Recomendaciones

- Implementar una solución que evite la inserción doble de una misma dependencia.
- Integrar el mecanismo desarrollado al proceso de compilación del SCADA GALBA.

Referencias bibliográficas

1. *ROMERAL, J. B. Y. L. Autómatas programables 1997.*
2. *VÁZQUEZ, M. H. Introducción a la Arquitectura del Guardián del ALBA 2008.*
3. *C/LINUX, E. J. Y. Librerías estáticas y dinámicas. . In., 4 febrero 2007.*
4. *A History of MTSA.*
5. *ROBÓTICA, A. In., 5 noviembre 2012.*
6. *MADRID, U. C. I. D. SISTEMAS OPERATIVOS: COMUNICACIÓN Y SINCRONIZACIÓN ENTRE PROCESOS. Available from Internet:<http://www.google.com.cu/url?sa=t&rct=j&q=ventajas+e+inconvenientes+de+las+bibliotecas+dinamicas&source=web&cd=4&ved=0CD8QFjAD&url=http%3A%2F%2Fwww.arcos.inf.uc3m.es%2F~ii_ssoo%2Fdokuwiki%2Flib%2Fexe%2Ffetch.php%3Fid%3Dmaterial_de_clase%26cache%3Dcache%26media%3Dmaterial%3Asem11.pdf&ei=YJWaUYLUOIbX0gGK-oDIAQ&usg=AFQjCNE3q2nTES_VkxKwCL8J1xsw7RlRmQ&bvm=bv.46751780,d.dmg&cad=rja>.*
7. *SYSTEMS, Z. Curso C++. 1990-2013. Available from Internet:<http://www.zator.com/Cpp/E1_4_4b.htm>.*
8. *INTECO, O. Cómo comprobar la integridad de los ficheros. Instituto Nacional de Tecnologías de la Comunicación.*
9. *MUÑZQUIZ, P. R. Sistemas operativos 2003.*
10. *Modelado UML. In., 25 de mayo de 2009.*
11. *SÁNCHEZ, M. A. M. Metodologías De Desarrollo De Software. In.*
12. *Lenguaje de Programación. In., 27 de marzo de 2009.*
13. *RODICIO, C. G. El lenguaje interpretado Bash 2013.*
14. *Bash Reference Manual. In.*
15. *OpenSSL Project«OpenSSL 1.0.0g released » lista de correo openssl-announce 18 de enero de 2012.*
16. *RHODES, T. OpenSSL. In Manual de FreeBSD.*
17. *Entornos de desarrollo Integrado. In.*
18. *HERNANDIS, J. A. Why Visual Paradigm for UML? In., 2005, vol. 8 marzo 2013.*
19. *CALCOTE, J. FREE SOFTWARE MAGAZINE 12 abril 2008.*
20. *FOUNDATION, F. S. GNU Operating system 1 mayo 2012.*

21. EMILIO A. SANCHEZ, P. L., JOSE H. CANOS. Mejorando la gestion de historias de usuario eneXtreme Programming. 2013. Available from Internet:<http://www.scribd.com/doc/62663167/Gestion-de-Historias-de-Usuario>.

22. ESCRIBANO, G. F. Introducción a Extreme Programming. .

23. JOSKOWICZ, J. Reglas y Prácticas en eXtreme Programming. 10 febrero 2008. Available from Internet:<http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf>.

Bibliografía

1. *Antonio Villalón Huerta. SEGURIDAD EN UNIX Y REDES. Julio, 2002.*
2. *Roger Farnsworth. Introduction to Information Security.1999.*
3. *Dr. Julio Cerezal Mezquita, Dr. Jorge Fiallo Rodríguez. ¿Cómo investigar en pedagogía?. 2005.*
4. *Dr. Cs. Carlos Alvarez de Zayas. METODOLOGIA DE LA INVESTIGACION CIENTIFICA. Santiago de Cuba 1995.*
5. *Darwin Durand. INTEGRIDAD DE DATOS. 2013.*
6. *VÁZQUEZ, M. H. Introducción a la Arquitectura del Guardián del ALBA 2008.*
7. *C/LINUX, E. J. Y. Librerías estáticas y dinámicas. . In., 4 febrero 2007.*
8. *INTECO, O. Cómo comprobar la integridad de los ficheros. Instituto Nacional de Tecnologías de la Comunicación.*
9. *MUÑZQUIZ, P. R. Sistemas operativos2003.*
10. *Modelado UML. In., 25 de mayo de 2009.*
11. *SÁNCHEZ, M. A. M. Metodologías De Desarrollo De Software. In.*
12. *Lenguaje de Programación. In., 27 de marzo de 2009.*
13. *RODICIO, C. G. El lenguaje interpretado Bash 2013.*
14. *Bash Reference Manual. In.*
15. *OpenSSL Project«OpenSSL 1.0.0g released » lista de correo openssl-announce 18 de enero de 2012.*
16. *RHODES, T. OpenSSL. In Manual de FreeBSD.*
17. *Entornos de desarrollo Integrado. In.*
18. *HERNANDIS, J. A. Why Visual Paradigm for UML? In., 2005, vol. 8 marzo 2013.*
19. *CALCOTE, J. FREE SOFTWARE MAGAZINE 12 abril 2008.*
20. *FOUNDATION, F. S. GNU Operating system 1 mayo 2012.*
21. *EMILIO A. SANCHEZ, P. L., JOSE H. CANOS. Mejorando la gestion de historias de usuario eneXtreme Programming. 2013. Available from Internet:<http://www.scribd.com/doc/62663167/Gestion-de-Historias-de-Usuario>.*
22. *ESCRIBANO, G. F. Introducción a Extreme Programing.*

Anexos

Anexo 1: Tareas abordadas en la primera iteración

Tarea	
Número de la tarea: 1	**Número de HU:** 1
Nombre de la tarea: Crear las variables necesarias.	
Tipo de tarea: Desarrollo	
Fecha de inicio: 18/02/2013	**Fecha de fin:** 19/02/2013
Programador responsable: Liuba Sosa Fridrij	
Descripción: Se crean todas las variables donde se almacenará el contenido, el descriptor y el tamaño del fichero además del código MD5.	

Tabla 25 Tarea 1: Crear las variables necesarias

Tarea	
Número de la tarea: 2	**Número de HU:** 1
Nombre de la tarea: Obtener el descriptor del fichero en cuestión.	
Tipo de tarea: Desarrollo	
Fecha de inicio: 20/02/2013	**Fecha de fin:** 21/02/2013
Programador responsable: Liuba Sosa Fridrij	
Descripción: Se obtiene el descriptor del fichero utilizando la función open, el cual se almacenará en una variable para su posterior uso.	

Tabla 26 Tarea 2: Obtener el descriptor del fichero en cuestión.

Tarea	
Número de la tarea: 3	**Número de HU:** 1
Nombre de la tarea: Obtener el tamaño del fichero.	
Tipo de tarea: Desarrollo	
Fecha de inicio: 22/02/2013	**Fecha de fin:** 22/02/2013
Programador responsable: Liuba Sosa Fridrij	

Descripción: Se obtiene el tamaño del fichero utilizando la función fstat, el cual se almacenará en una variable para su uso posterior.

Tabla 27 Tarea 3: Obtener el tamaño del fichero.

Tarea	
Número de la tarea: 4	**Número de HU:** 1
Nombre de la tarea: Cargar en memoria dicho fichero.	
Tipo de tarea: Desarrollo	
Fecha de inicio: 25/02/2013	**Fecha de fin:** 26/02/2013
Programador responsable: Liuba Sosa Fridrij	
Descripción: Se obtiene el contenido del fichero mediante la función mmap, el cual se almacenará en una variable para su uso posterior.	

Tabla 28 Tarea 4: Cargar en memoria dicho fichero.

Tarea	
Número de la tarea: 5	**Número de HU:** 1
Nombre de la tarea: Generar código MD5 puro.	
Tipo de tarea: Desarrollo	
Fecha de inicio: 27/02/2013	**Fecha de fin:** 28/02/2013
Programador responsable: Liuba Sosa Fridrij	
Descripción: Se pasan las variables con el contenido y el tamaño del fichero y la variable donde se almacenará el código MD5 generado a la función que ofrece la biblioteca libssl.	

Tabla 29 Tarea 5: Generar código MD5 puro.

Tarea	
Número de la tarea: 6	**Número de HU:** 1
Nombre de la tarea: Convertir el resultado de dicha función a valores alfanuméricos y retornarlo.	
Tipo de tarea: Desarrollo	
Fecha de inicio: 29/02/2013	**Fecha de fin:** 1/03/2013
Programador responsable: Liuba Sosa Fridrij	

Descripción: Se convierte el código generado a valores alfanuméricos entendibles.

Tabla 30 Tarea 6: Convertir el resultado de dicha función a valores alfanuméricos y retornarlo.

Tarea	
Número de la tarea: 7	**Número de HU:** 2
Nombre de la tarea: Crear un mapa para almacenar el nombre de la biblioteca y el valor del MD5	
Tipo de tarea: Desarrollo	
Fecha de inicio: 4/03/2013	**Fecha de fin:** 8/03/2013
Programador responsable: Liuba Sosa Fridrij	
Descripción: Se crea un mapa en el cual se insertará como llave el nombre de la biblioteca y el valor el MD5 generado en tiempo de compilación.	

Tabla 31 Tarea 7: Crear un mapa para almacenar el nombre de la biblioteca y el valor del MD5.

Tarea	
Número de la tarea: 8	**Número de HU:** 2
Nombre de la tarea: Verificar existencia de la biblioteca.	
Tipo de tarea: Desarrollo	
Fecha de inicio: 11/03/2013	**Fecha de fin:** 15/03/2013
Programador responsable: Liuba Sosa Fridrij	
Descripción: Mediante la función dlopen se genera un apuntador hacia la biblioteca en caso de que esta exista realmente.	

Tabla 32 Tarea 8: Verificar existencia de la biblioteca.

Tarea	
Número de la tarea: 9	**Número de HU:** 2
Nombre de la tarea: Verificar la existencia de la función definida que chequea las dependencias en la biblioteca.	
Tipo de tarea: Desarrollo	
Fecha de inicio: 11/03/2013	**Fecha de fin:** 15/03/2013
Programador responsable: Liuba Sosa Fridrij	

Descripción: Mediante la función dlsym se verifica la existencia de la función "check_dependencies" dentro de la biblioteca que se esté procesando en ese momento.

Tabla 33 Tarea 9: Verificar la existencia de la función definida que chequea las dependencias en la biblioteca.

Tarea	
Número de la tarea: 10	**Número de HU:** 2
Nombre de la tarea: Comprobar la igualdad entre el MD5 generado en tiempo de compilación y el MD5 generado en tiempo de ejecución.	
Tipo de tarea: Desarrollo	
Fecha de inicio: 18/03/2013	**Fecha de fin:** 22/03/2013
Programador responsable: Liuba Sosa Fridrij	
Descripción: Se comprueba que el MD5 que se almacenó en tiempo de compilación sea igual que el MD5 que se genera en tiempo de ejecución.	

Tabla 34 Tarea 10: Comprobar la igualdad entre el MD5 generado en tiempo de compilación y el MD5 generado en tiempo de ejecución.

Anexo 2: Tareas abordadas en la segunda iteración

Tarea	
Número de la tarea: 11	**Número de HU:** 3
Nombre de la tarea: Almacenar la ruta de las bibliotecas.	
Tipo de tarea: Desarrollo	
Fecha de inicio: 25/03/2013	**Fecha de fin:** 27/03/2013
Programador responsable: Liuba Sosa Fridrij	
Descripción: Se almacena en una variable la ruta de las bibliotecas de las que depende para luego ser utilizadas más adelante para poder generar el MD5.	

Tabla 35 Tarea 11: Almacenar la ruta de las bibliotecas.

Tarea	
Número de la tarea: 12	**Número de HU:** 3
Nombre de la tarea: Almacenar el/los nombre(s) de las bibliotecas de las que depende la	

aplicación o biblioteca en cuestión.	
Tipo de tarea: Desarrollo	
Fecha de inicio: 25/03/2013	**Fecha de fin:** 27/03/2013
Programador responsable: Liuba Sosa Fridrij	
Descripción: Se almacenan los nombres de las bibliotecas de las que depende para que junto con la ruta de donde está dicha biblioteca se pueda acceder hasta ellas y generar el código MD5.	

Tabla 36 Tarea 12: Almacenar el/los nombre(s) de las bibliotecas de las que depende la aplicación o biblioteca en cuestión.

Tarea	
Número de la tarea: 13	**Número de HU:** 3
Nombre de la tarea: Adicionar src/check_dependencies.cpp dentro de AC_CONFIG_FILES.	
Tipo de tarea: Desarrollo	
Fecha de inicio: 27/03/2013	**Fecha de fin:** 29/03/2013
Programador responsable: Liuba Sosa Fridrij	
Descripción: Se adiciona dicho fichero para que se pueda generar check_dependencies.cpp de la plantilla check_dependencies.cpp.in y se le pueda hacer todas las modificaciones necesarias.	

Tabla 37 Tarea 13: Adicionar src/check_dependencies.cpp dentro de AC_CONFIG_FILES.

Tarea	
Número de la tarea: 14	**Número de HU:** 4
Nombre de la tarea: Adicionar al _SOURCES el fichero check_dependencies.cpp.	
Tipo de tarea: Desarrollo	
Fecha de inicio: 1/04/2013	**Fecha de fin:** 5/04/2013
Programador responsable: Liuba Sosa Fridrij	
Descripción: Se adiciona el fichero check_dependencies.cpp al código de la aplicación o biblioteca para que esta contenga las funcionalidades necesarias para chequear las dependencias.	

Tabla 38 Tarea 14: Adicionar al _SOURCES el fichero check_dependencies.cpp.

Tarea	
Número de la tarea: 15	**Número de HU:** 4
Nombre de la tarea: Adicionar una regla para insertar dentro de check_dependencies.cpp el nombre de la biblioteca de la cual depende y su respectivo MD5 antes de generar el fichero check_dependencies.o.	
Tipo de tarea: Desarrollo	
Fecha de inicio: 8/04/2013	**Fecha de fin:** 12/04/2013
Programador responsable: Liuba Sosa Fridrij	
Descripción: Se adiciona la regla para poder modificar el fichero check_dependencies.cpp antes de ser compilado.	

Tabla 39 Tarea 15: Adicionar una regla para insertar dentro de check_dependencies.cpp el nombre de la biblioteca de la cual depende y su respectivo MD5 antes de generar el fichero check_dependencies.o.