Vol. III, No. 2, mayo - agosto, 2012, pp. 43 - 49

Buenas prácticas sobre gestión de configuración en proyectos con metodología MDA

Manuel Morejón Espinosa
Correo electrónico:mmorejon@ceis.cujae.edu.cu
Instituto Superior Politécnico José Antonio Echeverría, Cujae, La Habana, Cuba

Artículo de Reflexión

Resumen
La Gestión de Configuración de Software (GCS) forma parte de los procesos que intervienen en el desarrollo de software. Su principal meta es la de coordinar este desarrollo minimizando la confusión al máximo. Para alcanzar la meta se realizan actividades como: identificación de elementos, control de cambios, control de versiones, auditorías y emisión de reportes. Dentro de las aplicaciones empresariales se emplea la metodología de guiar el desarrollo de software a partir del modelo del sistema (MDA). El empleo de este paradigma busca el objetivo de aumentar la velocidad de desarrollo a través de la generación automática de gran parte del código; así como mantener actualizada la documentación del sistema en relación con el código existente. La manera de llevar a cabo las actividades presentes en el proceso de GCS difieren entre un proyecto que emplea el paradigma MDA y otro en que no se emplea. Existen aspectos que deben tenerse en cuenta durante la puesta en marcha de cada una de las actividades descritas en el proceso de gestión de configuración. En este aspecto el autor identifica consideraciones a tener en cuenta en cada área del proceso, para garantizar la obtención de una guía de cómo proceder ante proyectos que utilizan este paradigma de desarrollo.

Palabras clave: gestión de configuración de software, arquitectura guiada por modelos, cambios

Recibido: 18 de marzo del 2012 Aprobado: 29 de abril del 2012

INTRODUCCIÓN

Las grandes, medianas y pequeñas empresas que han tenido éxito dentro de la industria de desarrollo de software, se caracterizan por tener muy bien definido cada proceso que realizan, siendo esta una buena arma para poder enfrentar cada uno de los inconvenientes y dificultades presentados a diario. Los inconvenientes y las dificultades son, en muchos de los casos, cambios inesperados a realizar dentro de los productos de la empresa [1 - 3].

La GCS es un componente importante de garantía de la calidad de software. Para garantizar esta calidad se necesita realizar un conjunto de tareas dentro de la organización. Se tienen identificadas cinco tareas estándares que debe realizar toda organización: identificación, control de versiones, control de cambios, auditoría de configuración y generación de informes. Es necesario conocer que cada tarea pueda ser implementada de manera distinta según el proyecto en curso. Todo depende de las necesidades del grupo de trabajo, entorno de desarrollo y de la envergadura del proyecto. [4 - 5]

Por tal motivo es preciso que en cada grupo de desarrollo de software exista como mínimo una persona encargada de la GCS que tendrá la responsabilidad de adecuar los procesos (las técnicas, herramientas y procedimientos) existentes a las particularidades que presente la entidad.

ARQUITECTURA GUIADA POR MODELOS

Una de las principales primicias de la Ingeniería de Software (IS) establece: "Mientras más rápido empiezas, más te demoras", sin embargo, a los programadores no les agrada la idea de aplazar la hora de escribir códigos. [2 - 6] Por tal motivo, es frecuente intentar tomar atajos en el proceso de IS. El tomar atajos puede traer como consecuencia, un atropello de etapas durante el modelado del negocio. Por consiguiente, en la captura de requisitos no se logran identificar u obtener todos los requisitos existentes. A la tendencia descrita se le debe incorporar un componente relacionado con el proceso de desarrollo de software, que plantea como característica de cada producto, un

comportamiento cíclico e incremental, provocando en cada proyecto tendencias a cambios a través de las distintas etapas de su desarrollo. [7]

La idea clave que establece el paradigma MDA es desarrollar las aplicaciones guiadas por los modelos del software. Propone basar el desarrollo de software en modelos especificados utilizando UML, con el objetivo de, a partir de estos modelos, realizar transformaciones que generen códigos o nuevos modelos con características de una tecnología en particular. [8] Esto implica no solo que la documentación del proyecto será consecuente con el mismo y rendirá grandes frutos en la arquitectura; sino que también se obtendrán beneficios importantes en aspectos fundamentales como son la productividad, la portabilidad y el mantenimiento.

GESTIÓN DE CONFIGURACIÓN DE SOFTWARE

La GCS forma parte de los procesos que intervienen en el desarrollo de software. Son muchas las definiciones existentes sobre esta disciplina. Todo especialista que la ha definido ha aportado nuevos puntos de vista, así como tareas a tener en cuenta. En ocasiones, pudieran existir diferencias dentro de estos conceptos en cuanto a nombres o prioridades entre tareas a realizar, sin embargo, todos señalan la importancia de esta disciplina. Roger S. Pressman la definió de la siguiente manera: "El arte de coordinar el desarrollo de software para minimizar la confusión se denomina gestión de configuración. La gestión de configuración es el arte de identificar, organizar y controlar las modificaciones que sufre el software que construye un equipo de programación. La meta es maximizar la productividad minimizando los errores". [2]

La GCS abarca un conjunto de actividades y técnicas para iniciar, evaluar y controlar los cambios del producto de software durante y después del proceso de desarrollo. Haciendo énfasis en el control de la configuración dentro de la administración de producción de software. Entre sus principales funciones se encuentran el velar que exista: una documentación referente a los cambios realizados y productos que de alguna manera no ocasionen la ruptura de la integridad del software. De manera adicional, brinda garantía de la calidad del software, lo cual influye en todas las fases del proceso de IS.

La GCS no tiene establecida métricas cuantitativas que evalúen los resultados o que permitan conocer el estado actual de un grupo de desarrollo o empresa. Sin embargo, se pueden evaluar los niveles de madurez alcanzados en esta disciplina utilizando métricas cualitativas. Estas métricas pueden ser preguntas estructuradas en función de los aspectos a evaluar [9] y algunas serán vistas en el documento en epígrafes posteriores.

CARACTERÍSTICAS DEL PROYECTO A ANALIZAR

El sistema tomado como referencia en el presente estudio es un proyecto empresarial basado en el estándar establecido para las aplicaciones desarrolladas con Java. [10] En la figura 1 se muestra la división de cada una de las capas presentes en la aplicación. De manera adicional, el sistema utiliza el paradigma MDA para la generar código a partir del modelo del dominio.

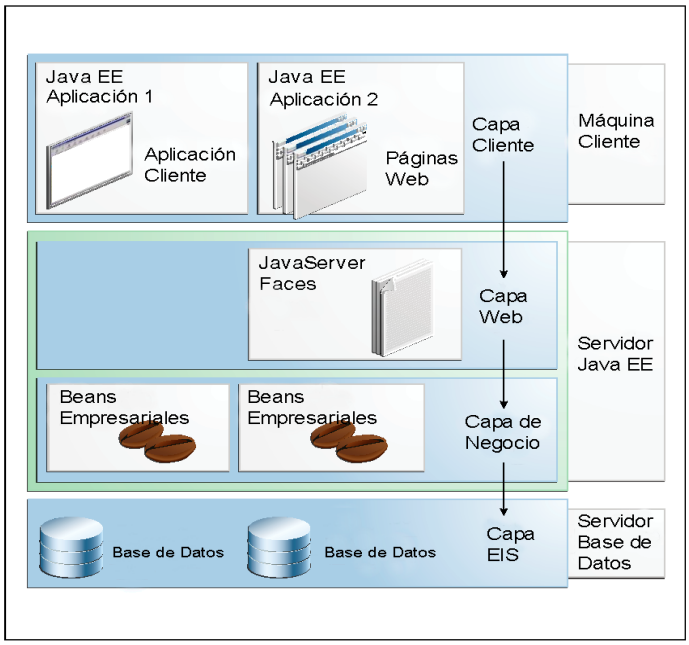

Fig. 1. Estructura de un proyecto empresarial.

Capa de negocio y almacenamiento

Para llevar a cabo el paradigma MDA se utilizó el *framework* AndroMDA. AndroMDA posee un repositorio de cartuchos *cartridges* los cuales se encargan de generar código específico para el *framework* Enterprice Java Beans (EJB) 2.0. El código es generado a partir del modelo UML creado en la herramienta MagicDraw. Ver figura 2.

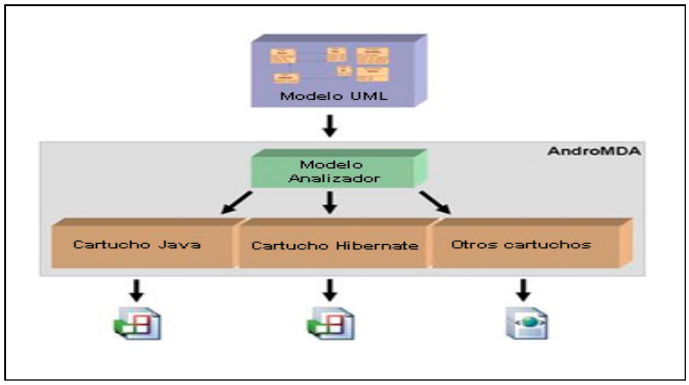

Fig. 2. Flujo de trabajo de la herramienta AndroMDA.

De esta forma se construye un primer proyecto (proyecto - servidor) encargado de gestionar toda la lógica del negocio. El proyecto cuenta con la misma estructura que los demás proyectos desarrollados con el *framework* EJB 2.0. La única diferencia es que la mayor parte de los ficheros son generados de manera automática. Quedaría solamente por parte de programador, el desarrollo de cada una de las firmas de los métodos definidos en modelo UML.

Dentro del proyecto también queda generado el fichero que contiene la estructura de la base de datos. Al configurar la herramienta de transformación se le especifica el gestor de base de datos con el que se desea trabajar. De manera tal que al terminar queda creada la base de datos lista para utilizarse. Para este caso se utilizó el gestor PostgreSQL.

Capa web y capa cliente

La capa web es confeccionada completamente por los programadores del equipo. Hasta el momento, la herramienta AndroMDA no cuenta con la posibilidad de generar, a partir de modelos, código fuente para la tecnología Java ServerFaces en casos de uso complejos. De esta forma se construye un segundo proyecto (proyecto-Web) encargado de gestionar la lógica relacionada con la interfaz de usuario. Utiliza los servicios que le brinda el proyecto servidor para realizar las diferentes actividades.

La capa cliente queda conformada con las páginas web que consulta el usuario relacionadas con el proyecto web. De manera adicional existe una aplicación de escritorio desarrollada por el grupo de trabajo dedicada a la administración del proyecto. Esta aplicación también es la encargada de gestionar los usuarios y los distintos tipos de privilegios con que cuenta cada uno.

Queda conformado un tercer proyecto (proyecto-administración) encargado de la seguridad del sistema. El mismo se desarrolló utilizando las clases visuales presentes en el paquete SWING, propuestas por la tecnología Java.

GCS DENTRO DEL PROYECTO

En el epígrafe anterior han quedado bien identificadas las características del proyecto con el cual se estará trabajando. El siguiente paso es desarrollar cada una de las actividades identificadas dentro de la GCS en el proyecto actual. Dentro de las actividades identificadas por la GCS están: identificación de elementos, control de versiones, control de cambios, auditorías, emisión de reportes.

Identificación de elementos

Son muchos los Elementos de Configuración de Software (ECS) que se pudieran estar reconocidos dentro del proyecto. Pueden identificarse los documentos, el código fuente, los casos de prueba, entre otros. Sin embargo, en el presente trabajo se mencionarán aquellos elementos que, por sus características, manifiesten la integridad de la aplicación.

En este aspecto se hace referencia a la línea base del proyecto. Al conformarse la línea base hay que almacenar todos los ECS que permitan volver a estructurar el sistema. En la aplicación tomada como prueba hay que tener en cuenta los siguientes elementos:

1.**Proyecto-servidor**, debe quedar correctamente almacenado al código fuente de este proyecto para garantizar el estado actual de las implementaciones de las firmas de los métodos relacionada con la lógica del negocio.

2. **Modelo del negocio**, debe quedar almacenado junto con el código fuente del proyecto-servidor el modelo por el cual fue generado el código fuente haciendo uso de la herramienta AndroMDA. Es en el modelo donde quedan definido las diferentes firmas de los métodos de los servicios utilizados posteriormente por la aplicación cliente.

3. Para el caso específico de la **aplicación descrita** en el epígrafe anterior, este modelo queda almacenado en un fichero con extensión xmi. De utilizarse otra herramienta pudiera cambiar la extensión. Sin embargo, sin importar el formato en que esté almacenado, lo importante es tener la información relacionada con el modelo de negocio, de manera tal, que pueda volverse a cargar en la herramienta CASE. De no realizarse esta identificación correctamente pudiera intentarse reconstruir la aplicación en un punto donde el modelo de negocio sea de fechas anteriores al código generado. Esto daría como consecuencia, que al generar código fuente a partir de este modelo, se eliminarían todas aquellas implementaciones de firmas de métodos existentes después de esa fecha, lo cual atrasaría el trabajo realizado por los programadores del equipo de trabajo.

4. **Base de datos**, debe realizarse una salva de la base de datos operacional actual. Es cierto que dentro del código fuente del proyecto-servidor existe un fichero con el *script* de creación de la base de datos. Con este fichero se podría crear una base de datos en blanco. Sin embargo, cuando el sistema se encuentra en funcionamiento las cosas suceden de otra forma para garantizar no perder la información actual. Al realizarse cambios que implican modificaciones en la estructura de la base de datos sucede lo siguiente: Se hacen cambios en el modelo de datos, después es generado el código asociado al modelo y, por último, se realizan consultas directamente en la base de datos que modifiquen su estructura.

5. De ser necesario regresar a **versiones anteriores** de la aplicación, es importante tener una salva con el estado de la base de datos realizado justamente en aquel momento. De lo contrario no se podría recrear exactamente dicha versión. No podrían funcionar los servicios creados puesto que harían referencia a estructuras no existentes en la base de datos actual.

6. **Proyecto-web**, debe quedar almacenado el código fuente de este proyecto. Es dentro de este proyecto donde se realizan las peticiones a los servicios existente en el proyecto-servidor. De no guardar correctamente su estado actual, pudiera suceder que al regresar a versiones anteriores no coincida la lógica establecida en el servidor con los casos de uso descritos en las interfaces webs. De igual manera pudiera ocurrir que se intente acceder a servicios que no existían o que no se habían implementado en versiones previas.

7. **Proyecto-administración**, debe quedar almacenado el código fuente de este proyecto. Las razones son las mismas relacionadas con el proyecto web. Es importante tener en cuenta que la lógica de negocio es una sola y todos los proyectos que dependan de ella deben ser correctamente guardados para garantizar su integridad. Vale la pena aclarar que pudieran existir dentro del proyecto algunas dependencias adicionales a la lógica de negocio, como son: servicios web brindados desde la propia aplicación y emisión de reportes empleando una herramienta externa. A continuación se expondrán algunas recomendaciones para cada caso.

8. **Servicios web**, debe quedar almacenado el código de este proyecto de existir. Las razones son las mismas por las cuales se almacenan los códigos fuente de los proyectos web y de administración.

9. **Diseños de reportes**, deben almacenarse los diseños por los cuales fueron realizados los reportes de la aplicación. Esta pequeña pieza de la aplicación suele en ocasiones pasarse por alto. Como resultado de este olvido pudiese provocarse un incremento de las horas de trabajo dedicadas a una tarea ya antes realizada.

Control de versiones

Un gran número de equipos de desarrollo de software han incluido, como parte de su trabajo diario, el manejo de sistemas controladores de versiones (SCV) para garantizar el correcto almacenamiento de la evolución de sus productos. [11 - 12] Ante los posibles cambios que pudieran aparecer, siempre es importante estar preparados. Sin embargo, en la actualidad existen numerosos controladores de versiones de código, cada uno con sus fortalezas y debilidades. Estos se agrupan en tres grandes grupos: locales (figura 3), centralizados (figura 4) y distribuidos (figura 5).

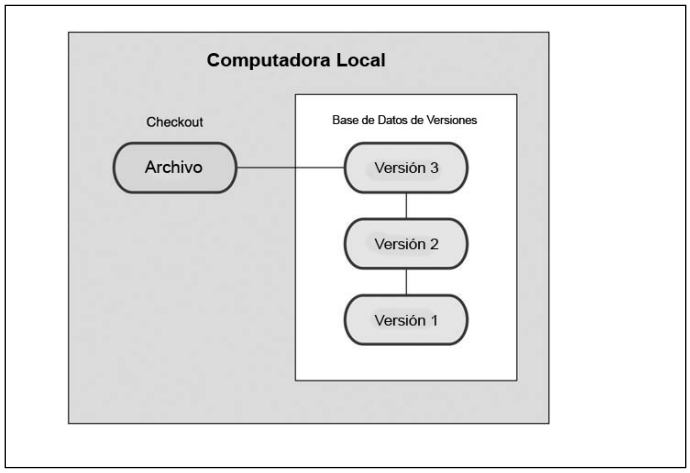

Fig. 3. Arquitectura de los SCV locales.

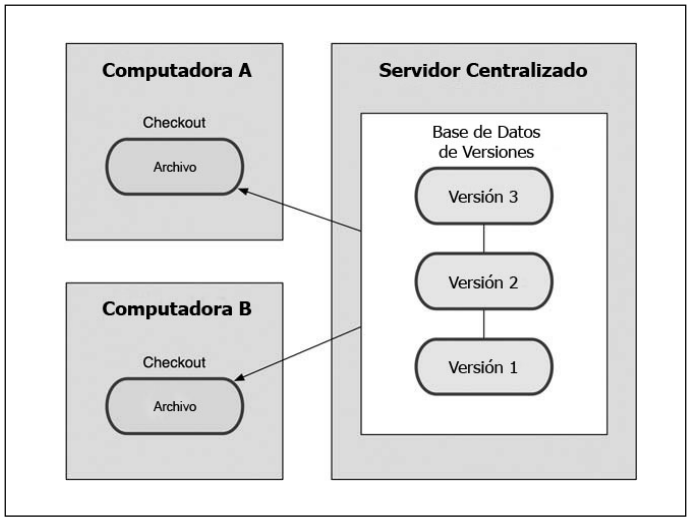

Fig. 4. Arquitectura de los SCV centralizados.

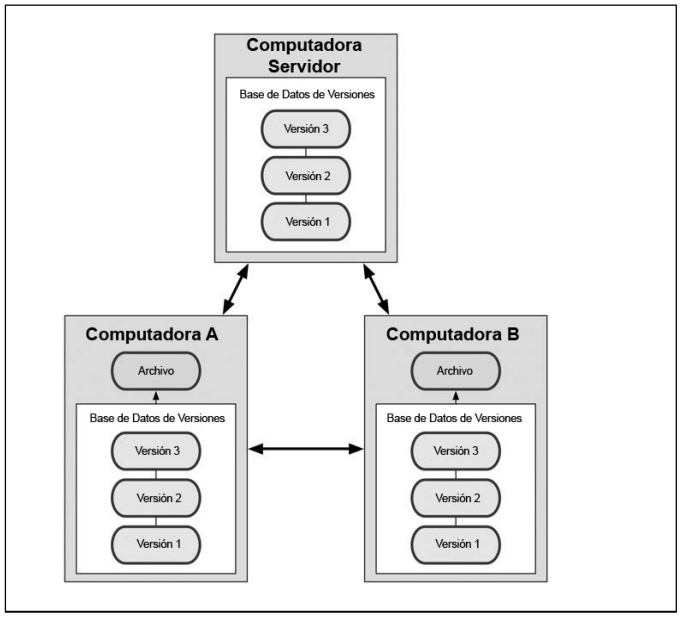

Fig. 5. Arquitectura de los SCV distribuidos.

Es difícil decidir cuál es el SCV que más se adapta a las necesidades del proyecto. En muchos de los casos depende de las necesidades de la entidad y de los intereses de los jefes. De cualquier manera, el interés del trabajo no es seleccionar una herramienta u otra, sino saber algunas consideraciones respecto al actual proyecto.

Un primer paso es colocar cada uno de los proyectos mencionados como ECS en el controlador de versiones de la entidad. Se pudiera pensar que esto es todo, pero no es así. Hay que tener en cuenta algunas cuestiones específicas:

El modelo de negocio. Este no puede ser comparado con versiones anteriores. El modelo no es código Java que pueda ser trabajado de igual manera que el resto del código existente en el proyecto. Es utilizado para generar código a partir de él solamente. Pudiera ocurrir que queden salvados en el SCV cada una de las fases por las cuales pasan las distintas clases y sin embargo no se salva el modelo. Es importante tener en cuenta que no se podrá comparar el fichero actual con versiones anteriores, lo cual es un inconveniente porque tendrían que estar bien claros los programadores de los estados anteriores antes de regresar a esas versiones. Se aconseja realizar una descripción detallada del estado del modelo siempre que sea salvado, tanto para formar parte de una nueva versión en la línea base del proyecto o porque se realicen salvas periódicas.

Código autogenerado. La aplicación utiliza la herramienta AndroMDA para generar de manera automática código en el proyecto - servidor. Este código autogenerado cambia con mucha rapidez a consecuencia de las diferentes modificaciones realizadas en el modelo de datos. Este código es adicionado una primera vez al SCV cuando genera la primera vez. Después es aconsejable indicarle al SCV que ignore los cambios que se realizan sobre estos ficheros.

Revista Cubana de Ingeniería. Vol. III, No. 2, mayo - agosto, 2012, pp.43 -49, ISSN 2223 -1781

De esta forma, solamente se actualizarán las implementaciones de los métodos definidos en el modelo de datos.

Se pudiera pensar que se pierde la historia de estas clases autogeneradas y es cierto, sin embargo, no son necesarias almacenar. La causa es que cada vez que se deseen obtener, se pueden lograr haciendo uso del modelo.

Control de usuarios. Los repositorios utilizados para almacenar la información referente al proyecto deben incluir seguridad relacionada con los usuarios. Debe existir una identificación de cada uno de los usuarios que tienen acceso a los repositorios, así como una definición de los lugares a los cuáles tiene acceso y a cuáles no. Esto es referente a que no todos los usuarios tienen las mismas responsabilidades con el proyecto. Existen SCV que permiten crear grupos de usuarios y asignarles diferentes tipos de recursos entre grupos. De esta forma pudiera ganar en especialización el grupo de trabajo; aunque es cierto que siempre que se efectúe un cambio, este puede ser revertido a versiones anteriores, por ello, es mejor evitar derrochar horas de trabajo corrigiendo la información cuando este proceso puede ser controlado.

Control de cambios

Dentro del Control de Cambios (CC) se combinan los procedimientos humanos y las herramientas automáticas en aras de proporcionar el mecanismo para el control de los cambios realizados dentro del proyecto. [4] Es el proceso utilizado para gestionar la preparación, justificación, evaluación, coordinación, disposición e implementación de los cambios ingenieriles propuestos. Así como las desviaciones que afectan a los ECS y a las líneas base. [13]

Para llevar a cabo el cumplimiento de esta tarea dentro de la organización, el jefe del área relacionada con la GCS, debe tener un modelo donde se indiquen los pasos a seguir en caso de recibir una solicitud de cambio. En este modelo deben quedar reflejados todos los posibles caminos alternativos por los cuales puede pasar la solicitud. De igual manera deben estar presentes todas las acciones realizadas por el equipo de desarrollo en torno a la solicitud.

Además de la existencia de este modelo, los equipos de trabajo en la actualidad utilizan herramientas que apoyan el seguimiento de cada una de las incidencias llegadas al grupo de desarrollo. Dentro de este tipo de herramientas aparece Bugzilla, Mantis Bug Tracker, entre otras. En dependencia de las características de los integrantes del equipo y de la tecnología que emplean para programar, se debe seleccionar uno de estos sistemas que ayuden al seguimiento y control de las incidencias dentro del grupo. Existen además, sistemas creados con el objetivo de guiar el proceso de desarrollo de una o más aplicaciones durante todo su ciclo de vida. En caso de utilizar esta última variante se emplea la propia herramienta contenida en este sistema integrador.

El tener implantado dentro del grupo de trabajo este tipo de herramienta garantiza algunos aspectos, como son:

1. Mejora el tiempo de trabajo de los programadores, permitiéndoles observar sus tareas desde cualquier lugar.
2. Permite tener un mayor control sobre el tiempo que los programadores del equipo emplean en resolver una incidencia.
3. Proporciona al usuario la facilidad de observar en qué etapa del proceso se encuentra la incidencia reportada.
4. Disminuye el número de incidencias reportadas por vía telefónica.
5. Elimina el uso de papel en relación con la recepción de solicitudes de incidencias.

El CC cuenta con un Comité de Control de Cambios (CCC). Este comité se encuentra formados por expertos en los proyectos con la responsabilidad de tomar decisiones en relación con las incidencias realizadas. El CCC tiene que decidir qué hacer con cada uno de los elementos involucrados con los cambios. En muchos casos, la modificación de un ECS implica la modificación de otros ECS asociados. Por tal motivo se recomienda crear un grafo de ECS donde queden reflejadas las relaciones entre cada elemento y de esta forma los especialistas tendrán una mayor visión de la magnitud del cambio a realizar. Así mismo, este brinda una mejor idea del tamaño del problema y se podrán establecer mejores estimaciones relacionadas con los plazos de cumplimiento.

Como parte del CC se recomienda realizar una identificación de cada uno de los componentes que integran las versiones liberadas del producto. Quedaría entonces en un documento la relación del producto liberado con todos los elementos que lo conforman y de cada elemento estaría igualmente registrada su actual versión. Así permanecería reflejada la evolución de los productos existentes dentro del grupo de desarrollo.

Emisión de reportes

La generación de informes de estado de la configuración (IEC) desempeña un papel vital en el éxito de los proyectos de desarrollo de software. Dentro de las organizaciones se encuentran involucradas muchas personas, por tal motivo, es muy fácil que se dé el síndrome "de la mano izquierda". Puede que dos programadores intenten modificar el mismo ECS con intereses distintos. Un equipo de desarrollo de software pudiera gastar meses de esfuerzo construyendo un software a partir de una especificación obsoleta. Pudiera ser que la persona que identifique grandes efectos secundarios relacionados con un cambio propuesto, no esté al tanto de que el cambio se está realizando. Estos y más ejemplos pueden tener lugar dentro del desarrollo de software, sin embargo, el IEC ayuda a eliminar estos problemas mejorando la comunicación de las personas involucradas.

En aras de evitar los problemas mencionados anteriormente en el proyecto se propone identificar un conjunto de reportes necesarios para mantener actualizado al equipo

de trabajo. Los reportes serán emitidos ante diferentes eventos sucedidos, que pueden ser: cambios en la arquitectura del proyecto, modificaciones en los requisitos del negocio, acuerdos relacionados con la organización y distribución de recursos humanos, entre otros.

Los reportes emitidos deben contener un formato adecuado, así como la información debe ser clara y concisa. Los reportes pueden ser publicados de manera general en un lugar dedicado para este tipo de información o pudieran ser enviados por correo. Es importante también recordar que todos los reportes no son para todas las personas. Por tal motivo debe haber una diferenciación entre las personas para no sobrecargarlos con mensajes que no sean de su interés.

Auditorías

La auditoría es una actividad que no se presenta en todas las definiciones de GCS. En ocasiones esta actividad se encuentra dentro del aseguramiento general de la calidad, asociada a los diferentes productos o procesos. De manera general la auditoría es una evaluación planificada e independiente de uno o más productos o procesos para determinar complacencia con un grupo de acuerdos. En la mayoría de los casos, los acuerdos se encuentran generalmente relacionados con los requerimientos. La auditoría, de manera general, proporciona garantía y constituye una actividad de consulta diseñada para adicionar valor y mejora las operaciones dentro de una organización.

Dentro de las actividades que deben realizarse para llevar a cabo el trabajo de auditoría están:

1. Definiciones de itinerarios y procesos de auditorías.
2. Identificación de quién realizará la auditoría.
3. Establecimiento de auditorías en las líneas bases establecidas.
4. Generar reportes relacionada con los procesos de auditorías realizados.

En los procesos de auditorías realizados, son muchos los elementos que se pudiesen estar chequeando. La selección de los elementos va en dependencia del grado de rigurosidad que desee aplicar el líder del proyecto dentro de sus revisiones. Entre las actividades que se deben realizar dentro del proceso de auditoría están: Comparar las especificaciones con el producto y registrar las discrepancias y anomalías, marcar las salidas de los ciclos de pruebas y registrar las discrepancias y anomalías, realizar recomendaciones sobre la base de las anomalías y discrepancias encontradas, presentar informes de las revisiones, entre otras.

CONCLUSIONES

Cumplimentar el desarrollo de un proyecto que utiliza la filosofía MDA resulta muy trabajoso y difícil. Por tal motivo es necesario implementar la disciplina de Gestión de Configuración de Software en aras de aumentar la calidad del producto. El trabajo identifica cuáles son los elementos de configuración de software que no deben dejar de tenerse en cuenta dentro de este tipo de proyectos y se realiza un análisis de los diferentes tipos de controladores de versiones existentes en la industria de software y se establece la necesidad de crear roles y usuarios para acceder a los repositorios del controlador de versiones. Queda resaltada la necesidad de implementar en los grupos de desarrollo un mecanismo de control de cambios que muestre la trazabilidad de las actividades en el proyecto. Se detallan además aspectos a tener en cuenta para realizar un correcto control y seguimiento de los cambios realizados dentro del grupo de trabajo. Por último, se identifica la retroalimentación que existe entre la auditoría y el proceso de cambios, y se muestra cómo no se pueden controlar los cambios si primero no se encuentran identificados. El trabajo demuestra que los cambios son inevitables, sin embargo, se puede estar preparado para ellos.

REFERENCIAS

1. **HOOVER, Carol.** *Evaluating Project Decisions*. Addison-Wesley. 2010, p. 398. ISBN 978-0-321-54456-8.
2. **PRESSMAN, Roger.** *Ingeniería de software. Un enfoque práctico*. McGraw-Hil. 2005, pp. 745. ISBN 9701054733.
3. **AHERN, Dennis.** *Cmmi® Distilled: a Practical Introduction to Integrated Process Improvement*, Third edition. Addison Wesley. 2008, p. 288. ISBN 0-321-46108-8.
4. **AIELLO, Bob.** *Configuration Management best Practices Practical Methods that Work in the Real World*. Addison-Wesley. 2011, p. 268. ISBN 978-0-321-68586-5.
5. **BELLAGIO, David.** *Software Configuration Management Strategies and ibm Rational Clearcase Second Edition a Practical Introduction*. Addison Wesley Professional. 2005, p. 384. ISBN 0-321-20019-5.
6. **NANZ, Sebastian.** *The Future of Software Engineering*. Springer. 2011, p. 194. ISBN 978-3-642-15186-6.
7. **KULPA, Margaret.** *Interpreting the cmmi a Process Improvement Approach*. Taylor & Francis Group. 2008, p. 424. ISBN 978-1-4200-6052-2.
8. **PAGÉS CHACÓN, Daniel.** "Incremento de la productividad en el desarrollo de la capa cliente de aplicaciones j2ee haciendo uso de mda". Tesis de Maestría, Tutor: Julio Pablo Martínez Prieto. Instituto Superior Politécnico José Antonio Echeverría, Cujae, La Habana. 2008.
9. **MCMAHON, Paul.** *Integrating cmmi® and Agile Development*. Addison Wesley. 2011, p. 358. ISBN 978-0-321-71410-7.
10. **JENDROCK, Eric.** *The Java ee 6 Tutorial Basic Concepts Fourth Edition*. Addison Wesley. 2011, p. 588. ISBN 978-0-321-71410-7.

11. SWICEGOOD, Travis. *Pragmatic Guide to Git*. Pragmatic Programmers. 2010, p. 149. ISBN 1-934356-72-7.

12. CHACON, Scott. *Pro git. Apress*. 2009, p. 290. ISBN 978-1-4302-1833-3.

13. KEYES, Jessica. *Software Configuration Management*. Auerbach Publications. 2004, p. 619. ISBN 0849319765.

AUTOR

Manuel Morejón Espinosa
Ingeniero Informático, Instructor, Grupo de Investigación y Desarrollo del Sistema de Gestión de la Nueva Universidad (SIGENO), Departamento de Ingeniería de Software, Facultad de Ingeniería Informática, Instituto Superior Politécnico José Antonio Echeverría, Cujae, La Habana, Cuba

Good Practices for Software Configuration Management with MDA Metodology

Abstract

Software Configuration Management (SCM) forms part of the software development process. Its principal goal is to coordinate this development and minimize all possible errors. In order to meet its goal various activities are carried out, of which can be identified: items identification, change control, version control, audit and status reporting. Inside enterprise applications the software development can be guided from system model as methodology. The name of this methodology is Model Driven Architecture (MDA). The use of this methodology has the objectives of increasing development´s velocity through the automatic generation code as well as keeping updated all the documents belonging to the system in relation to the existing code. The ways to carry out the mentioned activities in the SCM process differ between projects that use MDA methodology and other projects that do not use MDA methodology. Some aspects must be taken into account during the implementation of each activity described in the SCM process. Following this, the author mentions some ideas to take into account in each aspect of the process to ensure giving a guide on how to proceed in front of these kinds of project that use MDA methodology.

Keywords: software configuration management, model driven architecture, changes