Universidad de las Ciencias Informáticas

Facultad 1

"Modelo de desarrollo de componentes para la distribución cubana de GNU/Linux Nova"

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

Autor: Yileni Hechavarría González
Tutor: Mtr. Yusleydi Fernández del Monte
Ing. Mairim Delgado Muñiz
Consultante: Ing. Alexis López Zubieta

Habana, junio de 2015

Declaración de Autoría

Declaro ser la autora del presente Trabajo de Diploma y se reconoce a la Universidad de las Ciencias Informáticas, los derechos patrimoniales del mismo con carácter exclusivo. Para que así conste firmo la presente declaración jurada de autoría en La Habana a los días _____ del mes de _____ del año _____.

Yileni Hechavarría González

_____

Firma de la autora

Mtr. Yusleydi Fernández del Monte

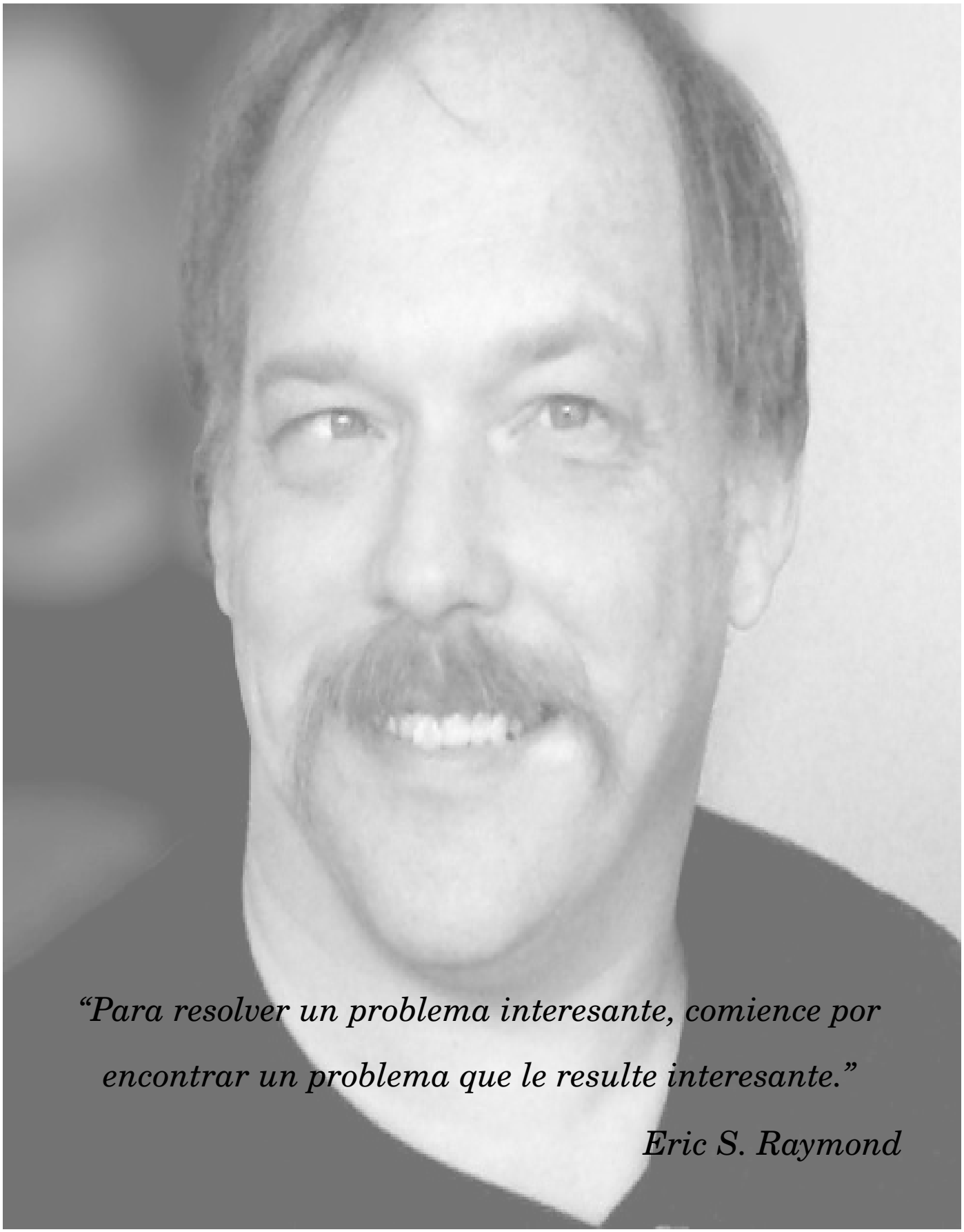
_____

Firma de la Tutora

Ing. Mairim Delgado Muñiz

_____

Firma de la Tutora



“Para resolver un problema interesante, comience por encontrar un problema que le resulte interesante.”

Eric S. Raymond

Dedicatoria

A mi abuelo por guiarme con su ejemplo y darme todo su amor y cariño.

A mi mamá por apoyarme en todas mis decisiones y demostrarme lo incondicional que puede ser el amor de una madre.

A mi hermano por brindarme su alegría.

A todas aquellas personas que me apoyaron y sirvieron de guía para que fuera una mejor persona.

Agradecimientos

A mi mamá y mi abuelo por su sacrificio y amor.

A mi hermano por su apoyo.

A Yudelín, Juan Carlos y Eva por demostrarme lo valiosa que puede ser la amistad.

A Yoel por acompañarme a lo largo de la carrera.

A mi antiguo grupo, que nunca dejaron de ser mis compañeros y amigos.

A mi compañera de cuarto Ariagna por brindarme su compañía y estresarme de vez en cuando.

A mi grupo por acogerme y demostrarme que siempre se pueden hacer nuevos amigos.

A Manuel, Neyvis, Yisel por las horas del café que tanto nos ayudaron a despejar del estrés de la tesis.

A los chicos del proyecto, a mis tutoras y consultante por ayudarme y aconsejarme con la tesis.

A todas aquellas personas que de una forma u otra han compartido conmigo y me han aconsejado y ayudado.

Resumen

El proceso de desarrollo de componentes constituye uno de los fundamentos de la Ingeniería de software basada en componentes. Es el proceso de definir, implementar e integrar componentes independientes débilmente acoplados. La metodología Nova – OpenUp es la encargada de guiar el proceso de desarrollo de la distribución cubana de GNU/Linux Nova. En la disciplina "Desarrollo" se plantea la posibilidad de desarrollar software propio pero no define las fases, actividades, roles y artefactos para guiar el proceso dificultando la integración de los mismos con la distribución cubana. Para dar solución a los problemas identificados se plantea como objetivo de la investigación, proponer un modelo de desarrollo de componentes para la metodología Nova – OpenUp que permita la integración de desarrollos propios a la distribución cubana de GNU/Linux Nova. En la presente investigación se propone un modelo compuesto por fases, actividades, roles y artefactos, utilizándose un método experimental para la validación del mismo, el cual demuestra que la solución propuesta facilita el proceso de integración de componentes con la distribución cubana y aumenta el alcance de la metodología Nova – OpenUp.

Palabras clave: Componentes, Distribución cubana de GNU/Linux Nova, Integración, Metodología Nova – OpenUp.

Índice

Índice de Figuras

Índice de tablas

Introducción

A partir de la necesidad de llevar la tecnología a diversos sectores de la sociedad cubana y desarrollar la industria del software manteniendo los principios: soberanía tecnológica, seguridad, socio adaptabilidad y sostenibilidad [1]; en el año 2004 el Consejo de Ministros decide dar inicio a un proceso de migración a plataformas libres[1] y de código abierto[2]. Actualmente la Universidad de las Ciencias Informáticas (UCI) representa un eslabón principal en dicho proceso, formando parte del Grupo Nacional para la migración a plataformas de código abierto [2]. El Centro de Software Libre (CESOL) es uno de los centros productivos de la UCI, el cual tiene como misión guiar los procesos de migración a aplicaciones de código abierto y desarrollar la distribución cubana de GNU/Linux Nova lanzada en el 2009 [3].

Desde sus inicios la distribución cubana ha realizado una serie de desarrollos propios enfocados en su mayoría a aportar mejoras o a complementar servicios existentes. Una variada gama de productos y soluciones fueron creciendo junto a la distribución cubana de GNU/Linux Nova. En el momento en que fueron realizados dichos productos el proceso de integración a la distribución no estaba descrito. Los componentes de software después de creados eran empaquetados y subidos por los propios desarrolladores al repositorio; un espacio colaborativo donde se encuentran herramientas y aplicaciones de manera organizada.

En las distribuciones GNU/Linux el proceso de integración de componentes se realiza de una manera similar al que se hace en la distribución cubana, un ejemplo de ello es el desarrollo de módulos para el *kernel* de Linux. Cuando se actualiza el *kernel* los desarrolladores lo hacen a través de unidades individuales llamadas parches[3]. Los parches desarrollados son enviados a otros desarrolladores de la comunidad, que comparten opiniones hasta que se encuentran listos para ser liberados. A partir de ese momento los parches son aceptados por un desarrollador o mantenedor de Linux, el cual se encarga de

1 Plataformas libres: conjunto de tecnologías, aplicaciones basadas en la terminología de software libre y código abierto.

2 Código abierto: expresión con la que se conoce al software distribuido y desarrollado libremente.

3 Parche: consta de cambios que se aplican a un programa para corregir errores, agregarle funcionalidad, actualizarlo, etc.

realizar una extensa revisión para enviársela al líder de la distribución de GNU/Linux. Solo esta persona tiene el privilegio de publicar de forma oficial el o los módulos desarrollados.

Entre los desarrollos realizados en la distribución cubana de GNU/Linux Nova se encuentran: Entorno de Escritorio (EE) Guano [4], basado en una arquitectura modular. Guano no fue pensado como un EE predeterminado sino como opción para aquellos usuarios que deseaban un escritorio ligero, sencillo y amigable. A su vez Guano constituyó una forma de enfrentar el problema de la obsolescencia de las estaciones de trabajo en el entorno cubano. Un instalador que aplicaba técnicas de Integración Continua y Desarrollo Dirigido por Pruebas (TDD[4]) fue desarrollado bajo el nombre de SERERE, permitía en su proceso de desarrollo la colaboración de la comunidad. Casi al mismo tiempo se desarrolla el Panel o Centro de Control de Nova para mejorar la gestión de la administración, incluía herramientas administrativas y de configuración. Se desarrolló Ecumenix y Capoeira, el primero de ellos orientado tanto a administradores de sistemas como a usuarios finales de escritorio; su función principal era unir una computadora personal con el sistema operativo GNU/Linux a un dominio controlado por un directorio activo de *Microsoft*. Capoeira por su parte permitía compartir archivos a través de Samba[5].

Actualmente el proceso de desarrollo de la distribución cubana lo guía la metodología Nova - OpenUp [5], que surge a partir de la necesidad de erradicar los problemas detectados en las auditorías realizadas por el proyecto Nova QALIT[6] en el período 2010 - 2013. En su mayoría estas deficiencias están relacionadas con las áreas de gestión de la configuración de software, administración de requisitos, aseguramiento de la calidad, la planificación de proyectos y las desviaciones en el proceso de desarrollo [6].

Nova - OpenUp abarca todo el ciclo de vida de la distribución nacional, desde la obtención de los requisitos hasta la liberación del sistema operativo final y las pruebas al mismo. Contenida en la disciplina

4 TDD: por sus siglas en inglés *Test-driven development.*

5 Samba: implementación libre del protocolo de archivos compartidos de *Microsoft Windows*.

6 Nova QALIT: proyecto que pertenece al departamento Sistemas Operativos que tiene como meta principal, garantizar la calidad de los procesos y productos que tienen lugar en dicho departamento.

desarrollo de la metodología, se encuentra la actividad "Desarrollar software propio" que describe el ciclo de vida para los desarrollos realizados por la comunidad o el equipo de desarrollo, para finalmente ser incorporados a la distribución. Aún cuando la metodología describe esta parte del ciclo de vida, en la actualidad presenta la limitante de no detallar el proceso de integración de los desarrollos propios a la distribución. La no determinación de fases, actividades, roles y artefactos en el proceso de desarrollo de nuevos componentes propicia la duplicidad de esfuerzos e incompatibilidad entre componentes. En consecuencia se dificulta la planificación y la reutilización de los mismos.

Tomando como punto de partida la problemática anteriormente descrita se plantea como **problema de la investigación**: ¿Cómo integrar a la metodología Nova - OpenUp el proceso de desarrollo de componentes para la distribución cubana de GNU/Linux Nova? Definiendo como **objeto de estudio** el proceso de desarrollo de software basado en componentes. El **objetivo general** que se persigue es: proponer un modelo de desarrollo de componentes para la metodología Nova - OpenUp que permita la integración de desarrollos propios a la distribución cubana de GNU/Linux Nova. Para darle cumplimiento al objetivo general propuesto se definen los siguientes **objetivos específicos:**

- Sistematizar el estudio de modelos de desarrollo de componentes.
- Diseñar un modelo para el proceso de desarrollo de componentes que permita la integración de los mismos con la distribución cubana de GNU/Linux Nova.
- Evaluar el modelo desarrollado de manera que se compruebe que permite la integración de componentes con la distribución cubana de GNU/Linux Nova.

Se establece como **campo de acción** el proceso de desarrollo de componentes para la distribución cubana GNU/Linux Nova. Para dar cumplimiento al objetivo general se identificaron las siguientes **tareas de la investigación**:

- Análisis de bibliografías asociadas a la integración de componentes en distribuciones de GNU/Linux para conocer cómo se realiza el proceso de integración de componentes.

- Análisis de las tendencias en el proceso de desarrollo de la distribución cubana de GNU/Linux Nova para la integración de componentes.
- Diseño de un modelo de desarrollo de componentes a partir de los resultados obtenidos en la revisión bibliográfica.
- Evaluación a través de casos de estudio para obtener una simulación de las posibles respuestas a la aplicación del modelo.

La presente investigación plantea como **idea a defender** que el diseño de un modelo de desarrollo de componentes que se asocie a la metodología Nova - OpenUp debe permitir la integración de los desarrollos propios realizados para la distribución cubana de GNU/Linux Nova, aumentando el alcance de la misma.

Métodos Utilizados

Para el desarrollo de la investigación se utilizaron métodos teóricos que son descritos a continuación:

- **Histórico-Lógico:** la utilización de este método permite conocer la evolución que ha tenido el proceso de desarrollo de componentes en la distribución cubana de GNU/Linux Nova así como las causas que dieron paso a su surgimiento, las características comunes que se han mantenido con el transcurso del tiempo y las nuevas que han surgido.
- **Analítico-Sintético:** se utiliza para dividir el problema en subproblemas que faciliten el estudio del proceso de desarrollo de la distribución cubana de GNU/Linux Nova y luego sintetizarlo en una solución general acorde al problema propuesto.
- **Análisis bibliográfico:** empleado para la revisión de la literatura necesaria durante el proceso de investigación sobre los procesos de desarrollo de componentes y los aspectos fundamentales de la metodología Nova – OpenUp.
- **Método experimental:** utilizado para verificar el cumplimiento del modelo de desarrollo propuesto a partir de la técnica de casos de estudio. La aplicación del método permite realizar una

comparación entre el proceso que se estaba empleando y el que se propone para observar los cambios que ocurren.

- **Entrevista:** utilizada para conocer cómo se realiza el proceso de desarrollo e integración de componentes en la distribución cubana de GNU/Linux Nova.
- **Modelación**: empleado para representar gráficamente el funcionamiento del proceso de desarrollo de componentes.

Para una mejor comprensión del contenido la investigación se estructuró en tres capítulos, conclusiones y bibliografía utilizada. Los capítulos se organizan de la siguiente forma:

Capítulo 1: Modelos de desarrollo basados en componentes. En este capítulo se realiza un estudio de diferentes bibliografías asociadas a los modelos de componentes existentes, sus características, herramientas utilizadas y formas de desarrollo. También se realiza un análisis de los desarrollos propios realizados para la distribución cubana GNU/Linux Nova y el ciclo de vida que propone la metodología Nova – OpenUp para los desarrollos propios.

Capítulo 2: **Propuesta del modelo de desarrollo de componentes para la distribución cubana de GNU/Linux Nova.** Este capítulo muestra la solución propuesta describiendo las fases, actividades, roles y artefactos del ciclo de vida del modelo de desarrollo de componentes así como sus características, alcance y principios.

Capítulo 3: Evaluación de la propuesta de solución. Se evalúa el modelo propuesto a través del método experimental para obtener posibles resultados de la aplicación del modelo de desarrollo de componentes.

Capítulo 1: Modelos de desarrollo basados en componentes

Introducción

El presente capítulo brinda una descripción general de los modelos de desarrollo basados en componentes abordando definiciones y conceptos fundamentales. Se realiza además un estudio crítico de la metodología Nova – OpenUp enfocado al desarrollo de software propio así como la realización de los mismos para la distribución cubana de GNU/Linux Nova. Se especifican algunas definiciones de interés para lograr una mejor comprensión del tema.

1.1 Definiciones de interés

1.1.1 Componentes

Según el diccionario de la Real Academia Española es un elemento *"que compone o entra en la composición de un todo"*. En el ámbito de la informática los componentes del sistema son descompuestos según Pressman[7] en cuatro tipos fundamentales: hardware, software, datos y personas. La presente investigación acota su estudio a los componentes de software, los cuales Pressman define como: "*un elemento de software que se puede tratar por separado, tales como subrutinas, funciones o procedimientos* [7]." Disímiles autores varían su concepto para dichos componentes en dependencia del propósito para el que sean utilizados.

- Clemens Szyperski[8] lo define como una unidad de composición con interfaces especificadas contractualmente y solamente dependencias explícitas de contexto. Puede ser desplegado independientemente y está sujeto a composición por terceros [8].

7 Roger S. Pressman: ingeniero de software norteamericano convertido en una reconocida autoridad internacional en las mejoras del proceso del software y en las tecnologías de la Ingeniería de software.

8 Clemens Alden Szyperski: director gerente de grupo de Ingeniería de software de *Microsoft*.

- Somerville[9] plantea que un componente es una unidad de software independiente que puede estar compuesto por otros componentes y que se utiliza para crear un sistema software [9].
- El instituto de ingeniería de software (SEI) propone un concepto de componentes con la idea de consolidar la diversidad de opiniones. *"Una implementación opaca de funcionalidad, sujeta a composición por terceros y que cumple con un modelo de componente* [10]."

Los autores antes mencionados coinciden en que un componente es una unidad de composición que puede ser independiente y compuesta por terceros. Pero los conceptos analizados no son los más apropiados para la presente investigación, aunque tengan puntos de contactos con la misma. Los componentes a tratar serán vistos como paquetes de software, los cuales poseen características similares a los antes mencionados pero también difieren de los mismos.

- El diccionario de informática y tecnología explica que un paquete de software es un grupo de uno o más archivos que son necesarios tanto para la ejecución de un programa de computadora, o como para agregar características a un programa ya instalado en una computadora o en una red de computadoras. Puede hacer referencia a cualquier componente que pueda ser integrado en el programa principal [11].
- Conjunto de aplicaciones que se agrupan para formar un juego completo. Cada uno incluye varios programas e información sobre la versión, instalación y uso [12].
- Conjunto de ficheros relacionados con una aplicación que contiene diferentes objetos [13].

Los paquetes de software al igual que los componentes pueden ser una parte o el todo de una aplicación. Ambos pueden ser desarrollados por terceros y de forma independiente, por lo que el concepto que más se ajusta a la investigación es el planteado por Councell[10] y Heineman[11]: *"un elemento software que se*

9 Ian Sommerville: académico británico, destacado investigador en el campo de la fiabilidad de sistemas y la informática social.

10 Bill Councillis fue gerente de sistemas y procesos de software de *Mannatech*.

11 George T. Heineman: profesor asistente de Ciencias de la Computación en el Instituto Politécnico de *Worcester*.

ajusta a un modelo de componente y que puede ser desplegado y compuesto de forma independiente sin modificación según un estándar de composición [14].*"*

Características de los componentes

Los componentes de software poseen diversas características que a su vez funcionan como principios, debido a que su cumplimiento es fundamental para garantizar que un componente se pueda reutilizar y de esta manera integrarlo con algún sistema que los necesite. Sommerville define algunas de ellas [9]:

- Estandarizado: tiene que ajustarse a algún modelo estandarizado de componentes. Este modelo puede definir interfaces, metadatos, documentación, composición y despliegue.
- Desplegable: debe ser capaz de funcionar como una entidad autónoma o sobre una plataforma de componentes que implemente el modelo de componentes.
- Documentado: tiene que estar documentado para que los usuarios puedan decidir si satisfacen o no sus necesidades.

A estas características es necesario agregarle algunas de las propuestas por Matilda, Arapé y Colmenares en el artículo "Desarrollo de Software Basado en Componentes" [15] para abarcar todas las necesidades de la investigación:

- Mantenido: para que esté inmerso en un proceso de mejoramiento continuo para la obtención de nuevas versiones.
- Identificable: debe tener una identificación clara y consistente que facilite su catalogación y búsqueda en repositorios de componentes.
- Autocontenido: para que dependa lo menos posible de otros componentes para cumplir su función de forma tal que pueda ser desarrollado, probado, optimizado, utilizado, entendido y modificado individualmente.
- Accesible sólo a través de su interfaz: debe exponer al público únicamente el conjunto de operaciones que lo caracteriza (interfaz) y ocultar sus detalles de implementación para permitir que

sea reemplazado por otro que implemente la misma interfaz.

1.1.2 Integración

La integración representa el proceso desarrollado cuando las partes son combinadas para formar partes más complejas y finalmente en productos completos. Los elementos críticos en la integración incluyen descripción y gestión de interfaces, la secuencia en la cual los componentes son integrados y la comunicación entre diferentes involucrados. Incluso se reconoce que también se incluyen requerimientos y propiedades del sistema que no pueden ser comprobados desde un nivel de componente, pero que deben ser comprobados a nivel de sistema [16].

Otros autores lo expresan como el acto de combinar componentes de software (programas y archivos) en un sistema de software. En proyectos grandes se puede hablar de múltiples componentes que pueden llegar a ser sólo archivos de código de bajo nivel compilados para pequeños proyectos [17]. Ambas definiciones coinciden en que la integración es el resultado de unir partes hasta conformar un todo. Pero la más adecuada para la investigación es la primera, que trata la integración como un proceso.

1.2 Modelos de desarrollo de software

Un modelo de desarrollo es una representación abstracta de un proceso de software. Establece el orden en que se realizaran las actividades en el proyecto. Provee los requisitos de entrada y salida de cada una de las actividades que se deben realizar. Puede ser modificado y adaptado de acuerdo a las necesidades del software en desarrollo [9].

1.2.1 Modelo de desarrollo basado en componentes

La Ingeniería del software basada en componentes (CBSE[12]) surge a finales de los 90, como una aproximación a la reutilización de componentes llegándose a convertir en una disciplina de la Ingeniería de software. Constituye el proceso de definir, implementar e integrar componentes independientes débilmente

12 CBSE: por su significado en inglés *Componet-Based Software Engineering*.

acoplados. Dentro sus fundamentos se encuentran los estándares y el proceso de desarrollo de componentes (PDC), el primero facilita la integración y son incluidos en un modelo de componentes, el último implica la adaptación a CBSE [9].

Modelos de componentes

Los modelos de componentes especifican las reglas del diseño que se deben obedecer y la especificación de un marco regulatorio estandarizado para lograr la integración entre los componentes. Constituye una definición de estándares para la descripción de componentes, documentación y despliegue, las interfaces y el lenguaje a utilizar [9]. Definen los componentes que se van a emplear, cómo deben ser y cómo se relacionan entre ellos y con el sistema [18]. Resultan ser evolutivos porque van creando versiones hasta conseguir el producto final y están diseñados para ajustarse al cambio, por lo que exige un enfoque iterativo. Para la creación de un modelo de componentes es necesario:

- Especificar cómo deben configurarse los componentes binarios para un entorno de despliegue particular.
- Definir cómo debe ser el proceso de empaquetado.
- Información de despliegue para incluir información sobre los contenidos de un paquete y su organización binaria.
- Incluir reglas para regular cuándo y cómo se permite el reemplazo de un componente.
- Definir la documentación asociada a los componentes.

En la figura 1 se muestra el modelo del proceso genérico para los CBSE propuesto por Sommerville. En dicho modelo las etapas referentes a especificación de requerimientos y validación del sistema coinciden con los modelos tradicionales pero varían las etapas intermedias [9]:

- Análisis de componentes: se realiza la búsqueda de componentes para dar solución al problema o a una parte.
- Modificación de requerimientos: se analizan los componentes obtenidos en la etapa anterior y de

ser posible se modifican los requisitos basándose en los componentes encontrados.

- Diseño del sistema con reutilización: se reutiliza el marco de trabajo a partir de los componentes encontrados y de no encontrar ninguno se diseña uno nuevo.
- Desarrollo e integración: se crea el software y se integra formando este último una parte del proceso de desarrollo y no una actividad separada.

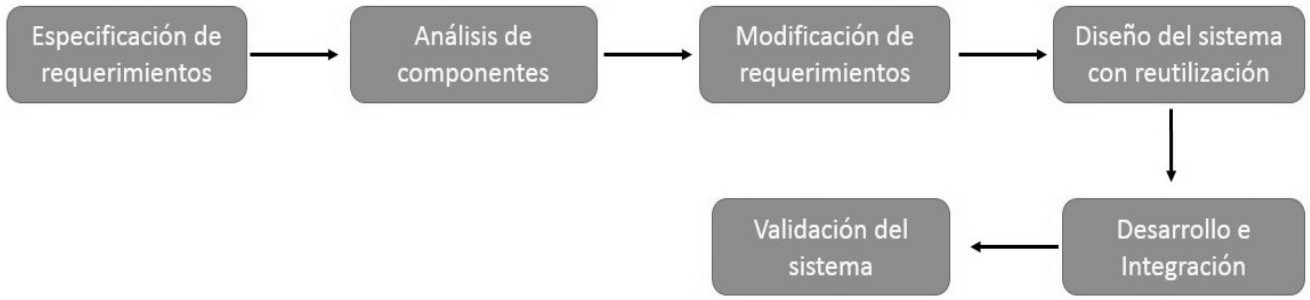


Figura 1: Ingeniería de desarrollo basado en componentes.

Fuente: SOMMERVILLE, Ian. Software engineering. Addison-Wesley Pub Co, 6ta edición, Agosto 2000.

Pressman plantea en su libro Ingeniería del Software que la CBSE es bastante similar a la Ingeniería del software orientado a objetos tradicional. El proceso comienza con el establecimiento de requisitos que se realiza mediante las técnicas tradicionales (inicio, obtención, elaboración, negociación, especificación, validación y gestión). Se establece un diseño arquitectónico pero no guiado al diseño sino a examinar los requisitos para determinar cuáles pueden ser para la composición y no para la construcción. En caso de existir componentes que satisfagan las necesidades se realiza un conjunto de actividades (cualificación, adaptación, composición y actualización), sino se implementan siguiendo las reglas de la ingeniería de software tradicional [19]. Ver figura 2.

El estudio de CBSE es de gran importancia, es preciso conocer el PDC para lograr una correcta integración entre los componentes desarrollados y la distribución cubana logrando así interiorizar las características comunes de ambos procesos. Teniendo en cuenta los criterios de Pressman y Sommerville

explicados anteriormente se puede llegar a la conclusión de que el PDC debe poseer características relacionadas con el análisis de requerimientos, análisis de componentes, desarrollo e integración de componentes y la realización de pruebas de integración.

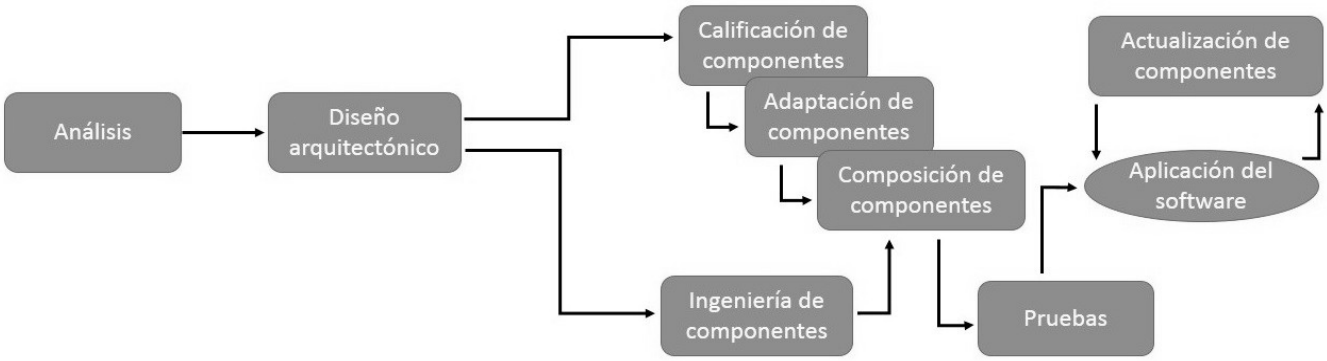


Figura 2: Modelo de proceso para CBSE.

Fuente: PRESSMAN, Roger S. Métodos convencionales para la ingeniería del software.

Proceso de desarrollo de componentes

El PDC es una actividad de la CBSE paralela a la ingeniería de dominio (identificar, construir, catalogar y diseminar componentes). Se centra en la realización de componentes reutilizables con el objetivo de producir repositorios activos que puedan ser utilizados en el desarrollo de software. El PDC posee dos trayectorias paralelas: cuando existen componentes reutilizables disponibles, que solo tienen que cualificarse y adaptarse, y cuando se requieren nuevos componentes. Cuando existe algún componente éste debe cualificarse para garantizar que el componente se acople con el estilo arquitectónico del sistema. Para la realización de nuevos componentes se recomienda utilizar métodos de diseño convencionales, pero siempre teniendo en cuenta que deben ser componentes reutilizables [19].

Ventajas del PDC

- Menor tiempo de desarrollo: porque al reutilizar componentes no es necesario desarrollar de cero sino utilizar componentes ya existentes, lo que implica un consumo menor de tiempo en el

desarrollo general de la aplicación.

- Mayor productividad: permite a una empresa asumir más proyectos, ya que no están gastando tiempo en desarrollar un nuevo software debido a que están reutilizando las clases que ya se han desarrollado. Esto da a los ingenieros más tiempo para trabajar en otros proyectos.
- Mayor calidad: debido a que se utilizan componentes que ya han sido probados hay menos problemas por resolver.
- Coste: posee una reducción en el tiempo de desarrollo, una mayor productividad y una mayor calidad lo que produce un aumento de los ingresos [20].

Desventajas del PDC

- Disponibilidad de componentes: no siempre se cuenta con la cantidad suficiente de componentes reutilizables.
- Falta de estandarización: en muchas ocasiones no se cuenta con un modelo capaz de estandarizar el proceso de integración de los componentes.
- Falta de procesos de certificación que garanticen la calidad de los componentes.

1.2.2 Modelo evolutivo

El modelo evolutivo se basa en la realización de versiones de un producto hasta llegar al resultado final. Cada versión se expone y se refina de acuerdo a los comentarios o a los nuevos requisitos que van surgiendo. Las fases de especificación, desarrollo y validación se entrelazan en vez de separarse. Una de las tendencias del modelo evolutivo es el modelo incremental. En dicho modelo a medida que se presenta un prototipo se produce un incremento, que se considera una especie de iteración. El primer incremento a menudo es un producto esencial que afronta requisitos básicos. Los primeros incrementos son versiones incompletas del producto final pero proporcionan al usuario la funcionalidad que precisa y también una plataforma para la evaluación.

1.2.3 Fábricas de software

El término Fábricas de software (FSW) fue introducido por primera vez en el año 1968 por Bemer[13] quien planteaba: *"una fábrica, sin embargo, tiene más que supervisión humana. Mide y controla la productividad y calidad. Se mantienen registros financieros para coste y planificación"*. En el año 1993 sale a relucir la expresión FSW basado en la reutilización, cuando Griss[14] señala que una reutilización efectiva requería más que tecnología para bibliotecas y código y que utilizar sólo la metáfora de la biblioteca limitaba los resultados de la reutilización, la solución pasaba por familias de soluciones relacionadas [21].

De manera general las FSW constituyen un modelo dinámico enraizado en un ciclo de vida, que discurre siguiendo una metodología capaz de proporcionar los servicios que el cliente demanda sobre las plataformas tecnológicas existentes en cada caso y asegurando altos niveles de calidad. El enfoque de las FSW propone el desarrollo de métodos específicos para cada tipo de aplicación. Utilizando estos métodos, los desarrolladores especifican cada sistema utilizando un lenguaje de modelado. Las FSW se rigen por principios, los cuales complementan: la gestión de requisitos, las pruebas, la gestión de la configuración y los aspectos organizativos, que resultan clave para el éxito de la fabricación de software. Además se centra en el desarrollo de sistemas similares promoviendo la reutilización de arquitecturas y componentes de software [22].

Para un mayor entendimiento de cómo funcionan las FSW se decide estudiar el modelo "Fábricas de software basada en componentes" propuesto por el departamento de Señales Digitales del centro de desarrollo de software "Geoinformática y Señales Digitales" en la UCI. El modelo permite la reutilización de código, la utilización de componentes existentes y la posibilidad de integrar diferentes proyectos con características semejantes para alcanzar un desarrollo de software ágil y eficiente a la medida del cliente, está estructurado en entidades o componentes organizacionales [23]:

---

13 Robert William Bemer: informático conocido por sus trabajos en IBM, se le conoce como el padre del ASCII.

14 Martin Griss: científico principal de investigación de la universidad Carnegie Mellon, Silicon Valley. Con experiencia en desarrollo de software, la educación y la investigación.

- La producción basada en componentes donde exista un área de producción de software que incluya y fomente la producción de componentes.
- La entidad repositorio de componentes que permita la gestión óptima de los componentes de código y los activos del proceso.
- El uso de estándares de calidad que eleven el proceso de desarrollo de software y la producción de componentes.
- La definición de reglas que permitan la coordinación de cada una de las personas que intervienen en el proceso y el ensamblaje de cada uno de los componentes.
- La definición de la entidad proceso y los elementos indispensables en un proceso.
- La entidad persona deberá incluir la generalización y especialización de equipos y personas.
- La creación de bases tecnológicas que se enfoquen en mantener una ardua y efectiva vigilancia tecnológica que provee información exacta para cada producto.

La utilización de las FSW en el proceso de desarrollo de la distribución garantiza un mejor aprovechamiento de los recursos humanos disponibles en el departamento Sistemas Operativos, que actualmente no cuenta con los necesarios. Dicho problema provoca que no se logre una especialización del personal en un rol específico teniendo que llevar varios a la vez, lo cual conlleva atraso en la entrega y culminación del producto.

Consideraciones generales de los modelos de desarrollo

La descripción de los modelos realizada permite conocer las características fundamentales del modelo de desarrollo de componentes, las FSW y los modelos evolutivos, demostrando que en la actualidad se tiende a fusionarlos. El modelo que se aplica en la distribución cubana de GNU/Linux Nova es el modelo de desarrollo basado en componentes, ya que combina la técnica de realizar versiones de los modelos evolutivos con la de las FSW de reutilizar código y componentes.

1.3 Metodologías

En el presente epígrafe se realiza un análisis de varias metodologías para comprender como definen el proceso de integración. Las metodologías a analizar son: Proceso de Desarrollo Unificado (RUP[15]) considerada la metodología universal para el proceso de desarrollo de software, Programación Extrema (XP[16]) unos de los procesos ágiles de desarrollo de software más destacados según la guía de ingeniería del software establecida por el Laboratorio Nacional de Calidad del Software de INTECO, Proceso Unificado Ágil (AUP[17]), metodología que guía el proceso de desarrollo de software de la UCI y Nova – OpenUp debido a que es la metodología que guía el proceso de desarrollo de la distribución cubana de GNU/Linux Nova. Para poder estudiarlas es necesario conocer que significa el término metodología.

Según el diccionario de la Real Academia Española una metodología es un "*conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal*". Para acercarse más al tema de investigación se estudia el concepto planteado por el Laboratorio Nacional de Calidad del Software en la guía "Ingeniería del Software: Metodologías y Ciclos de Vida" que la define como "*un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo. Es un proceso de software detallado y completo. Las metodologías se basan en una combinación de los modelos de proceso genéricos. Definen artefactos, roles y actividades, junto con prácticas y técnicas recomendadas*" [24].

1.3.1 RUP

RUP es un modelo de software que permite el desarrollo de software a gran escala mediante un proceso continuo de pruebas y retroalimentación garantizando el cumplimiento de ciertos estándares de calidad. Permite enfocar el esfuerzo de los recursos humanos en términos de habilidades, competencias y

15 RUP: por su significado en inglés *Rational Unified Process.*

16 XP: por su significado en inglés *Extreme Programing.*

17 AUP: por su significado en inglés *Afile Up.*

capacidades a asumir roles específicos con responsabilidades bien definidas [25]. RUP divide su proceso de desarrollo en cuatro fases y nueve flujos de trabajo, dentro de la que se encuentra el flujo de trabajo implementación. En el mismo se definen un conjunto de actividades encaminadas a la creación e integración de los componentes de un sistema [26]:

- Estructurar el modelo de implementación: generalmente la confección del modelo de implementación se realiza a partir de un conjunto de subsistemas de implementación que pueden ser desarrollados relativamente independientes. Es una actividad regida por el arquitecto en la fase de elaboración, permite organizar el modelo de implementación para administrar su complejidad y se reconocen las funcionalidades principales que conforman el sistema.
- Planear la integración: se debe elaborar un plan que contenga la planificación de la integración, el cual estará enfocado en los subsistemas de implementación que se implementarán y el orden en que los subsistemas de implementación deben ser integrados en la iteración actual.
- Implementar componentes: se detallan los elementos que conformarán un componente y se desarrollan cumpliendo los requisitos del sistema.
- Integrar cada subsistema: se integran los componentes que ya han sido implementados y validados según la planificación y los mecanismos detallados en el plan de integración. Se describen los cambios ocurridos en la integración por varios programadores para crear una nueva versión consistente del componente.
- Integrar el sistema: el integrador guiándose por el plan de integración de construcciones va adicionando los componentes liberados para la integración dentro del área de trabajo de integración de sistemas. En cada construcción la integración es probada. Después de la última integración a la construcción se le pueden realizar pruebas en el sistema completo.

RUP define además de las actividades, los artefactos necesarios para realizar el proceso de integración, los cuales son responsabilidad del rol integrador. Los artefactos generados son: crear espacio de trabajo

de integración, planear integración de subsistemas, planear integración del sistema, crear líneas base, integrar subsistemas, integrar el sistema, promover las líneas bases. Además de los artefactos planea pruebas de integración para demostrar que el sistema funciona correctamente después de haber realizado la integración de los componentes.

Aún cuando RUP presta gran atención a la integración posee deficiencias que afectan la calidad del software y del desarrollo de software en general. Al realizar las pruebas de integración al finalizar todos los componentes se detectan un mayor número de errores introducidos mucho tiempo atrás, por lo que a la hora de erradicarlos resulta engorroso y costoso.

1.3.2 XP

XP constituye una metodología de desarrollo ágil, que pone más énfasis en la adaptabilidad que en la previsibilidad. 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. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes donde existe un alto riesgo técnico [27].

XP es el más destacado de los procesos ágiles de desarrollo de software. Centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores y propiciando un buen clima de trabajo [24]. Dentro de las prácticas definidas en XP se encuentra la Integración Continua. A continuación se profundiza debido a la importancia que representa para la investigación.

Integración Continua

La Integración Continua es un modelo informático propuesto por Martin Fowler[18] que lo define como: "*Práctica de desarrollo software donde los miembros del equipo integran su trabajo frecuentemente, al*

18 Martin Fowler: ingeniero de software británico, autor y orador internacional sobre desarrollo de software, especializado en análisis y diseño orientado a objetos, UML, patrones de diseño.

menos una vez al día. Cada integración se verifica con un build[19] automático (que incluye la ejecución de pruebas) para detectar errores de integración tan pronto como sea posible". Generalmente está asociada con metodologías de programación externa y desarrollo ágil. La Integración Continua se basa en un software especializado en automatizar tareas y hacer que estas se ejecuten de forma automática cuando se produce un evento. Sus características principales son [29]:

- Mantener un único repositorio de código fuente.
- Automatizar la construcción del proyecto.
- Hacer que la construcción del proyecto ejecute sus propios *tests*.
- Entregar los cambios a la línea principal todos los días.
- Mantener una ejecución rápida de la construcción del proyecto.
- Probar en una réplica del entorno de producción.
- Hacer que todo el mundo pueda obtener el último ejecutable de forma fácil.
- Publicar qué está pasando.
- Automatizar el despliegue.
- Construir la línea principal en la máquina de integración.

Las principales ventajas de la Integración Continua radican en que los desarrolladores pueden detectar y solucionar problemas de integración de forma continua. Disponibilidad constante de una versión para pruebas, ejecución inmediata de las pruebas unitarias y la monitorización continua de las métricas de calidad del proyecto. La Integración Continua conlleva a una mejora en la calidad de software, propiciando la detección de errores en el menor tiempo posible, en fases tempranas para poder solucionarlo rápidamente. Mejora la calidad del proceso dando visibilidad en el proceso de desarrollo en general, de manera que se conozcan las fases por las que va pasando el código y el estado del software en cada momento. Otro aspecto importante es la calidad de las personas, que mediante este modelo aumenta ya

19 Build: total de etapas necesarias para la compilación, creación de entregables al momento de ejecutar los *tests*.

que posibilita que los integrantes de los equipos de desarrollo aprendan a realizar distintos tipos de pruebas (unitarias, integración), mejores prácticas de programación y les brinda una mayor confianza [28].

1.3.3 AUP

AUP constituye la metodología de desarrollo de software propuesta para ser aplicable en todos los centros de desarrollo de software de la UCI. Su objetivo principal es unificar el proceso de desarrollo de software. La variación establecida para la UCI cuenta con tres fases (inicio, ejecución, cierre). Dentro de cada fase se encuentran ocho disciplinas (modelado del negocio, requisitos, análisis y diseño, implementación, pruebas internas, pruebas de liberación, pruebas de aceptación, gestión de la configuración, planeación de proyecto, monitoreo y control de proyecto). AUP describe el proceso de desarrollo de un producto, pero no define el proceso de integración de estos procesos con los desarrollos propios. Por dicho motivo no es aplicable para resolver el problema de investigación existente.

1.3.4 Nova – OpenUp

Nova - OpenUp, metodología creada con el objetivo de guiar el proceso de desarrollo de la distribución cubana de GNU/Linux Nova. Abarca todo el ciclo de vida de los productos realizados por la distribución mediante un conjunto de disciplinas compuestas por actividades que se relacionan entre sí, roles y artefactos. Combina las características de las metodologías pesadas con las de las metodologías ágiles. Nova – OpenUp es recomendada cuando se pretende: desarrollar una distribución GNU/Linux o una personalización de esta, obtener altas evaluaciones de calidad mediante el cumplimiento de exigencias para optar por el nivel dos de CMMI[20], trabajar con equipos de desarrollo tanto grandes como pequeños, en locales centralizados como dispersos geográficamente, prever y mitigar riesgos importantes a tiempo. La metodología está guiada por seis principios fundamentales:

- Colaborar para sincronizar intereses y compartir conocimientos.

20 CMMI: por su significado en inglés *Capability Maturity Model Integration*. Modelo para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software.

- Equilibrar las prioridades para maximizar el beneficio obtenido por los interesados en el proyecto.
- Centrarse en la construcción de principios para minimizar los riesgos y organizar el desarrollo.
- Desarrollar evolutivamente para obtener retroalimentación y mejoramiento continuo.
- Utilizar una infraestructura tecnológica adecuada que fomente el trabajo en entornos de software libre.
- Aplicar métodos para la obtención de calidad en procesos y productos.

Nova – OpenUp tiene en cuenta las prácticas definidas por OpenUp, las relacionadas con la administración de proyectos (desarrollo iterativo, ciclo de vida basado en el valor y el riesgo, planificación a dos niveles, equipo completo, gestión de cambio) y con las prácticas técnicas (pruebas concurrentes, integración continua, arquitectura completa, visión compartida, desarrollo basado en los requisitos de la distribución). Responde al modelo de desarrollo incremental y tiene iteraciones que presentan características del modelo concurrente, debido a que permite la ejecución de varias actividades a la vez. Comprende cuatro fases que guían el desarrollo de una distribución de GNU/Linux: inicio, elaboración, construcción y transición. Además define disciplinas compuestas por actividades, roles, artefactos y las relaciones entre ellos. Las mismas son: requisitos, arquitectura, desarrollo, prueba, dirección de proyecto y gestión de la calidad.

Entre las actividades de la disciplina “Desarrollo” se encuentra “Desarrollar software propio” de manera opcional. Un proceso de desarrollo de software propio consiste en el desarrollo de nuevas aplicaciones realizadas completamente desde cero que pueden ser incorporadas al repositorio de fuentes y binarios. La metodología plantea que los desarrollos propios pueden ser realizados a partir de un subproyecto/proyecto de desarrollo y su ciclo de vida y la alineación con la distribución varía en dependencia del producto. Describe de manera general como debe ser el proceso de desarrollo para el software propio, el mismo se muestra en la figura 3.

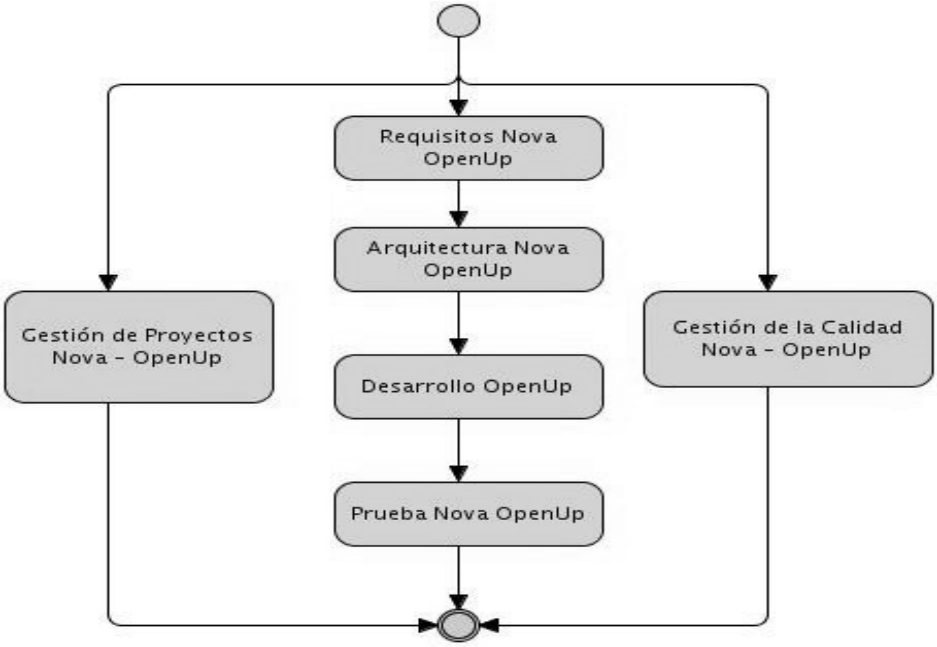


Figura 3: Ciclo de vida para desarrollos propios

Fuente: http://repositorio_institucional.uci.cu/jspui/handle/ident/8096.

Nova – OpenUp recomienda seguir el ciclo de vida que se propone en la figura 3, de manera que se genere un paquete de código fuente y binario para incorporarlo a la distribución. Para la realización de los desarrollos propios la metodología propone como artefactos de entrada: “Arquitectura_de_software” y “Descripción_de_requisito_ágil” y como salida el “código fuente”.

Aún cuando Nova – OpenUp define el ciclo de vida de los desarrollos propios no establece las actividades, roles y artefactos para llevar a cabo la integración de los componentes con la distribución. Sólo plantea de manera general el proceso que se debe realizar a la hora de desarrollar algún componente. Dicha característica constituye una debilidad en la metodología, debido a que la realización de componentes es muy variable, lo que provoca atrasos en las entregas, mayor esfuerzo de los trabajadores y que en ocasiones los componentes implementados no puedan ser reutilizados.

Consideraciones generales de las metodologías

El estudio de RUP, XP, AUP y Nova – OpenUp garantiza un entendimiento de cómo funciona el proceso de integración de componentes, así como el desarrollo de los mismos. Por lo que es necesario que el modelo a diseñar mantenga el ciclo de vida de desarrollo de software propio que propone Nova – OpenUp, pero con variantes enfocadas principalmente a detallar el PDC de manera general, detallando las actividades, roles y artefactos que intervienen en el mismo. De la metodología XP es necesario incorporar el proceso de integración continua con el objetivo de detectar los errores de forma temprana, para de esta manera poder erradicarlos. De RUP adquirir algunas características relacionados con las actividades, roles y artefactos que intervienen en el proceso de integración.

1.4 Desarrollo en comunidades de usuarios de software libre y código abierto

Debido a que la distribución cubana de GNU/Linux Nova se caracteriza por realizar un desarrollo en comunidad, es necesario estudiar las características del mismo, con el objetivo de incorporar este proceso en el diseño del modelo a proponer. El desarrollo en comunidad ha alcanzado un gran auge a partir del surgimiento de GNU/Linux. Linus Torvals[21] expresaba “con muchas miradas, todos los errores saltarán a la vista”; criterio que marcó la evolución del desarrollo en comunidad, conocido como la “Ley de Linus”. Actualmente el desarrollo de componentes en las comunidades de usuario de software libre y código abierto se realiza a través de parches, los cuales son enviados por Internet, correo, foros o la *wiki*[22] a los desarrolladores de la comunidad.

En el estudio realizado por Eric Raymond[23] en La Catedral y el Bazar [30] se destaca la característica de compartir los parches con otros desarrolladores, de manera que varias personas puedan proponer una

21 Linus Benedict Torvalds: ingeniero de software conocido por iniciar y mantener el desarrollo del "*kernel*" de Linux.

22 Wiki: sitio web cuyas páginas pueden ser editadas directamente desde el navegador, donde los usuarios crean, modifican o eliminan contenidos que, generalmente comparten.

23 Eric Steven Raymond: uno de los principales teorizadores del movimiento del código abierto y uno de los fundadores del *Open Source Initiative.*

solución. Linus Torvals también hace referencia a este proceso cuando plantea *"libere rápido y a menudo, delegue todo lo que se pueda, delegue todo hasta el punto de promiscuidad"* [30]. En estos dos principios se basa el desarrollo en comunidad, en lograr que otras personas resuelvan los problemas de manera que puedan existir diversos criterios para escoger el más acertado. Eric Raymond en su estudio define una serie de características que son necesarias a la hora de realizar un desarrollo en comunidad:

- Identificar las necesidades personales del programador.
- Conocer que reescribir y reutilizar es una mejor opción que programar desde cero.
- Tener siempre presente que los usuarios son colaboradores y tratarlos como tal, es la forma más apropiada de mejorar el código y la más efectiva de depurarlo.
- Liberar rápido y a menudo.
- Con una base suficiente de desarrolladores asistentes casi cualquier problema puede ser caracterizado rápidamente y su solución ser obvia al menos para alguien.
- Lo más grande, después de tener buenas ideas, es reconocer las buenas ideas de sus usuarios. Esto último es a veces lo mejor.

Estas lecciones resumen el objetivo y el comportamiento del desarrollo en comunidad permitiendo una mejora sustancial de la calidad del software, mayor difusión y sostenibilidad en el tiempo. El estudio del desarrollo en comunidad es de gran importancia debido a que en el modelo a diseñar es necesario tener en cuenta los criterios planteados por la misma.

1.5 Experiencias en la UCI

En el presente epígrafe se pretende analizar la Guía metodológica para gestionar la integración de componentes, así como algunos de los desarrollos propios realizados en el departamento Sistemas Operativos. De manera que facilite el diseño del modelo de desarrollo de componentes para la distribución cubana de GNU/Linux Nova.

La Guía metodológica para gestionar la integración de componentes está enfocada a la gestión de los proyectos de CEDRUX[24], donde los componentes son desarrollados de forma independiente y es necesario integrarlos para que el sistema funcione. Los proyectos de CEDRUX aplican un modelo de desarrollo por componente donde cada componente se realiza de forma evolutiva. La Guía metodología que se analiza propone el modelo de Integración Continua debido a la gran aceptación que ha adquirido entre los desarrolladores de software en el mundo. La aplicación de una integración al finalizar el desarrollo de componentes provoca un cúmulo de errores, que pueden ser erradicados de manera constante si se aplica el método de Integración Continua.

Los roles encargados del proceso de integración son los mismos definidos en el proyecto CEDRUX. Aún cuando los roles no coinciden con los establecidos en el departamento Sistemas Operativos es de utilidad estudiarlos para conocer las responsabilidades que deben poseer los roles encargados del proceso de integración:

- Arquitecto de componente: es el principal responsable de la integración dentro de un proyecto, desempeña las tareas que estaban definidas para un arquitecto de proyecto y las propuestas para gestionar la integración de componentes, tanto interna como externa.
- Arquitecto del sistema: es el coordinador de la integración con otros sistemas, el encargado de velar porque se sigan los estándares especificados para el desarrollo y controlar el trabajo de los arquitectos de componentes en los proyectos.
- Desarrollador: implementa las funciones que le asigna el arquitecto de componentes de su proyecto. Aunque no es el rol de más responsabilidad, su trabajo es determinante para la calidad del proceso de integración.

A estos roles se le suman los relacionados con el control del trabajo: jefe de proyecto, líder de gestión y jefe de línea. Aunque la Guía metodológica para gestionar la integración de componentes detalla el

24 CEDRUX: sistema que gestiona un número importante de procesos empresariales donde la gestión de los procesos son traducidos en subsistemas y componentes en el sistema.

proceso de integración de componentes, no es aplicable al departamento Sistemas Operativos debido a que sólo está enfocada a los proyectos de CEDRUX.

1.5.1 Experiencias en el departamento Sistemas Operativos

Después de haber desplegado la distribución cubana de GNU/Linux Nova en su versión 3.0, en el departamento Sistemas Operativos se comenzaron a realizar una serie de desarrollos propios, los cuales satisfacían necesidades o complementaban servicios existentes. Todos eran realizados al margen de la planificación del proyecto, entre los más significativos se encuentran: ACF (Auditoría a código fuente), PPUSB (Protección al puerto USB), Moonlight (EE basado en una Arquitectura Orientada a Servicios).

Moonlight

EE donde la integración se alcanza internamente a través del agrupamiento de los diferentes módulos. La creación del nuevo EE surge porque se detectan requisitos que no pueden ser satisfechos con las aplicaciones que se encontraban en el repositorio. Los artefactos utilizados para el desarrollo fueron: código fuente, control de incidencias para lograr mejoras y corregir los errores detectados y la *wiki* para la documentación del proyecto, descripción de la arquitectura, la lista de requisitos y un grupo de instrucciones de como colaborar con el desarrollo del proyecto.

En el desarrollo del EE intervienen el jefe de proyecto y desarrolladores, los cuales realizan el proceso completo, aunque ninguno es especialista en ningún área a no ser la programación, motivo por el cual se apoyan en la comunidad. El proceso de desarrollo comienza cuando se identifica la visión del proyecto, los objetivos y lo que se quiere desarrollar. Luego se comparte con la comunidad y se aloja el proyecto en una forja de código. Se crea el equipo con los voluntarios, se diseña el flujo de trabajo a través de hitos (componentes) y se realiza el levantamiento de requisitos. Actualmente el proceso de integración con la distribución está desfasado debido a que la metodología que rige el desarrollo de la distribución no cubre el desarrollo de software propio de manera eficiente.

ACF

Software enfocado a la línea de seguridad que constituye uno de los cuatro principios por los que se rige la distribución cubana de GNU/Linux Nova. Garantiza la seguridad del código fuente y del repositorio de forma que no existan puertas traseras y problemas de inyección SQL. El proyecto surge a partir de la necesidad de garantizar que la distribución cubana cumpliera con uno de sus principios, la seguridad. Los artefactos utilizados fueron los contenidos en el expediente de proyecto 3.3 de Calisoft[25].

El proceso de desarrollo de ACF comienza cuando se detecta la necesidad de realizar un producto nuevo, a partir de esto se crea el equipo de desarrollo. Al crear el equipo de desarrollo se distribuye la carga de trabajo a través de los roles, los cuales eran otorgados en dependencia de las características de cada persona. Los roles desempeñados en el desarrollo de ACF son: líder de proyecto, analista, administrador de la calidad, administrador de configuración, arquitecto, probador, diseñador de casos de prueba, programador, diseñador de bases de datos, monitoreo y control. En ocasiones varios roles eran desempeñados por una misma persona.

Las tareas para el desarrollo de la aplicación eran controladas a través del GESPRO[26], las cuales se orientan semanalmente a cada trabajador o estudiante. Las tareas son enfocadas a la etapa en que se encuentra el proceso de desarrollo planteado por el programa de mejora, el cual es guiado por la metodología OpenUp. La integración del software con la metodología se realizó de forma manual, es decir, no siguió ningún proceso de desarrollo, simplemente fue subido al repositorio por los programadores. Dicha operación provoca que en ocasiones se encuentren errores a la hora de integrar el sistema con la distribución, los cuales pueden ser erradicados si se sigue un proceso de desarrollo capaz de revisar los errores en momentos tempranos y evaluar en el momento indicado cada parte de la distribución cubana de GNU/Linux Nova.

25 Calisoft: Centro Nacional de Calidad de Software.

26 GESPRO: herramienta utilizada para la gestión y control de proyectos.

PPUSB

PPUSB está enfocado a la protección de los puertos USB, como lo indica su nombre. Surge en el año 2011 cuando se decidió llevar a cabo el proceso eleccionario en el país utilizando tecnologías libres. Uno de los requerimientos planteados fue la necesidad de poder tratar de forma adecuada la información sensible que se manejaba en muchas estaciones de trabajo. Los sistemas de GNU/Linux en general carecían de una aplicación que hiciera dicha tarea por lo que se decide construir dicho producto. Para la creación del producto se utilizó la metodología ágil OpenUp, pero no se documentaron todos sus artefactos, sólo se hizo énfasis en los requerimientos e implementación de funcionalidades. Los artefactos utilizados fueron los contenidos en el expediente de proyecto 3.3 de Calisoft, principalmente especificación de requisitos, diseños de casos de prueba y vistas arquitectónicas. La integración del producto a la distribución al igual que ACF se realizó de forma manual, sin seguir ningún proceso de desarrollo, provocando que los componentes no estén correctamente documentados y esto afecte la reutilización de sus partes.

Valoraciones de los desarrollos propios realizados en la distribución cubana de GNU/Linux Nova.

Actualmente los desarrollos propios que son realizados por la distribución cubana de GNU/Linux Nova no cuentan con una guía para la realización del proceso de integración con la distribución. Los productos son realizados de maneras diferentes y en ocasiones no cumplen con todos los parámetros establecidos. Para la realización de los mismos se utilizan diversos artefactos, los cuales varían en dependencia de lo que se esté realizando. Por dicho motivo es necesario que exista un proceso capaz de controlar la realización de los desarrollos propios, los roles que van a guiar el proceso y los artefactos necesarios para lograr una buena documentación.

Conclusiones del capítulo

El estudio de modelos de desarrollo demostró que el modelo de desarrollo de componentes es el más apropiado a utilizar porque combina la técnica de reutilizar componente de las FSW y realizar iteraciones

de los modelos evolutivos. El PDC debe tener actividades relacionadas con la búsqueda de componentes, diseño de sistema y la implementación de componentes, así como las pruebas de los mismos. El modelo de componente a diseñar debe poseer la técnica de Integración Continua que propone XP, el ciclo de vida para los desarrollos propios que define Nova – OpenUp y algunos de los roles y responsabilidades que propone RUP.

Capítulo 2: Propuesta del modelo de desarrollo de componentes para la distribución cubana de GNU/Linux Nova

Introducción

En el presente capítulo se describen las características, principios, alcance y el ciclo de vida del modelo a diseñar. Se definen las actividades, roles y artefactos comprendidos en cada fase del ciclo de vida propuesto. De manera que se garantice la integración de los componentes a la distribución cubana de GNU/Linux Nova.

2.1 Modelo propuesto para el desarrollo de componentes de la distribución cubana de GNU/Linux Nova

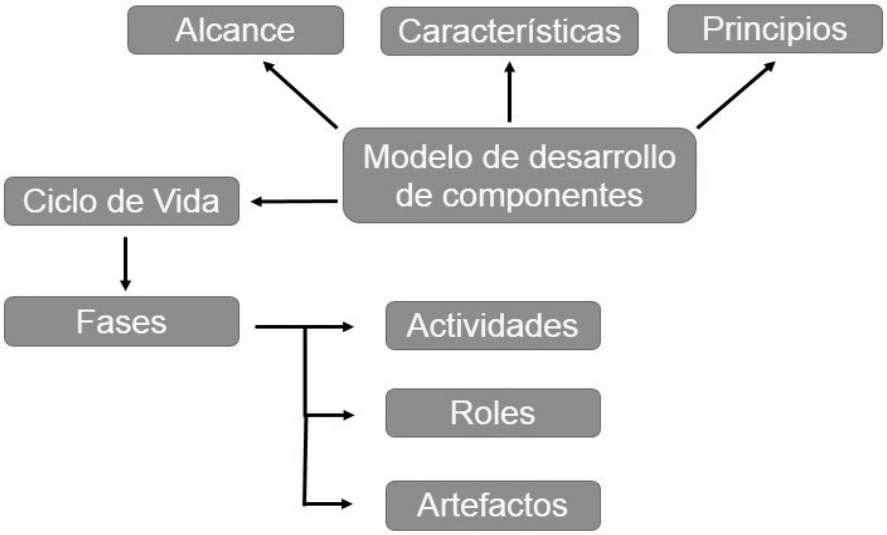


Figura 4: Estructura general del modelo de desarrollo de componentes para la distribución cubana de GNU/Linux Nova

Para lograr una buena aplicación del mismo es necesario que los componentes realizados para la distribución sean reutilizables de manera que puedan ser utilizados en otras ocasiones. Garantizando que se acorte el ciclo de vida de la distribución, una mayor productividad, mayor calidad, menor coste y menor tiempo de desarrollo. Para lograr una buena reusabilidad es necesario que los componentes posean las características descritas en el epígrafe 1.1.1.

El modelo de desarrollo de componentes propuesto está compuesto por fases, las cuales están conformadas por actividades, roles y artefactos. Se describen las características del modelo, así como su alcance y los principios que lo guían. La figura 4 muestra de manera general los elementos del modelo de componentes y sus relaciones.

2.1.1 Alcance

El modelo de desarrollo de componentes propuesto es aplicable para el proceso de desarrollo de los componentes realizados en el departamento Sistemas Operativos que tributen a la distribución cubana de GNU/Linux Nova.

2.1.2 Principios

El modelo persigue las tendencias de la CBSE (definir, implementar e integrar componentes) para lograr un mejoramiento en la integración del proceso de desarrollo de componentes con la distribución cubana de GNU/Linux Nova desarrollada en el departamento Sistemas Operativos. Además incluye algunas características de la metodología Nova - OpenUp (desarrollar evolutivamente para obtener retroalimentación y mejoramiento continuo, utilizar una infraestructura tecnológica adecuada que fomente el trabajo en entornos de software libre, aplicar métodos para la obtención de calidad en procesos y productos). A estos principios se le agregan los siguientes:

- Basado en componentes: se decide adoptar el principio basado en componentes debido a que el trabajo en la distribución se encuentra dividido en pequeñas partes que se van desarrollando de

manera independiente para luego integrarse. De esta manera se reducen los esfuerzos.

- Desarrollo colaborativo: debido a que constituye un faro para las distribuciones libres. Garantiza que un mayor número de errores sean detectados antes de liberar el producto final y que el tiempo de desarrollo sea menor ya que la comunidad brinda un gran apoyo, tanto en ideas como en la programación en sí.
- Trabajo con pocas personas y entornos cambiantes.

2.1.3 Características

Orientado a la arquitectura

El desarrollo de la distribución es muy cambiante, por lo que el modelo de desarrollo a proponer debe estar orientado a la arquitectura, garantizando que pueda adaptarse a los cambios realizados en fragmentos del sistema. Que el modelo sea orientado a la arquitectura posibilita que la arquitectura sea resistente y flexible, de manera que organice los componentes para que sean manejables, evitando que los cambios conlleven grandes esfuerzos.

Iterativo

El PDC en la distribución conlleva muchas horas de esfuerzo que puede durar varios meses, motivo por el cual es factible ir desarrollándolo en pequeñas partes, donde cada parte constituye una iteración. Las mismas se van realizando hasta obtener el producto final. El modelo propone una fase que funciona de manera transversal, denominado control de versiones, para controlar las iteraciones que se vayan realizando.

2.1.4 Elementos del modelo

El modelo está compuesto por tres fases que describen el ciclo de vida del desarrollo de los componentes y tres actividades de apoyo que funcionan a lo largo de todo el ciclo de vida. Cada fase contiene un grupo de actividades que son llevadas a cabo por determinados roles.

2.1.5 Roles y responsabilidades

En el modelo que se propone intervienen diferentes roles, los cuales coinciden con los propuestos por la metodología Nova – OpenUP para garantizar un mejor entendimiento. Se decidió no proponer nuevos roles debido a que los ya existentes pueden cubrir el desarrollo de componentes, solo es necesario agregar nuevas funcionalidades. Los roles a utilizar en el PDC para la distribución cubana de GNU/Linux Nova son: comunidad de usuarios, integrador de componentes, mantenedor de paquetes, arquitecto de software, programador, analista, planificador y documentador. Además de los roles antes mencionados se incluyen los relacionados con el control del trabajo: jefe de proyecto, administrador de la calidad y administrador de la configuración. Las responsabilidades de cada rol son descritas en el epígrafe 2.2.3.

2.2 Ciclo de vida del modelo de desarrollo de componentes

El ciclo de vida que se propone posee una estructura similar al definido por la metodología Nova – OpenUP, garantizando una mejor comprensión con la misma. Contiene una fase dedicada al análisis de requisitos, al diseño arquitectónico y al desarrollo de los componentes, así como actividades dirigidas al aseguramiento de la calidad, a la planificación y al control de versiones. En el modelo que se propone se agrega la descripción de cada una de las actividades por las que está compuesta cada fase, los roles que responden a cada una de ellas y los artefactos que deben ser utilizados. Las fases que conforman el ciclo de vida del modelo de desarrollo basado en componentes para la distribución cubana de GNU/Linux Nova se ven reflejadas en la figura 5.

2.2.1 Descripción de las fases y las actividades de apoyo

Requisitos: en esta fase se lleva a cabo las actividades relacionadas con el tratamiento de los requisitos. Su objetivo principal es obtener los parámetros necesarios para desarrollar el componente. Dicha fase responde a los requisitos establecidos por la distribución cubana de GNU/Linux Nova, de manera que puedan ser integrados con la misma y respondan a las necesidades existentes. Las actividades y

artefactos correspondientes a la fase requisitos se ven reflejados en la figura 6.

Diseño arquitectónico: durante esta fase se crea o reutiliza un marco de trabajo para llevar a cabo el desarrollo de los componentes de manera que satisfagan los requisitos establecidos. Ver figura 7.

Desarrollo e integración: se realiza la implementación e integración de los componentes con la distribución cubana de GNU/Linux Nova. Ver figura 8.

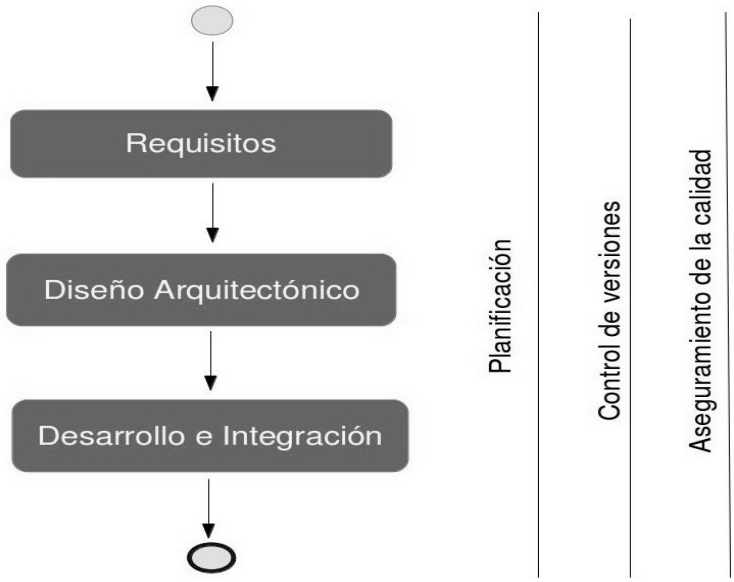


Figura 5: Ciclo de vida del modelo de desarrollo componentes para la distribución cubana de GNU/Linux Nova

Planificación: actividad que se lleva a cabo en todas las fases garantizando que los componentes sean desarrollados en el tiempo estimado y de esta manera poder ser integrados a la distribución.

Control de versiones: debido a que la distribución se realiza de manera iterativa y se aplica la técnica de Integración Continua es necesario realizar la actividad control de versiones para cada una de las iteraciones que se vayan realizando, de manera que se pueda controlar el trabajo realizado y la evolución del mismo.

Aseguramiento de la calidad: es una actividad que se lleva a cabo en todas las actividades del ciclo de vida, ya que la calidad es necesario controlarla desde la primera fase. Es necesaria para el proceso de desarrollo de componentes y de integración de los mismos.

2.2.2 Descripción de los artefactos

Los artefactos propuestos en el modelo de desarrollo de componentes coinciden en su mayoría con los definidos por la metodología Nova – OpenUp y en ocasiones son utilizados de la misma manera. Por otra parte los artefactos a utilizar para el PDC en la fase requisitos están enfocados en desagregar los requisitos definidos en la disciplina requisitos de la metodología antes mencionada. Los artefactos a utilizar en el desarrollo de componentes son: especificación de requisitos, criterios para validar requisitos, evaluación de requisitos, descripción de requisito ágil, lista de chequeo para detectar inconsistencia, registro de revisiones de inconsistencia y plantilla de no conformidades. En el PDC dichos artefactos no sufren modificaciones significativas, sólo son utilizados a nivel de componente y no a nivel de distribución cómo se realiza actualmente. Dicho cambio garantiza una mayor organización y facilita el trabajo debido que la gestión de requisitos se encuentra más enfocada a lo que se desea realizar.

Debido a que los artefactos que propone la metodología Nova – OpenUp no cubre el aspecto referente a la reusabilidad de los componentes es necesario crear una tabla donde se registren las características de cada componente. La tabla que se propone posee el nombre y las características necesarias para que un componente sea reutilizable, por lo que es necesario crear un artefacto nuevo. De esta manera se puede marcar las características que posee cada componente que se encuentre en el repositorio cuando se realice la actividad “Identificar componentes candidatos”.

Artefactos de la fase Diseño arquitectónico

Tabla 1: Descripción de los artefactos utilizados en la fase Diseño arquitectónico

Artefactos	Descripción

Requisito - componente	Registra la relación existente entre los componentes encontrados y los requisitos que satisfacen. De manera que se identifiquen aquellos requisitos que faltan por resolver.
Criterios para validar reusabilidad	Posee diversos criterios que permiten verificar si un componente es reutilizable o no.
Arquitectura vista del sistema	Se describe el sistema en función de los componentes, paquetes, relaciones, formas de interacción, escenarios y patrones de solución que supone la distribución cubana de GNU/Linux Nova.
Vista Lógica	Se crean los diagramas de clases, comunicación y secuencia.
Vista de Despliegue	Se crean los diagramas de componentes y de paquetes. (Opcional en dependencia del patrón arquitectónico que sigue un componente).

Artefactos de la fase Desarrollo e integración

Tabla 2: Descripción de los artefactos utilizados en la fase Desarrollo e integración

Artefactos	**Descripción**
Código fuente	Líneas de código que conforman el repositorio de código fuente.
Paquete de código fuente	Conjunto de archivos que contienen código que pueden ser compilados, posee un manual de la aplicación (para aplicaciones de usuario) o un manual de desarrollador (para bibliotecas).
Paquete binario	Constituye el paquete de código fuente compilado.
Plantilla de no conformidades	Se registran los errores encontrados al realizar las pruebas de integración.
Distribución de GNU/Linux Nova	El ISO de la distribución cubana de GNU/Linux Nova.

Artefactos de las actividades de apoyo

Tabla 3: Descripción de los artefactos utilizados en las actividades de apoyo.

Artefactos	**Descripción**
Cronograma	Se define y actualiza en la herramienta en el GESPRO.

Definiciones de la estimación del proyecto	Incluye toda la información necesaria para realizar las estimaciones de costo y esfuerzo así como la base para esas estimaciones y los puntos y/o circunstancias en las cuales será necesaria una re-estimación.
Plan de hitos internos	Un hito es un punto o evento significativo dentro del proyecto. Una lista de hitos identifica todos los hitos e indica si éstos son obligatorios, como los exigidos por contrato, o internos.
Notificación de escalabilidad del componente	Se suben las no conformidades
Registro de evaluación del componente	Se registran todas las revisiones que se van realizando a lo largo del desarrollo del proyecto.

2.2.3 Descripción de los roles

Los roles a utilizar en el desarrollo de componentes son los definidos por la metodología Nova – OpenUp con nuevas responsabilidades. En la tabla 4 se especifican las responsabilidades que deben cumplir cada uno.

Tabla 4: Descripción de los roles que intervienen en el proceso de desarrollo de componentes.

Roles	**Responsabilidades**	**Artefactos que realiza**
Jefe de proyecto	Participa en todas las actividades que conforman el proceso de desarrollo de componentes. Su función principal es la revisión, monitoreo y aprobación de las actividades realizadas.	Revisa los artefactos: • Descripción de requisito ágil a nivel de componentes. • Especificación de requisitos de software a nivel de componente. • Criterios para validar requisitos del componente. • Evaluación de requisitos de componentes. • Registro de revisiones de inconsistencia en requisitos de componentes.

		• Criterios para validar reusabilidad. • Requisitos componentes. • Plantilla de no conformidades. • Las funcionalidades implementadas y los componentes desarrollados. • Plan de desarrollo de software.
Administrador de la configuración	Identifica los elementos de la configuración relacionados con los componentes. Se encarga del control de versiones.	Crea: • Integra las funcionalidades. • Registro de problemas desviaciones. • Registro de estado del proyecto.
Administrador de la calidad	Se encarga de verificar la calidad de las actividades de cada fase del ciclo de vida del PDC. Participa en la elaboración de los planes de pruebas de unidad e integración de componentes y en las revisiones técnicas de los artefactos. Participa en las revisiones generales de componentes implementados.	Crea los artefactos: • Evaluación de requisitos de componentes. • Registro de revisiones de inconsistencia en requisitos de componentes. • Plantilla de no conformidades de requisitos. Consulta los artefactos: • Criterios para validar requisitos del componente. Revisa los artefactos: • Especificación de requisitos a nivel de componente. • Descripción de requisitos ágil a nivel de componente. • Arquitectura vista del sistema. • Vista de despliegue. • Vista lógica. • Pruebas unitarias.

		• Pruebas de integración. • Las funcionalidades implementadas y los componentes desarrollados.
Comunidad de usuarios	Participa en el levantamiento de requisitos, la búsqueda de componentes candidatos y en el desarrollo de los componentes.	Revisa: • Registro de revisiones de inconsistencia de componentes. • Descripción de requisito ágil a nivel de componente. • El código fuente.
Integrador de componentes	Se encarga de subir al repositorio los componentes que se realicen. Participa en el diseño de los mismos. Realiza la búsqueda de componentes en el repositorio. Informa cuando existen atrasos en el proyecto a causa de la integración.	Sube al repositorio el código binario. • Integra las funcionalidades. • Realiza las pruebas de integración de funcionalidades y de componentes. • Plantilla de no conformidades (de la integración de funcionalidades y componentes). Consulta: • Descripción de requisito ágil a nivel de componente.
Mantenedor de paquetes	Participa en la búsqueda y selección de componentes candidatos. Se encarga de realizar la relación que existe entre los componentes encontrados y los requisitos establecidos. Participa en la implementación e integración de los componentes y es el encargado del empaquetado de los mismos.	Compila el código fuente. • Actualiza el repositorio. • Sube el código binario. Consulta: • Descripción de requisito ágil a nivel de componentes. • Estándar de empaquetamiento.
Arquitecto de	Encargado de realizar las vistas	Crea los artefactos:

software	arquitectónicas de los componentes. Participa en la búsqueda y selección de componentes candidatos así como en el diseño del marco de trabajo.	• Arquitectura vista del sistema. • Vista de despliegue. • Vista lógica. Revisa los artefactos: • Especificación de requisitos a nivel de componente. • Descripción de requisitos ágil a nivel de componente.
Programador	Realiza la implementación y prueba de los componentes. Notifica los errores que se encuentren en algunos de los servicios que utiliza el componente.	Crea: • Código fuente. • Compila código fuente. Realiza: • Pruebas unitarias. Consulta: Descripción de requisitos ágil a nivel de componente. • Arquitectura vista del sistema. • Vista de despliegue. • Vista lógica.
Analista	Encargado de los artefactos relacionados con el levantamiento de requisitos. Participa en las actividades de la fase Diseño arquitectónico.	Crea los artefactos: • Especificación de requisitos de software a nivel de componente. • Descripción de requisito ágil a nivel de componente. • Criterios para validar requisitos del componente. Consulta los artefactos:

		• Estado del arte del producto a desarrollar.
Documentador	Se encarga de llevar toda la documentación del proceso de desarrollo de componentes.	Realiza: • Descripción de los paquetes. • Manual de usuario • Manual de desarrollo.
Planificador	Realiza el cronograma y se encarga del cumplimiento del mismo y la estimación de la duración de los componentes.	Realiza: • Plan de desarrollo de software. • Definiciones de estimaciones del componente. • Cronograma.

Actividades de apoyo

Las actividades planificación, control de versiones y aseguramiento de la calidad constituyen actividades de apoyo que son realizadas a lo largo de todo el ciclo de vida del PDC.

Actividad: Planificación

Descripción: Se lleva a cabo en cada una de las actividades de cada fase con el objetivo de verificar el tiempo de duración de cada actividad, para de esta manera poder estar acoplada al proceso de desarrollo definido por la metodología Nova – OpenUp.

Artefactos de entrada:

- Lista de chequeo para detectar inconsistencia en requisitos (distribución)
- Registro de revisiones de inconsistencia en requisitos (distribución)
- Acta de aceptación (distribución)
- Plan de datos (distribución)

Artefactos de salida:

- Cronograma

- Plan de desarrollo de software
- Definiciones de la estimación del componente

Roles: planificador, jefe de proyecto.

Actividad: Control de versiones

Descripción: se realiza un control de versiones para verificar cada una de las iteraciones realizadas así como la correcta integración con el repositorio. Actualmente el control de versiones se lleva a cabo a través del Subversion.

Artefactos de salida:

- Registro de problemas de desviaciones
- Registro del estado del proyecto

Roles: jefe de proyecto, administrador de la configuración, analista.

Actividad: Aseguramiento de la calidad

Descripción: en dicha actividad se verifican las normas de calidad que se deben cumplir en cada fase. Se ofrecen reportes que permiten a los proyectos tener una visión objetiva de la calidad de los productos y servicios y de los procesos ejecutados para desarrollarlos.

Artefactos de entrada:

- Notificación de escalabilidad
- Registro de evaluaciones del proyecto

Artefactos de salida:

- Notificación de escalabilidad (del componente)
- Registro de evaluaciones del proyecto (del componente)

Roles: administrador de la calidad, jefe de proyecto.

Actividades de la fase Requisitos (Ver figura 6).

Actividad # 1: Identificar requisitos de componentes

Descripción: se realiza la captura de los requisitos funcionales y no funcionales a partir de los requisitos identificados por la distribución. El principal objetivo es identificar los requisitos propios de cada componente.

Artefactos de entrada:

- Estado del arte del producto a desarrollar
- Oferta (Opcional)
- Identificación de requisitos

Artefactos de salida:

- Especificación de requisitos de software a nivel de componentes

Roles: analista, jefe de proyecto, documentador, arquitecto de software, comunidad de usuarios.

Actividad # 2: Especificar requisitos de componentes

Descripción: después de identificar los requisitos por cada componente se especifican de manera que los involucrados conozcan los que se debe desarrollar.

Artefactos de entrada:

- Estado del arte del producto a desarrollar
- Especificación de requisitos de software a nivel de componentes

Artefactos de salida:

- Criterios para validar requisitos del componente
- Evaluación de requisitos de componentes
- Especificación de requisitos de software a nivel de componente (Refinada)

Roles: analista, jefe de proyecto, documentador.

Actividad # 3: Detallar requisitos de componentes

Descripción: se describe cada requisito de manera detallada abordando cada una de sus características.

Artefactos de entrada:

- Especificación de requisitos de software a nivel de componente

Artefactos de salida:

- Descripción de requisito ágil a nivel de componente

Roles: analista, arquitecto de software, documentador, jefe de proyecto, comunidad de usuarios.

Actividad # 4: Verificar requisitos de componentes

Descripción: se evalúan los requisitos para determinar si la descripción de requisitos se realizó de manera correcta. En esta actividad es necesario tener en cuenta la verificación de requisitos que se realiza a nivel de distribución.

Artefactos de entrada:

- Descripción de requisito ágil a nivel de componente
- Lista de chequeo para detectar inconsistencias en requisitos de componentes

Artefactos de salida:

- Registro de revisiones de inconsistencia en requisitos de componentes

Roles: analista, jefe de proyecto, documentador, administrador de la calidad.

Actividad # 5: Validar requisitos de componentes

Descripción: se comprueba que los requisitos estén correctos de acuerdo a las características que debe satisfacer el producto. Se tiene en cuenta la validación de requisitos a nivel de distribución.

Artefactos de entrada:

- Especificación de requisitos de software a nivel de componente
- Descripción de requisito ágil a nivel de componente

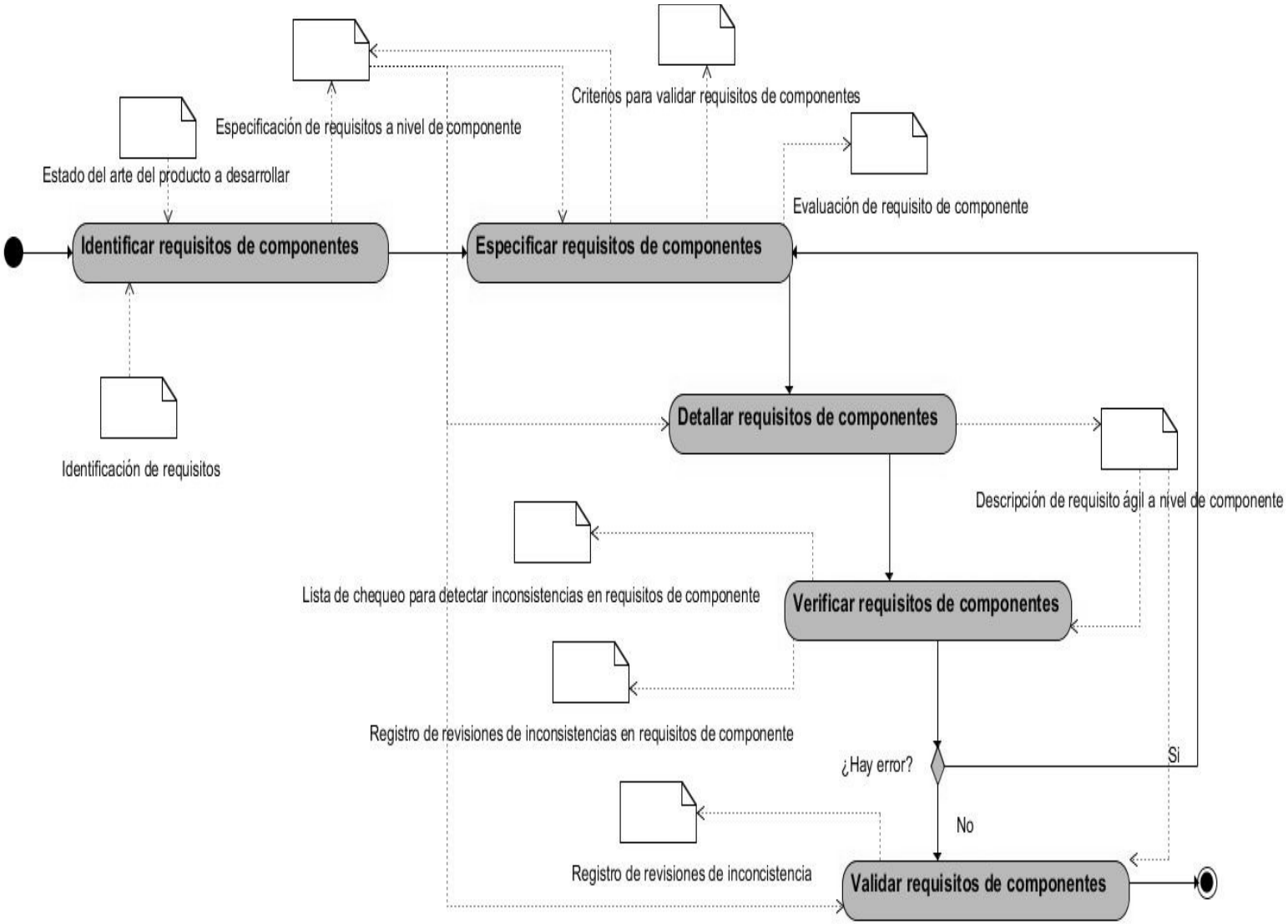


Figura 6: Relación de las actividades y artefactos de la fase Requisitos

Artefactos de salida:

- Registro de revisiones de inconsistencia en requisitos de componentes
- Plantilla de no conformidades

Roles: analista, administrador de la calidad, jefe de proyecto, comunidad de usuarios, documentador.

Actividades de la fase Diseño arquitectónico (Ver figura 7).

Actividad # 1: Identificar componentes candidatos

Descripción: se realiza la búsqueda de componentes en repositorios a partir de los requisitos definidos con el objetivo se encontrar componentes que puedan ser reutilizables o que satisfagan las necesidades existentes.

Artefactos entrada:

- Especificación de requisitos de software a nivel de componente
- Descripción de requisito ágil a nivel de componente

Artefactos de salida:

- Componente
- Requisito_componente

Roles: analista, integrador de componentes, mantenedor de paquetes, jefe de proyecto, comunidad de usuarios.

Actividad # 2: Seleccionar componentes

Descripción: se analizan los componentes que hayan sido encontrados con el objetivo de verificar si satisfacen todas las necesidades o si es necesario realizar alguna modificación.

Artefactos de entrada:

- Especificación de requisitos de software a nivel de componente
- Descripción de requisito ágil a nivel de componente

- Componente
- Requisito_componente

Artefactos de salida:

- Componente (opcional, solo cuando existen componentes que satisfacen algún requisito)
- Criterios para validar reusabilidad (opcional, solo cuando existen componentes que
- satisfacen algún requisito)

Roles: analista, integrador de componentes, mantenedor de paquetes, jefe de proyecto.

Actividad # 3: Diseñar el sistema

Descripción: se diseña el marco de trabajo, tiene como principal objetivo el desarrollo de software propio.

Artefactos de entrada:

- Arquitectura de software
- Arquitectura vista de sistema
- Especificación de requisitos de software a nivel de componente
- Descripción de requisito ágil a nivel de componente

Artefactos de salida:

- Vista despliegue (Opcional)
- Vista lógica
- Arquitectura vista del sistema

Roles: arquitecto de software, integrador de componentes, jefe de proyecto, analista, documentador.

Actividad # 4: Diseñar el sistema con reutilización

Descripción: se diseña o reutiliza el marco de trabajo a partir de los componentes encontrados en la etapa análisis de requisitos. Esta actividad tiene implícita tres actividades con el objetivo de seleccionar los componentes adecuados: calificación de componentes (garantiza que el componente seleccionado realice

la función requerida), adaptación de requisitos (se realiza el encubrimiento de caja blanca, caja gris y caja negra, con el objetivo de eliminar las inconsistencias que puedan aparecer), composición de componentes (se encarga del ensamblaje de los componentes que hayan sido calificados y adaptados correctamente).

Artefactos de entrada:

- Especificación de requisitos de software a nivel de componente
- Descripción de requisito ágil a nivel de componente
- Componente a modificar
- Arquitectura de software
- Arquitectura vista de sistema

Artefactos de salida:

- Vista despliegue (Opcional)
- Vista lógica
- Arquitectura vista del sistema

Roles: arquitecto de software, integrador de componentes, analista, jefe de proyecto, documentador.

Actividades de la fase Desarrollo e integración (Ver figura 8).

Actividad # 1: Implementar

La actividad Implementar está compuesta por varias subactividades (Implementar, Pruebas unitarias, Integrar, Pruebas de integración).

Subactividades de la actividad Implementar (Ver figura 9).

Subactividad # 1: Implementar

Descripción: se realiza la implementación de los componentes para satisfacer los requisitos establecidos. En caso de que no se haya encontrado ningún componente se realiza un desarrollo propio y de haberse

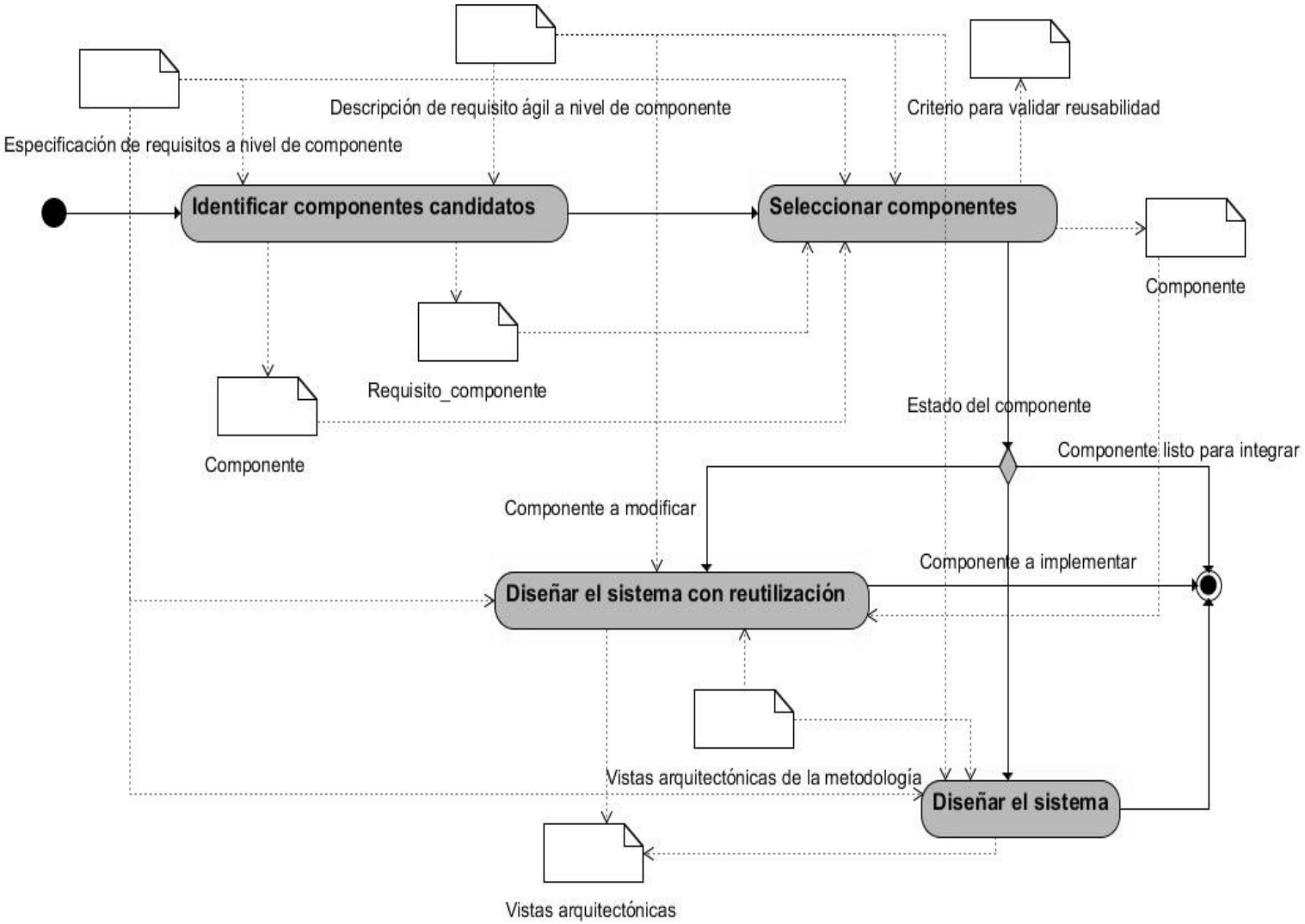


Figura 7: Relación de las actividades y artefactos de la fase Diseño Arquitectónico

encontrado alguno se implementan sólo aquellos requisitos para los cuales no se haya encontrado solución.

Artefactos de entrada:

- Vista de despliegue (Opcional)
- Vista lógica
- Arquitectura vista del sistema
- Especificación de requisitos de software a nivel de componente
- Descripción de requisito ágil a nivel de componente
- Componente a modificar (Opcional solo si se encuentran componentes candidatos)

Artefactos de salida:

- Código fuente

Roles: mantenedor de paquetes, programador, comunidad de usuarios, jefe de proyecto.

Subactividad # 2: Realizar pruebas unitarias

Descripción: se comprueba el correcto funcionamiento del código. Sirve para asegurar que cada uno de los pequeños trozos de código funcione correctamente facilitando las pruebas de integración.

Artefactos de entrada:

- Código fuente

Artefactos de salida:

- Código fuente
- Plantilla de no conformidades o trabajo con el GESPRO

Roles: programador, comunidad de usuarios, jefe de proyecto, documentador.

Subactividad # 3: Integrar funcionalidades

Descripción: integrar las funcionalidades que se vayan realizando para ir confeccionando el componente.

Artefactos de entrada:

- Código fuente
- Código binario

Artefactos de salida:

- Código fuente
- Código binario

Roles: programador, comunidad de usuarios, integrador de componentes, jefe de proyecto.

Subactividad # 4: Realizar pruebas de integración

Descripción: se realizan pruebas para unir las funcionalidades que se han implementado. Las funcionalidades son combinadas y probadas como un grupo.

Artefactos de entrada:

- Código fuente
- Código binario

Artefactos de salida:

- Código fuente
- Plantilla de no conformidades o trabajo con el GESPRO

Roles: programador, comunidad de usuarios, integrador de componentes, jefe de proyecto, documentador.

Actividad # 2: Empaquetar

Descripción: se empaqueta el código fuente luego de ser revisado. Las aplicaciones son proporcionadas en forma de paquetes los cuales están formados por los programas ejecutables de la aplicación, así como por todas las bibliotecas de las que depende y otros tipo de ficheros (como imágenes, ficheros de audio, traducciones y localizaciones), de forma que se proporcionan como un conjunto.

Artefactos de entrada:

- Código fuente
- Estándar de empaquetamiento

Artefactos de salida:

- Paquete de código fuente

Roles: mantenedor de paquetes, jefe de proyecto, integrador de componente.

Actividad # 3: Integrar componentes

Descripción: se unen los módulos que se realicen y se sube el paquete de código fuente al repositorio.

Artefactos de entrada:

- Paquete de código fuente

Artefactos de salida:

- Paquete fuente
- Paquete binario

Roles: mantenedor de paquetes, integrador de componentes, jefe de proyecto.

Actividad # 4: Realizar pruebas de integración

Descripción: se realizan pruebas de integración para verificar que la integración con los módulos y con la distribución cubana de GNU/Linux Nova se realizó de manera satisfactoria.

Artefactos de entrada:

- Diseño de casos de prueba
- Sistema GNU/ Linux Nova
- Repositorio

Artefactos de salida:

- Plantilla de no conformidades o trabajo con el GESPRO

Roles: integrador de componente, comunidad de usuarios, jefe de proyecto, documentador.

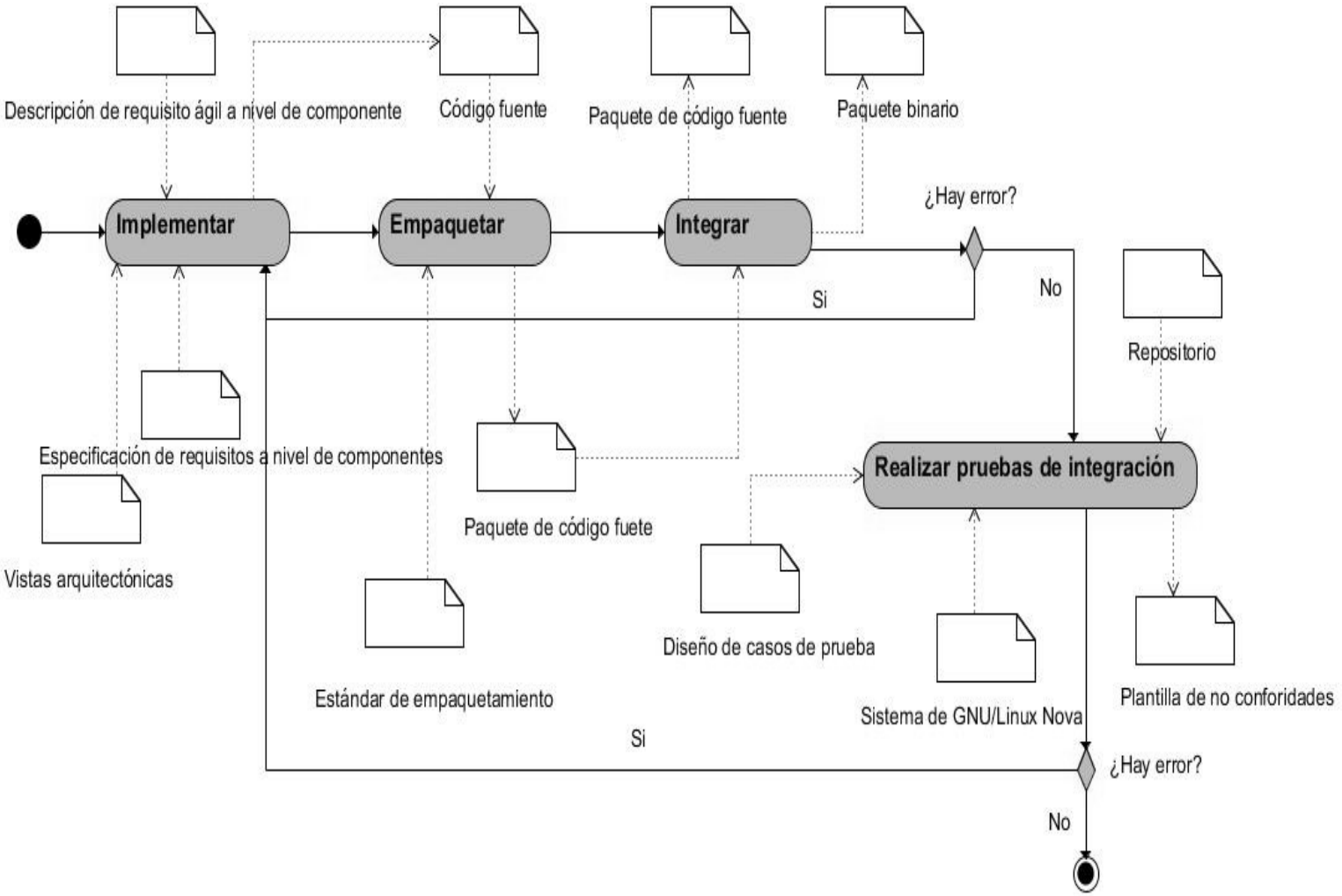


Figura 8: Relación de las actividades y artefactos de la fase Desarrollo e integración

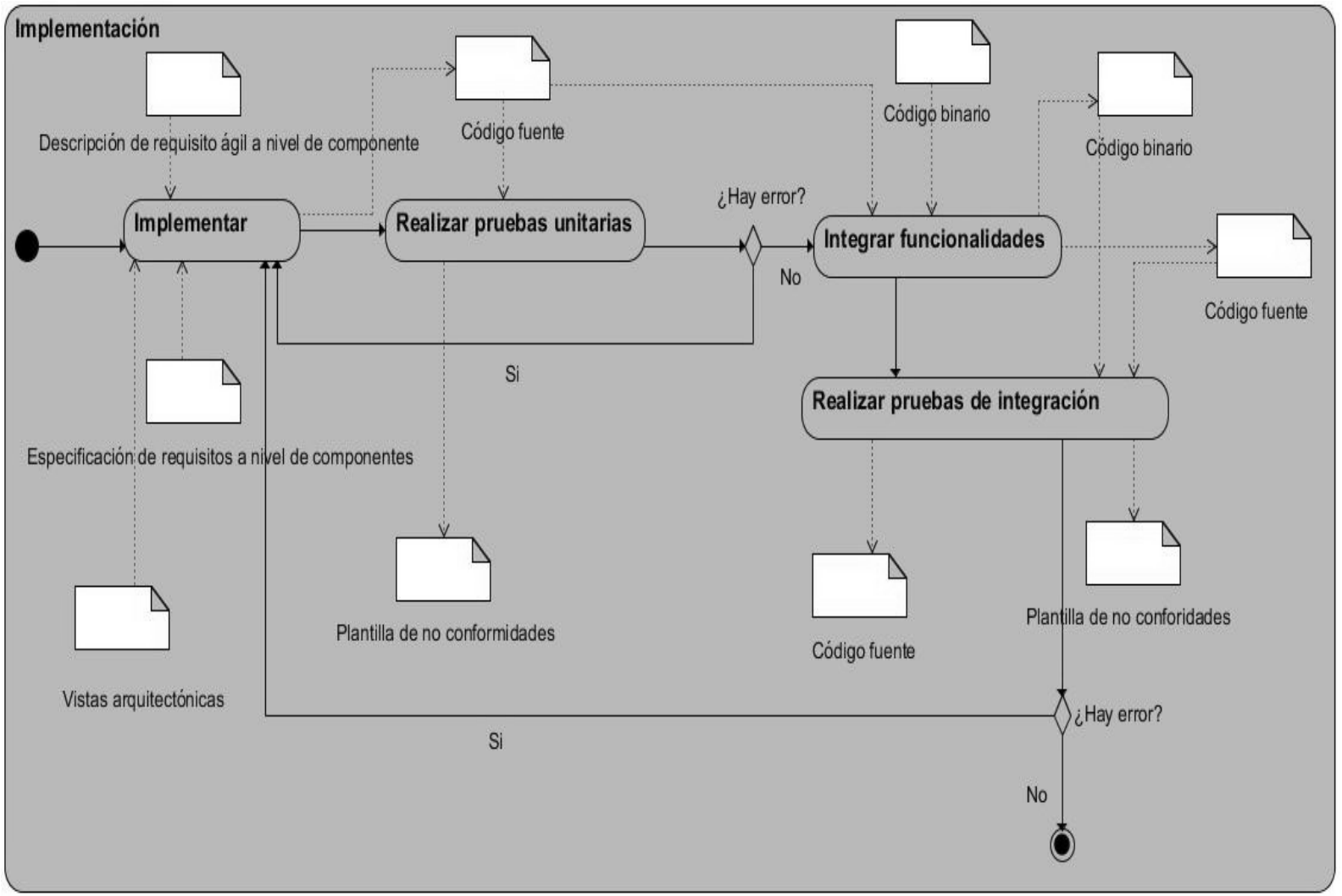


Figura 9: Relación de las subactividades y artefactos de la actividad Implementar

Conclusiones del capítulo

En el presente capítulo se da cumplimiento al objetivo relacionado con el diseño del modelo de desarrollo de componente para la distribución cubana de GNU/ Linux Nova. Con la modificación parcial del ciclo de vida de los desarrollos propios propuesto por la metodología Nova – OpenUp se facilita el PDC y la integración de los mismos con la distribución cubana de GNU/Linux Nova. La propuesta de un conjunto de principios, características, fases, actividades, roles y artefactos posibilitó la creación de un modelo de desarrollo de componentes que sirven de guía para el PDC.

Capítulo 3: Evaluación de la propuesta de solución

Introducción

En el presente capítulo se pretende evaluar el modelo de desarrollo de componentes que se propone utilizando el método experimental, de manera que se verifique el cumplimiento del problema de investigación y la correcta realización del modelo diseñado.

3.1 Método experimental

La aplicación del método experimental se basa en la definición de una o varias variables de estudio para controlar el cambio del valor de la misma. Su objetivo principal es afectar las variables independientes y observar el efecto que causa en las variables dependientes [31]. Para la aplicación del método se determinaron tres variables fundamentales que responden a la situación problemática existente (planificación, reutilización, integración).

3.2 Aplicación del método

A partir del estudio realizado a los desarrollos propios implementados en el departamento de Sistemas Operativos, se determinó que los principales problemas con el desarrollo de los componentes están enfocados en su mayoría a la planificación porque aumenta el índice de desviaciones, la reutilización de los componentes y la integración de los mismos con la distribución. Dichos problemas son causados debido a que no se cuenta con las fases, actividades, roles y artefactos que definan su proceso de desarrollo. Por dicho motivo se crea un modelo de desarrollo de componentes que sea capaz de guiar todo el proceso.

Para el cumplimiento del experimento se tomó como población los desarrollos propios realizados para la distribución cubana de GNU/Linux Nova y como muestra el EE del proyecto Nova Ligero. Se decide tomar Guano como el ejemplo al que no se le aplica el modelo de desarrollo de componentes debido a que

constituye el antecesor de Moonlight que sería el escenario donde se aplica el modelo propuesto de manera que se puedan observar los cambios que se vayan realizando. Para verificar las variables propuestas fue necesario definir diferentes indicadores basándose fundamentalmente en algunas de las fases planteadas en el modelo de desarrollo de componentes.

Variable # 1: Planificación

Para medir la presente variable se definieron diferentes indicadores:

1. Grado de utilización de un modelo de desarrollo de componentes.
2. Roles que se dedican al desarrollo de componentes.
3. Artefactos utilizados en el desarrollo de componentes.

Variable # 2: Integración

1. Actividades relacionadas con los requisitos.
2. Actividades relacionadas con el diseño arquitectónico.
3. Actividades relacionadas con el desarrollo e integración de los componentes.

Variable # 3: Reutilización

1. Impacto de la calidad de la documentación en la actividad identificación de componentes candidatos.

3.2.1 Variable Planificación

La planificación en el desarrollo de un producto software establece qué tareas realizar y cuándo se deben hacer, así como los recursos a utilizar. Cuando se realiza una buena planificación se puede determinar si el proceso está marchando en tiempo y si se están utilizando los recursos (tiempo, personas y dinero) de manera esperada.

En el momento en que Guano fue realizado no se contaba con un modelo de desarrollo de componente que guiara el proceso. Las actividades eran realizadas de manera empírica por lo que no eran incluidas en

la planificación de la distribución. Al no poseer un proceso que detallara que actividades realizar, cuándo y quienes la realizarían, existían atrasos en la entrega del producto e incluso desviaciones en el cronograma de la distribución. Además no se contaba con roles específicos para cada actividad del desarrollo de componentes y ni siquiera se conocía dicho proceso.

Al no existir una especificación de los roles el proceso resultaba desorganizado debido a que las personas no tenías responsabilidades marcadas lo que provocaba atrasos en los cronogramas. En el momento que Guano fue creado cuatro de los roles que se proponen en el modelo no existían. En ocasiones las responsabilidades de estos se encontraban divididas en diferentes roles. Los artefactos por otra parte existían, pero eran analizados a nivel de distribución y no a nivel de componente, lo que hace más engorroso el trabajo ya que para analizar los requisitos a realizar era necesario analizar los requisitos de la distribución que muchas veces no detallaba como debía ser el componente a integrar.

Moonlight aplica el modelo de desarrollo de componentes demostrando que al especificar las actividades se puede establecer la cantidad de personas que trabajará en cada una de ellas y el tiempo requerido para su ejecución. El establecimiento de los roles involucrados en cada actividad posibilita que cada persona posea una tarea específica y responda por el cumplimiento de la misma, de manera que se cumpla en el tiempo que se tiene establecido. Al desagregar los artefactos existentes se logra que cada rol posee la información que necesita sin necesidad de conocer toda la información de la distribución. Además se logra establecer la relación entre cada componente y los requisitos que satisface a través de un nuevo artefacto denominado Requisito_componente. Debido a que al establecer un modelo de desarrollo de componentes se logra un mayor control de lo que se debe realizar es posible disminuir el índice de desviaciones y realizar una mejor planificación.

3.2.2 Variable Integración

La variable integración representa el PDC de manera que pueda ser unido a la distribución con la menor cantidad de errores posibles. Guano es un EE realizado completamente antes de ser integrado. Este

método provocó que la mayor parte de los errores fueran detectados al finalizar el proceso, arrastrándolos desde las primeras etapas de desarrollo hasta la etapa de liberación del producto, lo cual aumentó el costo de desarrollo del mismo. Dicho problema aumenta el esfuerzo y alarga el plazo de terminación de los componentes, debido a que es necesario tener una etapa más en el proceso de desarrollo encargada de las pruebas, tanto unitarias como de integración. Otro aspecto que dificultaba la integración era la gestión de requisitos debido que eran analizados a nivel de distribución, los requisitos no eran específicos para algún componente y en ocasiones las plantillas de Especificación de requisitos eran muy extensas provocando un mayor esfuerzo a la hora de comprender lo que se quería realizar. La arquitectura no era específica de los componentes, por lo que cuando se realizaba alguno era necesario analizar toda la arquitectura de la distribución que en ocasiones omitía aspectos relevantes de la estructura, comportamiento y composición de los componentes.

En Moonlight la fase Requisitos es realizada a nivel de componentes garantizando que se detalle un poco más lo que se desea realizar, de manera que el desarrollador logre enfocarse en cada funcionalidad. La arquitectura también está enfocada al componente y tiene en cuenta la arquitectura de aquellos componentes que fueron reutilizados más la propia de la distribución a la cual será integrado el componente. En Moonlight se aplica la técnica de Integración Continua permitiendo la realización de pruebas unitarias y de integración a medida que se vaya implementando, dando la posibilidad de corregir los errores de manera temprana. La aplicación de dicha técnica garantiza que al finalizar el producto los errores sean mínimos.

3.2.3 Variable Reutilización

La efectividad de la reutilización de componentes se encuentra relacionada directamente con la calidad de la documentación, la cual está dada por la forma en que refleja las características del software. En Guano se realizó la descripción de los paquetes de acuerdo a la guía de empaquetado de Debian, la misma no cubre características importantes del software que influyen en la búsqueda de componentes. La

documentación que se generó en Guano consta de tres parámetros fundamentales, nombre, sinopsis y una descripción de manera general del componente. Dicha descripción no abarca todos los indicadores necesarios para realizar una buena búsqueda, por lo que a la hora de recuperar un paquete del repositorio se hacía más difícil, logrando que en la mayoría de las ocasiones fuera necesario desarrollar los componentes desde cero.

En Moonlight se agregan nuevos parámetros relacionados con el lenguaje de programación, requisitos, funcionalidades y la descripción de las interfaces. Dichas características facilitan a los usuarios la búsqueda de componentes en el repositorio debido a que detallan más sus funcionalidades. La correcta descripción de los componentes garantiza que puedan ser utilizados en futuros proyectos, disminuyendo el tiempo de desarrollo y el esfuerzo de los trabajadores, así como una disminución en los costos de desarrollo.

La siguiente tabla es el resultado del análisis de las actividades que forman parte del modelo propuesto teniendo en cuenta la presencia de éstas en el proceso de desarrollo de Guano y Moonlight. Además se tuvo en cuenta la opinión de desarrolladores que participaron en la confección de Guano y que actualmente forman parte del equipo de desarrollo de Moonlight.

Tabla 5: Comparación de un antes y un después de aplicado el modelo a partir de indicadores.

No.	Variables	% Antes	% Después
1	planificación	40	80
2	integración	30	100
3	reutilización	20	90

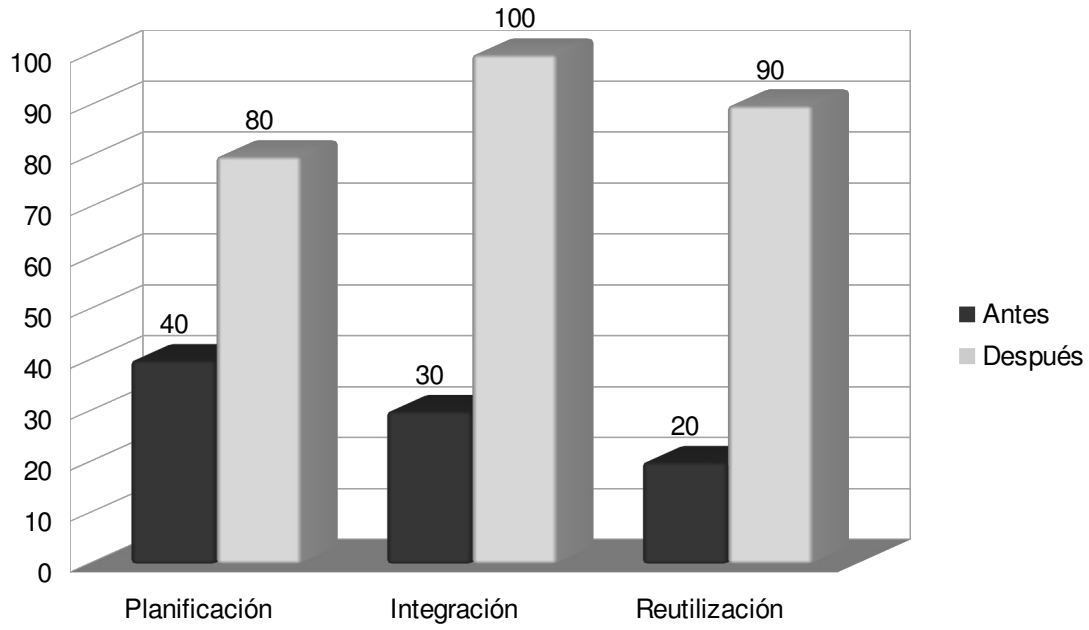


Figura 10: Valores obtenidos por indicadores.

La figura 10 muestra los resultados obtenidos por cada variable antes y después de haber aplicado el modelo. Teniendo en cuenta las variables analizadas se puede afirmar que el modelo de desarrollo de componentes que se propone contribuye a la integración de los desarrollos propios con la distribución. Además disminuye los errores de implementación en fases tempranas. La correcta documentación de los componentes implementados permite su posterior reutilización, mejorando la calidad del producto a desarrollar debido a que los componentes que se reutilizan se encuentran probados, reduciendo el tiempo de desarrollo y el esfuerzo de los trabajadores. El modelo propuesto apoya la identificación de requisitos detallando las características y cualidades que debe tener cada desarrollo propio.

Conclusiones del capítulo

La aplicación del modelo experimental demostró que el modelo de desarrollo de componentes puede contribuir a la mejora del proceso de integración de los desarrollos propios realizados para la distribución cubana de GNU/Linux Nova.

Conclusiones generales

Se determina como conclusiones de la presente investigación que:

- El estudio de los diferentes modelos de desarrollo demostró que el modelo basado en componentes es el más aplicable para lograr la integración de desarrollos propios con la distribución cubana de GNU/Linux Nova porque combina la técnica de realizar iteraciones, reutilizar componentes y describir las actividades del proceso de desarrollo.
- La descripción de actividades, roles y artefactos enfocados al proceso de desarrollo de componentes de una distribución influyó positivamente en la disminución de errores en etapas finales de desarrollo, en la mejora de la planificación y en la creación de componentes reutilizables así como la integración de los mismos con la distribución cubana de GNU/Linux Nova.
- La evaluación del modelo propuesto a través de la aplicación del método experimental demostró que con la aplicación del mismo se pueden disminuir los problemas existentes en cuanto a planificación, reutilización e integración y favorecer la calidad del producto final.

Recomendaciones

- Perfeccionar el modelo de desarrollo de componentes a través de la aplicación del mismo para lograr que sea más completo.
- Detallar las características que deben poseer las interfaces de los componentes.

Bibliografía

1 PIERRA FUENTES, Allan. Nova, distribución cubana de GNU/Linux : re-estructuración estratégica de su proceso de desarrollo. Tesis de maestría. Universidad de la Ciencias Informáticas. La Habana, 2011.

2 PÉREZ VILLAZÓN, Yoandy, et al. Buenas prácticas para la migración a código abierto. Cuba, UCI, 2014. 56 p.

3 GOÑI ORAMAS, Angel. Metodología para la gestión de proyectos de Consultoría en Migración a Tecnologías de Software Libre y Código Abierto. Tesis de maestría. Universidad de las Ciencias Informática. La Habana, 2012.

4 FÍRVIDA DOMÉSTEVEZ, Abel; RODRÍGUEZ PINO, Adisleydis. Guano, entorno de escritorio cubano, libre y de código abierto. Tesis de pregrado. Universidad de las Ciencias Informáticas. La Habana, 2009.

5 FERNANDEZ DEL MONTE, Yusleydi. Metodología para desarrollar la distribución cubana de GNU/Linux Nova. La Habana, UCI, 2013. 98p.

6 FERNANDEZ DEL MONTE, Yusleydi. [En línea]. Metodología para desarrollar la distribución cubana de GNU/Linux Nova, 2009. [Consultado el 28 de enero del 2015]. Disponible en: http://repositorio_institucional.uci.cu/jspui/handle/ident/8096.

7 PRESSMAN, Roger S. Métodos convencionales para la ingeniería del software. En: Darrel Ince. Ingeniería del software un enfoque práctico. México: 2001. Parte III, 258 p.

8 SZYPERSKI, Clemens. Component software: Beyond object-oriented programing. Addison-Wesley Pub Co, 2da edición, noviembre, 2002.

9 SOMERVILLE, Ian. Software engineering. Addison-Wesley Pub Co, 6ta edición, Agosto 2000.

10 BACHMANN, F, et al. Volumen II: Technical concepts of component-based software engineering, 2da

edición. Technical report, Software Engineering Institute, Carnegie Mellon University, Julio 2000.

11 ALEGSA, Leandro. Definición de paquete de software. [En línea] ¿Cuál es la definición de Paquete de software, 2009. [Consultado el 14 de febrero del 2015]. Disponible en: http://www.alegsa.com.ar/Dic/paquete%20de%20software.php

12 AVILA, Kety. ¿Qué es un paquete de software?. [En línea] ¿Qué es un paquete de software?. [Consultado el 15 de febrero del 2015]. Disponible en: http://www.cavsi.com/preguntasrespuestas/que-es-un-paquete-de-software

13 BERNAL, Oscar. Tipos de paquetes en linux/unix. [En línea] Tipos de paquetes en linux/unix. [Consultado el 14 de febrero del 2015]. Disponible en: http://www.oscarbernal.net/index.php?/content/view/36/20/

14 COUNCILL, W. T. ; HEINEMAN, G. T. Definition of a software component and its elements, 2001.

15 MATILVA, Jonás; ARAPÉ, Nelson; COLMENARES, Juan Andrés. Desarrollo de software basado en componentes. Venezuela. 9 p.

16 LARSSON, S. Improving software product integration. Department of Computer Science and Engineering Sweden, Mälardalen University. 2005.

17 DUVALL, P. Continuous Integration, Improving Software Quality and Reducing Risk, 2007. [En línea]. Disponible en: http://www.addison-wesley.de/main/main.asp?page=englisch/bookdetails&ProductID=121548

18 LÓPEZ MARTÍNEZ, Patricia. Desarrollo de sistemas de tiempo real basados en componentes utilizando modelos de comportamiento reactivos. Tesis Doctoral, Santander, España: Universidad de Cantabria, junio 2010. 316p.

19 PRESSMAN, Roger S. Métodos convencionales para la ingeniería del software. En: Darrel Ince.

Ingeniería del software un enfoque práctico. México: 2001. Parte V, 129 p.

20 Computer information. Las ventajas de modelado basado en componente. [En línea]. Las ventajas del modelado basado en componentes. [Consultado el 16 de febrero del 2015]. Disponible en: http://ordenador.wingwit.com/software/engineering-software/128681.html

21 PIATTINI, Mario; GARZÁS, Javier. Conceptos y evolución de las fábricas de software. [En línea]. [Consultado el 19 de febrero del 2015]. Disponible en: http://www.kybeleconsulting.com/articulos/concepto-y-evolucion-de-las-fabricas-software/

22 MUÑOZ, Javier; PELECHANO, Vicente. MDA vs Factorías de Software. Valencia, 10 p.

23 SALINAS ALGUACIL, Leyvis. [En línea]. Modelo de fábricas de software basado en componentes, 2014. [Consultado el 20 de febrero del 2015].

24 LABORATORIO NACIONAL DE CALIDAD DEL SOFTWARE. Ingeniería del software: Metodologías y ciclos de vida. Marzo 2009.

25 SANTIAGO ZARAGOZA, María de Lourdes. Desarrollando aplicaciones informáticas con el Proceso de Desarrollo Unificado (RUP). [En línea]. [Consultado el 19 de marzo del 2015]. Disponible en: http://www.utvm.edu.mx/OrganoInformativo/orgJul07/RUP.htm

26 RATIONAL, C. S. (2003). Rational Unified Process.

27 WESLEY, Addison. Una explicación de la programación extrema. Aceptar el cambio, 2000.

28 GARCÍA OTERINO, Ana. Aprende a implantar integración continua desde cero (I): ¿Por qué integración continua?. [En línea] Aprende a implantar integración continua desde cero (I), 2014. [Consultado el 15 de febrero del 2015]. Disponible en: http://www.javiergarzas.com/2014/08/implantar-integracion-continua.html.

29 LÓPEZ TOREST, José Manuel. Integración Continua, la solución para afrontar con éxito entornos de

desarrollo complejos. [En línea]. [Consultado el 16 de febrero del 2015]. Disponible en: http://www.mtp.es/noticias/290-integracion-continua-la-solucion-para-afrontar-con-exito-entornos-de-desarrollo-complejos.

30 RAYMOND, Eric S. La catedral y el bazar. 2009. 32 p.

31 ALONSO SERRANO, Atenea, et al. Métodos de investigación con enfoque experimental. 32 p

32 MONARCA, Laura. Desarrollo de software basado en componentes. 2003. 70 p.

33 GOMEZ GOMEZ, Omar Salvador. Integración Continua en componentes EJB, 2008. [En línea]. [Consultado el 16 de marzo del 2015]. Disponible en: http://www.osgg.net/omarsite/resources/papers/ic_ejb.pdf.

34 CALAS TORRES, Abrahan. Modelo de componentes para el marco de trabajo Sauxe. Tesis de pregrado. Universidad de las Ciencias Informáticas. La Habana, 2014.

35 DARIAS CAMACHO, Yeray. Integración continua.[En línea]. [Consultado el 17 de febrero del 2015]. Disponible en: http://emanchado.github.io/camino-mejor-programador/html/ch08.html

36 TRUJILLO, Yaimí. Modelo de Factoría de Software aplicando inteligencia. Modelo de Factoría de Software aplicando inteligencia. Habana, 2007.

37 GONZÁLEZ SILVA, Yaritza; LOPEZ ZUVIETA, Alexis. Núcleo del entorno de escritorio de Nova Ligero 2015. Tesis de pregrado. Universidad de las Ciencias Informática. La Habana, 2014.

38 ALBALAT AGUILA, Miguel; RAMÍREZ PÉREZ, Barbarita. Solución para hacer de Guano un entorno de escritorio usable. Tesis de pregrado. Universidad de las Ciencias Informáticas. La Habana, 2009.

39 VOLA, Gabriel; BOLATTI, Pablo. Por qué estandarizar. [En línea]. Gamut. Gestión de procesos, 2011. [Consultado el 10 de noviembre del 2014]. Disponible en: http://www.gamut.com.ar/joomla/index.php option=com_content&view=article&id=107&Itemid=670.