Metodología para el Desarrollo Colaborativo de Software Libre

Versión 2

Cenditel, Diciembre 2013

Licencia de Uso

Colaboradores

Para la elaboración de esta segunda versión de la Metodología para el Desarrollo Colaborativo de Software Libre se contó con el apoyo de miembros del Equipo de Desarrollo de la Fundación Cenditel, lo cual permitió incluir en esta nueva versión aspectos fundamentales para una práctica de desarrollo, los cuales apuntan a mejorar la práctica y a promover el desarrollo colaborativo de software libre. Cabe destacar que el apoyo brindado se basa en la aplicación de la primera versión de la metodología en muchos de los proyectos de desarrollo de la Fundación, a partir de la cual surgieron importantes aspectos a considerar para la segunda versión de esta metodología.

Cabe destacar el apoyo brindado por los siguientes miembros del Equipo de Desarrollo de la Fundación:

- Solazver Solé.
- Víctor Bravo.
- Joger Quintero.
- Antonio Araujo.
- Dhionel Díaz.

Tabla de Contenido

1. Metodología para el Desarrollo Colaborativo de Software Libre

La propuesta metodológica que se presenta en este documento constituye una segunda versión de la Metodología para el Desarrollo Colaborativo de Software Libre publicada por la Fundación Cenditel en el año 2007. Esta segunda versión se inspira en el concepto de práctica virtuosa definido por MacIntyre en el libro "Tras la Virtud" (1985), en el cual una práctica se define como una actividad humana que se desarrolla en un ambiente armónico entre sus integrantes, mediante el cual se posibilita el trabajo en torno a la búsqueda de la mejora continua de los bienes y/o servicios producidos en la práctica, lo que implica un proceso de cultivo de conocimiento entre sus integrantes en el que se persigue mejorar la práctica, a fin de producir bienes y servicios que brinden el mayor bienestar posible a la sociedad a la cual éstos van dirigidos. En específico, del concepto de práctica se toman, para este segunda versión de la metodología, algunos aspectos relacionados al cultivo del conocimiento y al sentido social de la práctica.

En lo que se refiere al cultivo de conocimiento se busca promover un proceso de enseñanza-aprendizaje en torno a la práctica de desarrollo software, que permita la mejora de ésta en base a las experiencias adquiridas en el ámbito de desarrollo de software, así como en base a modelos y patrones de excelencia que la comunidad de practicantes en este ámbito han ido cultivando en el transcurrir del tiempo. Para ello, esta segunda versión de la metodología de desarrollo se fundamenta en:

- Propuestas de mejoras a la primera versión de la metodología de desarrollo obtenidas como resultado de la aplicación de ésta en proyectos de software de la Fundación Cenditel.

- Planteamiento de una segunda versión del Proceso de Aseguramiento de Calidad en el Desarrollo de Software Libre, publicado por la Fundación Cenditel en el año 2013. Este proceso tiene como objetivo mejorar la calidad tanto a nivel de la práctica de desarrollo de software como a nivel de las aplicaciones desarrolladas.

- Modelos o patrones de excelencia que se han ido cultivando en el tiempo dentro de la comunidad de desarrollo de software (tanto a nivel propietario como libre), los cuales permiten ejecutar de una mejor manera actividades dentro de la práctica de desarrollo, mejorando así esta práctica y, por ende, la calidad de las aplicaciones desarrolladas en ella.

En lo que respecta al sentido social de la práctica se tiene como propósito consolidar una práctica de desarrollo que no solo permita ofrecer a la sociedad aplicaciones de software que satisfagan criterios de calidad, sino que permita hacer énfasis en la documentación del software, a fin de facilitar procesos de apropiación que apunten no solo al uso del software sino también al mantenimiento y mejora de éste, por parte de usuarios e interesados. En este sentido, se busca posibilitar una práctica de desarrollo que contribuya al alcance de la soberanía e independencia tecnológica de la Nación.

Para orientar este segunda versión de la metodología hacia una práctica de cultivo de conocimiento que involucre no solo a los practicantes del desarrollo de software sino también a los usuarios e interesados en las aplicaciones desarrolladas, se plantea en los procesos que componen la metodología un conjunto

de recomendaciones a tener en cuenta al momento de llevar a cabo muchas de las actividades planteadas en dichos procesos, con las cuales se persigue indicar una mejor forma de realizar la práctica de desarrollo en base a modelos y patrones de excelencia, así como en base a experiencias ganadas dentro de la práctica de desarrollo que se da en la Fundación Cenditel.

En este segunda versión de la metodología se continua con la estructura de procesos planteada en la primera versión, la cual se compone de los siguientes procesos: Conceptualización de Proyectos de Software Libre, Administración de Proyectos de Software Libre y Construcción de Aplicaciones de Software Libre. Estos procesos así como algunas de sus relaciones se presentan a continuación.

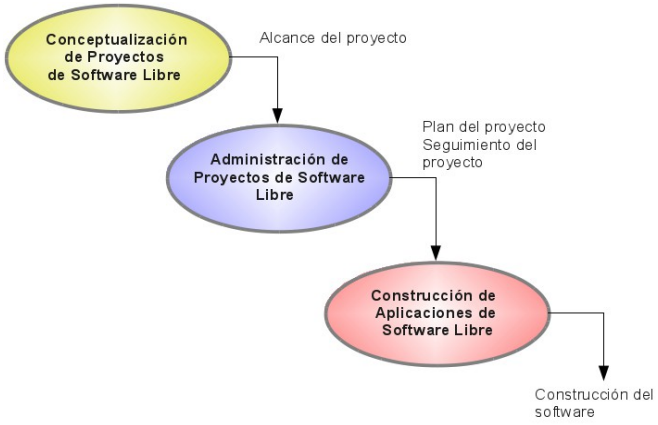

Figura 1. Relación entre los procesos que componen la metodología.

Tal como se muestra en la Figura 1, los tres procesos se retroalimentan entre sí a través de los principales productos que se generan en cada proceso. Por ejemplo, el alcance del proyecto generado en el proceso de conceptualización es suministrado al proceso de administración para generar allí el plan del proyecto en el que se indican las funcionalidades a desarrollar en el proceso de construcción.

Para agilizar y estandarizar la práctica de desarrollo en base a la segunda versión de la metodología, así como para apoyar las actividades de documentación del software planteadas en ésta, en la Fundación Cenditel se trabaja en una segunda versión del plugin de la plataforma *Trac* desarrollado en esta Fundación, en el año 2011, para apoyar la ejecución de los procesos de la metodología. Cabe destacar que este plugin junto a las demás herramientas que brinda la plataforma *Trac* permiten llevar a cabo la mayoría de las actividades que componen la práctica de desarrollo planteada en esta metodología. De esta manera, se facilita el uso de una plataforma robusta que concentra en ella un conjunto de herramientas y plantillas de documentación que sirven de apoyo a la ejecución de actividades referidas a la conceptualización, gestión y construcción de aplicaciones de software.

Es importante mencionar que tanto la plataforma *Trac* como el plugin desarrollado para ésta en la Fundación Cenditel se recomiendan en este documento como una más de las herramientas que existen para la gestión de proyectos, y, en el caso de la Fundación se promueve el uso de la misma como apoyo

en la ejecución de los procesos que componen la metodología de desarrollo, sin embargo, dicha metodología puede ser aplicada haciendo uso de cualquier otra plataforma que brinde servicios para la publicación del software y su documentación, así como para la gestión de errores, para la asignación y seguimiento de tareas en el Equipo de Desarrollo y para el control de versiones del código, entre otros. De igual manera, cabe destacar que la propuesta metodológica que se presenta no se encuentra atada a herramientas específicas de apoyo a la práctica de desarrollo, por lo que los procesos planteados en esta metodología pueden apoyarse en cualquier herramienta que facilite su ejecución y documentación.

1.1. Proceso de Conceptualización de Proyectos de Software Libre

En este proceso se recopila y analiza información concerniente a los procesos que se requieren automatizar en una aplicación de software, con el objetivo de comprender el dominio de la aplicación a desarrollar, así como los problemas o necesidades de los usuarios en relación a dichos procesos, todo ello con la finalidad de plantear una propuesta de desarrollo de software acorde a las necesidades de los usuarios.

A continuación en la Figura 2 se presenta el diagrama de flujo de trabajo correspondiente a la secuencia de ejecución de las actividades contempladas para el proceso de Conceptualización de Proyectos de Software Libre. Luego, en la Tabla 1 se describen las tareas correspondientes a cada una de las actividades indicadas en el flujo respectivo.

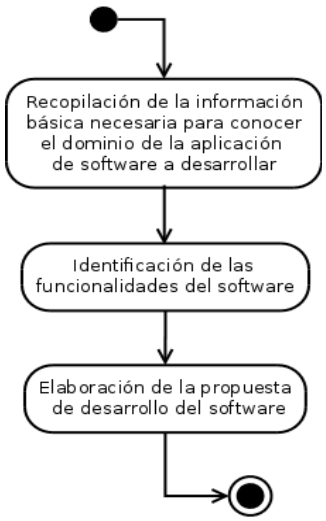

Figura 2. Flujo de trabajo del proceso de Conceptualización de Proyectos de Software Libre

Tabla 1. Tareas que integran las actividades asociadas al proceso de Conceptualización de Proyectos de Software Libre.

Actividad: Recopilación de la información básica necesaria para conocer el dominio de la aplicación de software a desarrollar	
Tarea: Identificar problemáticas y necesidades de los usuarios en torno a los procesos que se requieren automatizar.	*Recomendaciones*: • Para identificar los problemas y necesidades en torno a los procesos que se desean automatizar se requiere realizar reuniones entre miembros del Equipo de Desarrollo y los usuarios e interesados en el software a desarrollar. • En las conversaciones con los usuarios e interesados se debe recopilar información sobre los procesos a automatizar, en base a la cual se pueda modelar tales procesos con el objetivo de entender éstos en función de las actividades que los integran. • Es importante que en la información a recopilar sobre los procesos a automatizar se cuente con información que pueda aportar en la identificación de requerimientos no funcionales que deba cumplir el software a desarrollar, como por ejemplo, información respectiva al número aproximado de personas que usaran el software, así como su frecuencia de uso. Este tipo de información representa un insumo básico para definir el tipo de arquitectura del software.
	Herramientas: • Para los casos en los que se requiera realizar reuniones entre personas ubicadas en distintos espacios geográficos se pueden utilizar herramientas de comunicación como: *Hangout* (herramienta para videoconferencias de Google), *Skype*, entre otras. • Las minutas que se generen en las conversaciones con los usuarios y/o interesados pueden registrarse en la plataforma de desarrollo del proyecto.
	Productos: • Minutas.
	Responsables: Líder del Proyecto, Analistas.
	Colaboradores: Usuarios, interesados y cualquier miembro del Equipo de Desarrollo.
Actividad: Identificación de las funcionalidades del software	
Tarea: Elaborar los diagramas de procesos a	*Recomendaciones*: • Cada diagrama de proceso debe contener los elementos que describen el proceso, a saber: entradas, productos (salidas), recursos, reglas,

automatizar.	objetivos y actores. • Se pueden utilizar los diagramas caja negra para representar los procesos.
	Herramientas: • Para elaborar los diagramas de procesos se pueden utilizar herramientas gráficas como: *Dia*, *IDEFO, Bonita,* entre otras. • Los diagramas de procesos pueden registrarse en el wiki de "Análisis del dominio de la aplicación", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Diagramas de procesos.
	Responsable: Analistas.
	Colaboradores: Usuarios.
Tarea: Elaborar el diagrama de relación entre los procesos a automatizar.	*Recomendaciones*: • En este diagrama se debe representar la relación entre los procesos en términos de los productos que se generan en cada proceso y que se requieren como insumos (entradas y/o recursos) en otros procesos.
	Herramientas: • Para elaborar el diagrama de relación entre procesos se pueden utilizar herramientas gráficas como: *Dia, IDEFO, Bonita*, entre otras. • El diagrama de relación entre procesos puede registrarse en el wiki de "Análisis del dominio de la aplicación", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Diagrama de relación entre procesos.
	Responsable: Analistas.
	Colaboradores: Usuarios.
Tarea: Elaborar los diagramas de actividades correspondientes a cada proceso a automatizar.	*Recomendaciones*: • A fin de desarrollar un software que aporte mejoras en la ejecución de los procesos a automatizar, se recomienda analizar los diagramas de actividades de cada proceso, con el objetivo de identificar inconsistencias y conflictos en el flujo de ejecución de éstos, así como la necesidad de agregar o eliminar actividades en dichos procesos. De este análisis se pueden generar nuevos diagramas de actividades en los que se propongan mejoras en la ejecución de los procesos respectivos.
	Herramientas:

	• Para elaborar los diagramas de actividades se pueden utilizar herramientas gráficas como: *Dia, Umbrello, Bonita, CASEUML, ArgoUML, BOUML*, entre otras. • Los diagramas de actividades por proceso pueden registrarse en el wiki de "Análisis del dominio de la aplicación", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Productos: • Diagramas de actividades por proceso.
	Responsables: Analistas.
	Colaboradores: Usuarios.
Tarea: Validar con los usuarios los diagramas de procesos y de actividades.	*Producto*: • Diagramas de procesos validados. • Diagrama de relación entre procesos validado. • Diagramas de actividades validados.
	Responsable: Analistas y usuarios.
Tarea: Elaborar los diagramas de casos de uso en los que se represente el alcance del software en base a las funcionalidades generales del mismo.	*Recomendaciones*: • Las funcionalidades que debe brindar el software se identifican en base a los diagramas de procesos y actividades. • En los diagramas de casos de uso se deben representar tanto las funcionalidades del software como los usuarios que interactuaran con éstas. • Para facilitar la lectura de los diagramas de casos de uso se recomienda no superar mas de tres niveles de relación entre casos de uso (ya sean relaciones de inclusión, de extensión o de generalización).
	Herramientas: • Para elaborar los diagramas de casos de uso se pueden utilizar herramientas gráficas como: *Dia, Umbrello, Bonita, CASEUML, ArgoUML, BOUML*, entre otras. • En el caso de la Fundación Cenditel se plantea utilizar para elaborar los diagramas de casos de uso la herramienta *Platuml,* contenida en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Productos: • Diagramas de casos de uso (alcance del software).
	Responsable: Analistas.
	Colaboradores: Cualquier miembro del Equipo de Desarrollo.
Tarea: Identificar	*Recomendaciones*:

potenciales actores colaboradores en el desarrollo del software.	• A fin de fomentar el trabajo colaborativo en torno al desarrollo de los proyectos de software se considera pertinente identificar con que actores externos a la Fundación Cenditel se podría establecer dicho trabajo. Los actores colaboradores engloban tanto a los usuarios del software como actores que tengan conocimiento y experiencia en el desarrollo del tipo de software que se propone. En el caso de los usuarios es importante fomentar la participación de éstos en la práctica de desarrollo del software, pues de esta manera, el proceso de apropiación del software, que involucra no solo el planteamiento de requerimientos funcionales y el uso del software, podrá darse con mayor facilidad. • A partir de la identificación de los potenciales actores colaboradores se recomienda al Equipo de Desarrollo buscar un tipo de articulación que permita fomentar este trabajo colaborativo.
	Producto: • Potenciales actores colaboradores.
	Responsable: Líder del Proyecto.
	Colaboradores: Equipo de Desarrollo.
Actividad: Elaboración de la propuesta de desarrollo del software	
Tarea: Elaborar la propuesta de desarrollo.	*Recomendaciones*: • La propuesta de desarrollo debe contener información respectiva a la conceptualización del proyecto, por ejemplo, secciones de información referidas a: 1) Necesidades y/o problemáticas que se abordarán con el software a desarrollar; 2) Solución que se propone (tipo de software); 3) Alcance del software; 4) Descripción general de la arquitectura; 5) Potenciales actores colaboradores en el desarrollo del software; 6) Metodología de desarrollo; 7) Plataforma de operación; 8) Plataforma de desarrollo; 9) Licencias de código y documentación. • En la sección “Alcance del software” se recomienda mostrar los diagramas de casos de uso diseñados para representar las funciones generales del software, ello facilita la comprensión del alcance del mismo.
	Herramientas: • La información que debe contener esta propuesta puede registrarse en el wiki correspondiente a la “Propuesta de desarrollo”, contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Propuesta de desarrollo.

	Responsable: Líder del Proyecto.
	Colaboradores: Analista, Equipo de Desarrollo.

Para contar con una documentación del software que pueda servir de ayuda a los procesos de apropiación de éste, así como a la práctica de desarrollo, se requiere mantener actualizada la documentación generada en este proceso, así como la publicación de dicha documentación en la plataforma de desarrollo del proyecto y/o en el sitio web de éste.

1.2. Proceso de Administración de Proyectos de Software Libre

En este proceso se realizan actividades de planificación, coordinación y seguimiento de las tareas del Equipo de Desarrollo, así como un conjunto de actividades orientadas a facilitar la práctica de desarrollo colaborativo de software y la apropiación de éste.

Para describir las actividades y tareas que se contemplan en el proceso de Administración de Proyectos de Software Libre se utiliza la misma estructura presentada en el proceso anterior.

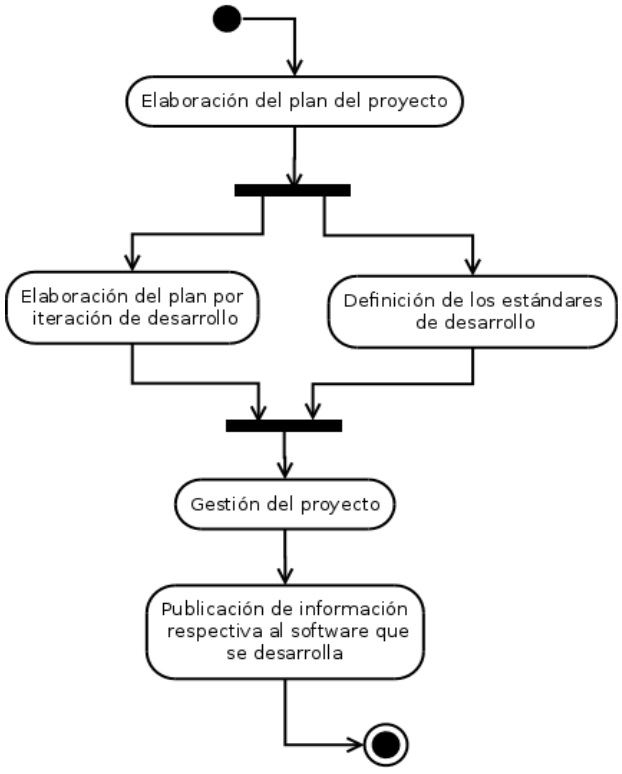

Figura 3. Flujo de trabajo del proceso de Administración de Proyectos de Software Libre

Tabla 2. Tareas que integran las actividades asociadas al proceso de Administración de Proyectos de Software Libre.

Actividad: Elaboración del plan del proyecto	
Tarea: Definir y priorizar las funcionalidades del software.	*Recomendaciones*: • Las funcionalidades generales del software se encuentran definidas en el alcance del mismo, sin embargo, al momento de priorizar las funcionalidades se recomienda estudiar con detalle las mismas. Dicha recomendación se plantea dado que en esta fase temprana del proyecto puede ser necesario definir nuevas funcionalidades o modificar las existentes. De ser así, es necesario actualizar en el documento de la propuesta de desarrollo el alcance del software. • A fin de poder establecer con mayor claridad el tiempo de desarrollo del software se recomienda desagregar aquellas funcionalidades que hallan sido definidas deforma general, es decir, aquellas funcionalidades que a su vez se compongan de otras. • A fin de liberar prototipos del software que sean útiles a los usuarios, conforme a la urgencia de sus necesidades, se recomienda que los usuarios indiquen la prioridad con la que requieren dichas funcionalidades.
	Herramienta: • Las funcionalidades del software junto a su priorización por parte de los usuarios pueden registrarse en el wiki correspondiente al "Plan del proyecto", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Priorización de funcionalidades por parte de los usuarios.
	Responsables: Usuarios y/o interesados, Líder del Proyecto.
	Colaboradores: Equipo de Desarrollo.
Tarea: Definir el orden de dependencia entre las funcionalidades del software.	*Recomendaciones*: • El establecer la dependencia entre funcionalidades, al igual que el priorizarlas las mismas, permite una mejor toma decisiones respecto a el orden en que deberían desarrollarse dichas funcionalidades, por lo cual estas tareas son fundamentales al momento de generar el plan del proyecto.
	Herramienta: • La dependencia entre funcionalidades del software puede registrarse en el wiki correspondiente al "Plan del proyecto", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.

	Producto: • Dependencia entre funcionalidades.
	Responsable: Líder del Proyecto.
	Colaboradores: Equipo de Desarrollo.
Tarea: Realizar un estudio sobre los riesgos de desarrollo del software.	*Recomendaciones*: • Por riesgo se entiende todo aquello que pueda interferir o dificultar el desarrollo de funcionalidades del software, como por ejemplo, la falta de conocimiento y experiencia en el tipo de lenguaje a utilizar para el desarrollo. Todo riesgo debe ir asociado a una o más funcionalidades. • El estudio de riesgos debe contener la definición y el impacto de los riesgos, así como la prioridad para abordar éstos y las acciones para prevenirlos. • La identificación de riesgos permite al Equipo de Desarrollo realizar una planificación concreta, ajustada a la dificultades que se puedan presentar durante el desarrollo del software, permitiendo así disminuir las diferencias entre lo planificado y lo ejecutado. • Los riesgos deben ser priorizados en términos de su impacto en el desarrollo de las funcionalidades del software y de su probabilidad de ocurrencia. • Es importante que se definan acciones preventivas que puedan llevarse a cabo en función de evitar la ocurrencia de los riesgos identificados. En el caso de riesgos en torno a complicaciones respecto al desarrollo de algunas funcionalidades o métodos del software, se recomienda como acción preventiva la puesta en práctica de "Pruebas de Concepto". Estas pruebas permiten implementar, de manera resumida e incompleta, un método, función o idea, con el propósito de verificar si es posible su desarrollo ("Prueba de concepto," 2013), así como determinar los aspectos a considerar durante el desarrollo del software relacionados a la implementación completa de dicho método, función o idea.
	Herramienta: • El estudio de riesgos puede registrarse en el wiki correspondiente al "Plan del proyecto", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Estudio de riesgos.
	Responsable: Líder del Proyecto.
	Colaboradores: Equipo de Desarrollo.

Tarea: Elaborar el plan del proyecto.	*Recomendaciones*: • La priorización de funcionalidades y riesgos, junto al orden de dependencia entre funcionalidades, constituyen los fundamentos sobre los cuales tomar decisiones en lo respectivo a la prioridad de desarrollo de las funcionalidades del software, por tanto, son la base para elaborar el plan de desarrollo de un proyecto de software. Esta manera de planificar permite ir desarrollando prototipos que sean útiles a los usuarios conforme sus necesidades, pero, teniendo en cuenta para ello el orden de dependencia entre las funcionalidades del software y la necesidad de abordar los riesgos más importantes del desarrollo en las etapas tempranas de éste. En base a la consideración de los tres fundamentos indicados en el párrafo anterior se ha planteado una formula para el cálculo de priorización de desarrollo de cada funcionalidad del software, la cual se especifica en el wiki del "Plan del proyecto" contenido en el plugin de la plataforma *Trac* desarrollado para la metodología. • El documento del plan del proyecto debe contener información respectiva a: 1) Priorización y orden de dependencia de las funcionalidades del software; 2) Estudio de riesgos (priorización y acciones preventivas); 3) Priorización de desarrollo de cada funcionalidad, 4) Cronograma de desarrollo del proyecto. • En el cronograma de desarrollo del proyecto se debe especificar el número de iteraciones a realizar, indicando por cada iteración las funcionalidades a desarrollar y las fechas de inicio y fin de cada iteración. La toma de decisiones sobre que funcionalidades desarrollar por iteración se realiza en base al calculo de la prioridad de desarrollo de cada funcionalidad.
	Herramienta: • La información que debe contener el plan del proyecto puede registrarse en el wiki correspondiente al "Plan del proyecto", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Productos: • Plan del proyecto.
	Responsable: Líder del Proyecto.
	Colaboradores: Equipo de Desarrollo.
Actividad: Definición de los estándares de desarrollo	
Tarea: Definir los estándares de	*Recomendaciones*: • Los estándares de desarrollo no solo abarca estándares de codificación,

desarrollo.	incluyen también estándares para interfaz gráfica. • Los estándares de codificación permiten la lectura rápida y simple del código, facilitando así el trabajo en conjunto, por lo cual la utilización de este tipo de estándares es fundamental tanto para el trabajo colaborativo como para los procesos de mejoras posteriores que se realicen al software.
	Herramienta: • Los estándares que se definan para el proyecto pueden registrarse en el wiki correspondiente a "Estándares de desarrollo", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Estándares de desarrollo.
	Responsable: Líder del Proyecto.
	Colaboradores: Equipo de Desarrollo.
Actividad: Elaboración del plan por iteración de desarrollo	
Tarea: Elaborar el plan para la iteración de desarrollo actual.	*Recomendaciones*: • En el plan de la iteración se indican las tareas a realizar para la construcción de las funcionalidades correspondientes a una iteración, según se plantea en el plan del proyecto. En este plan se deben indicar los responsables de cada tarea y el tiempo de entrega para los productos a obtener en las mismas. • Para cada una de las iteraciones indicadas en el plan del proyecto se debe generar una planificación de tareas.
	Herramientas: • Existen varias herramientas para la planificación de tareas, entre ellas: *Planner, XPTraker*, los sistemas de asignación de tareas incluidos en plataformas de desarrollo de software como *Gforge*, *SourceForge*, *Trac*, entre otras. • En el caso de la Fundación Cenditel se propone hacer uso del sistema de tickets que ofrece la plataforma *Trac*, a fin de utilizar el mismo para la definición y asignación de tareas respectivas al desarrollo de funcionalidades del software.
	Producto: • Plan de la iteración actual.
	Responsable: Líder del Proyecto.
	Colaboradores: Equipo de Desarrollo.

Actividad: Gestión del proyecto	
Tarea: Instalar una plataforma de desarrollo para gestionar el proyecto.	*Recomendaciones*: • Es importante utilizar una herramienta que apoye la gestión de la práctica de desarrollo, a través de la cual se puedan realizar actividades que faciliten dicha práctica, el desarrollo colaborativo y la apropiación del software. Entre las actividades a las que se hace referencia se encuentran: la publicación del software y su respectiva documentación, la gestión de errores, la asignación y seguimiento de tareas en el Equipo de Desarrollo, el control de versiones del código y su documentación, entre otras.
	Herramientas: • Existen varia plataformas de apoyo a la práctica de desarrollo de software, entre ellas: *XPTraker*, *Gforge*, *SourceForge*, *Trac*. La plataforma *Trac* es la herramienta utilizada en la Fundación Cenditel para apoyar la práctica de desarrollo de software. • En el caso de la herramienta *Trac* es importante destacar que la Fundación Cenditel ha desarrollado un plugin para dicha plataforma, el cual sirve de apoyo a los procesos planteados en esta metodología de desarrollo.
	Producto: • Plataforma de desarrollo del proyecto que contiene información respectiva a la práctica de desarrollo del software.
	Responsable: Líder del Proyecto.
	Colaboradores: Equipo de Desarrollo.
Tarea: Seleccionar un *framework* de desarrollo.	*Recomendaciones:* • Dado las bondades que ofrecen los *framework* de desarrollo se recomienda el uso de estas herramientas para facilitar y agilizar las tareas de codificación.
	Herramientas: • Eclipse. • Netbeans. • Aptana. • Otras.
	Producto: • *Framework* de desarrollo.
	Responsable: Líder del proyecto.
	Colaboradores: Equipo de Desarrollo.

Tarea: Realizar reuniones periódicas entre el Equipo de Desarrollo.	*Recomendaciones*: • Se sugiere tener reuniones semanales para discutir asuntos del proyecto y para verificar el avance del mismo. Es importantes que estas reuniones sirvan como medio para generar un proceso de enseñanza-aprendizaje entre los integrantes del Equipo de Desarrollo, pues en ellas se pueden exponer las dificultades que se presenten al equipo durante el desarrollo del software, así como las lecciones aprendidas al momento de abordar dichas dificultades. • A fin de llevar un registro de los asuntos y acuerdos discutidos en las reuniones periódicas del proyecto se sugiere elaborar minutas de cada reunión.
	Herramientas: • Las minutas que se generen pueden registrarse en la plataforma de desarrollo del proyecto. • Para los casos en los que se requiera realizar reuniones entre personas ubicadas en distintos espacios geográficos se pueden utilizar herramientas de comunicación como: *Hangout, Skype*, entre otras.
	Productos: • Minutas.
	Responsable: Líder del Proyecto.
	Colaboradores: Equipo de Desarrollo.
Tarea: Realizar seguimiento de las tareas asignadas al Equipo de Desarrollo.	*Recomendaciones*: • Se sugiere que el seguimiento de las tareas incluya, además de la revisión de los productos obtenidos como resultado de cada tarea realizada, la verificación del cumplimiento de estándares, lo cual constituye un factor muy importante para facilitar tanto el desarrollo colaborativo como las modificaciones o agregados que se realicen al software a futuro.
	Responsable: Líder del Proyecto.
Tarea: Gestionar los errores reportados sobre el software.	*Recomendaciones*: • Para realizar el seguimiento y control de los errores reportados y de la corrección de los mismo se recomienda contar con un sistema para gestión de errores (*Bug Tracking System*), que permita llevar el registro de dichos errores, así como de su priorización para ser atendido (la cual depende de la gravedad que representa el error y de la urgencia con la cual se requiere su solución). • Para el caso de los errores reportados por usuarios externos al equipo de desarrollo se recomienda contar con mecanismos sencillos para

	efectuar dichos reportes. Para una mejor organización del trabajo colaborativo en torno al desarrollo del software se sugiere colocar dichos mecanismos de reporte en el sitio web del proyecto. • Para facilitar la gestión en la corrección de los errores se recomienda que éstos sean reportados al Líder del Proyecto, quien debe gestionar los mismos para su respectiva corrección entre los miembros del Equipo de Desarrollo.
	Herramientas: • Existen muchas herramientas para la gestión de proyectos de software que incluyen sistemas para la gestión de errores, entre ellas se encuentran: *Trac*, *Redmine*, *Open Atrium*, *Proyect-Open*. • Entre las herramientas para gestión de errores se encuentran: *Bugzilla*, *Mantis*, *Request Tracker*, *Eventum*, entre otras. • En el caso de la Fundación Cenditel se plantea utilizar el sistema de *tickets* que incluye la plataforma *Trac* para realizar el seguimiento y control de errores en los proyectos que se desarrollan.
	Producto: • Seguimiento y control de los errores reportados.
	Responsable: Líder del Proyecto.
	Colaboradores: Equipo de Desarrollo.
Actividad: Construcción del sitio web del proyecto	
Tarea: Construir un sitio web para la publicación de información respectiva al proyecto.	*Recomendaciones*: • Para todo proyecto de software libre es fundamental contar con un sitio web de acceso público (página o repositorio), en el cual se pueda dar información sobre el proyecto que se lleva a cabo, así como acceso a las versiones del software y a cualquier otra documentación del mismo que se considere pertinente, tanto para facilitar la apropiación del software como para promover la colaboración en torno al desarrollo de éste. En relación a las versiones del software es muy importante la publicación de las versiones de prueba, pues ello facilita que personas ajenas al Equipo de Desarrollo puedan colaborar en la fase de pruebas del software, y reportar los errores conseguidos en el mismo. • Entre la información a publicar sobre el software es fundamental colocar información de contacto del Equipo de Desarrollo, y/o información sobre mecanismos de comunicación con este equipo, a fin de que éstos puedan ser contactados para facilitar información sobre el proyecto. De igual forma, es importante que a través del sitio de publicación del proyecto los usuarios del software puedan reportar

	errores sobre el mismo. • Se recomienda que en el sitio web del proyecto se coloque un enlace a la plataforma de desarrollo del software, a fin de que las personas interesadas en el proyecto puedan tener acceso a toda la documentación que se genera.
	Herramientas: • En el caso de la fundación Cenditel, por lo general, se utilizan los *blog* como sitios web de los proyectos de desarrollo y el repositorio *FusionForge*.
	Producto: • Sitio web del proyecto con información sobre éste.
	Responsables: Líder del Proyecto y miembros del Equipo de Desarrollo.

Para contar con una documentación del software que pueda servir de ayuda a los procesos de apropiación de éste, así como a la práctica de desarrollo, se requiere mantener actualizada la documentación generada en este proceso, así como la publicación de dicha documentación en la plataforma de desarrollo del proyecto y/o en el sitio web de éste.

1.3. Proceso de Construcción de Aplicaciones de Software Libre

Una vez elaborado el plan del proyecto se prosigue a llevar a cabo las iteraciones planificadas. En cada iteración se realizan un conjunto de actividades que conforman el proceso de Construcción de la Aplicación de Software Libre. Este conjunto de actividades se agrupan en las siguientes fases: Especificación de Requerimientos, Análisis y Diseño, Codificación, Pruebas y Liberación.

En la Figura 3 se presentan las fases que componen el proceso de construcción y las relaciones entre éstas.

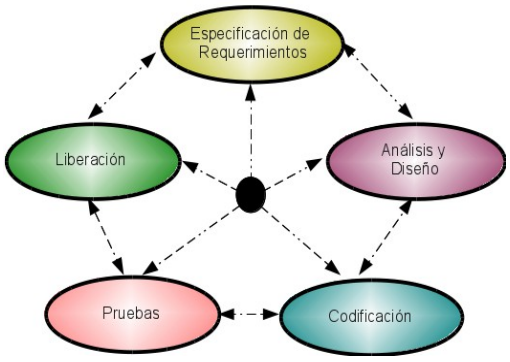

Figura 4. Gráfico de relación entre las fases que componen el proceso de Construcción de Aplicaciones de Software Libre

En la metodología la construcción de aplicaciones de software se da de manera incremental, de forma que en cada iteración[1] planificada se construye un número especifico de funcionalidades, a partir de las cuales se obtiene una versión o prototipo de prueba (versión beta) que se entrega a los usuarios para su validación. Los errores reportados por los usuarios son corregidos por el Equipo de Desarrollo, obteniendo así una versión estable del software. La construcción de una versión del software requiere llevar a cabo las fase indicadas en la Figura 4, o por lo menos la mayoría de éstas. Por lo general, una iteración de desarrollo comienza por la fase de Especificación de Requerimientos y culmina en la fase de Liberación.

En los proyectos de software los cambios en los requerimientos suelen ocurrir con frecuencia, así como las actualizaciones en su documentación, razón por la cual se plantea en la Figura 4 un proceso de construcción lo bastante flexible, en el cual se puede pasar de una fase a otra sin importar la secuencia entre éstas.

Cabe destacar que para contar con una documentación del software que pueda servir de ayuda a los procesos de apropiación de éste, así como a la práctica de desarrollo, se requiere mantener actualizada y publicada en la plataforma de desarrollo del proyecto y/o en el sitio web del mismo la documentación y el código generado en las fases del proceso de Construcción de Aplicaciones de Software Libre.

Las actividades que contempla cada una de las fases de este proceso se detallan a continuación. Para describir cada fase se utiliza un flujograma de trabajo en el que se indica la secuencia de ejecución de las actividades que componen la fase, y una tabla en la que se describen las tareas que conforman cada una de estas actividades.

1.3.1 Fase de Especificación de Requerimientos

En esta fase se especifican a detalle los requerimientos funcionales y no funcionales correspondientes a la versión del software a desarrollar en la iteración actual, de esta manera, la especificación de requerimientos irá evolucionando con el desarrollo de cada iteración.

La especificación de requerimientos funcionales corresponde a la descripción de los casos de uso del software, en los cuales se indica la interacción entre los usuarios y las funcionalidades del software, es decir, las acciones que pueden ejecutar los usuarios en el software y las respuestas de éste ante dichas acciones. Es por ello que la especificación de requerimientos funcionales representa el principal insumo para la fase de Análisis y Diseño, así como para las fases de Codificación y Pruebas.

Es importante resaltar que los casos de uso que se especificaran en esta fase corresponden a los casos de uso planteados en el proceso de Conceptualización de Proyectos de Software Libre para definir el alcance del software.

La especificación de requerimientos no funcionales corresponde a la definición de característica o

1 Las iteraciones pueden llevarse a cabo en paralelo siempre y cuando lo permitan las dependencias entre las funcionalidades a construir.

atributos de calidad del software, como por ejemplo:

- Tiempo de respuesta de las funciones de un software, referido éste al atributo *eficiencia* como característica de calidad, en específico a la subcaracterística *utilización de recursos del software*.
- Seguridad para el acceso a los datos que maneja un software, referido éste al atributo *funcionalidad* como característica de calidad, en específico a la subcaracterística *seguridad*.
- Intercambio de datos con otros software, referido éste al atributo *funcionalidad* como característica de calidad, en específico a la subcaracterística *interoperabilidad*.
- Manejo de fallas en el software, referido éste al atributo *confiabilidad* como característica de calidad, en específico a la subcaracterística *tolerancia a fallas.*
- Funcionamiento aceptable del software ante incrementos en el volumen de procesamiento. Éste corresponden al atributo *eficiencia* como característica de calidad, en específico a la subcaracterística *desempeño.*

En la Figura 5 se presenta el diagrama de flujo de trabajo correspondiente a la secuencia de ejecución de las actividades contempladas para la fase de Especificación de Requerimientos. Luego, en la Tabla 3 se describen las tareas correspondientes a cada una de las actividades indicadas en el flujo respectivo.

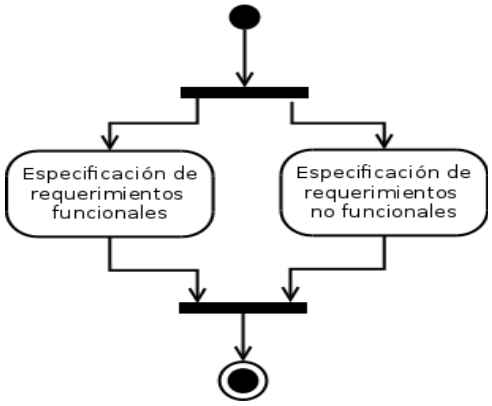

Figura 5. Flujo de trabajo de la fase de Especificación de Requerimientos

Tabla 3. Tareas que integran las actividades asociadas a la fase de Especificación de Requerimientos.

Actividad: Especificación de requerimientos funcionales	
Tarea: Describir textualmente los	*Recomendaciones*: • La descripción textual de un caso de uso se debe realizar en lenguaje natural, y la misma debe contener la siguiente información: a) Actores:

casos de uso correspondientes a la iteración actual.	usuarios y/o sistemas que interactúan con el caso de uso. b) Condiciones de entrada (precondiciones): condiciones que se deben cumplir para poder acceder al caso de uso. c) Condiciones de salida (postcondiciones): respuesta del software al ejecutarse exitosamente el caso de uso. d) Flujo básico: lista enumerada de los pasos que ejecuta el software y los actores para llevar a cabo el caso de uso, obteniéndose así la condición de salida. e) Flujos alternativos: pasos que ejecutan los actores y el software en situaciones no comunes, por ejemplo, ingreso de datos de entrada inválidos o la omisión de datos obligatorios, así como el comportamiento del software ante dicho ingreso u omisión de datos. f) Requisitos especiales: cualquier otra información que se considere pertinente para la construcción de la función a la cual se hace referencia en el caso de uso, por ejemplo, información referida a requerimientos no funcionales. • Los casos de uso no constituyen documentos de diseño de interfaz de usuario, por lo cual nunca debe hacerse referencia en ellos a elementos de la interfaz, tales como página principal, pantalla de ingreso o "clickear" botones ("Consejos para escribir," 2009). • La descripción del caso de uso debe ser hecha en forma detallada, clara y precisa, de manera que el Arquitecto de Software, el Diseñador Gráfico, los Programadores, los Probadores y los Documentadores no deban leer ningún otro documento para realizar su trabajo en relación con el software a construir (Peréz, s.f.). • Paro los requerimientos funcionales referidos a las operaciones CRUD (crear, obtener, actualizar y borrar) que mantengan un comportamiento similar en el software, cuyas operaciones sean cortas y simples, se recomienda especificar las misma en un solo caso de uso. Este planteamiento evita especificar por separado cada una de las operaciones CRUD, lo cual disminuye el número de casos de uso del software, y, por ende, el trabajo asociado a ello (Cuesta, 2007).
	Herramienta: • Los diagramas de casos de uso y la descripción textual de los mismos pueden registrarse en el wiki correspondiente a "Especificación de requerimientos funcionales", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Especificación de requerimientos funcionales. Este documento debe contener por cada requerimiento funcional: un diagrama de caso de uso (estos diagramas están contenidos en la propuesta de desarrollo del software) y su respectiva descripción textual.
	Responsables: Analistas.

	Colaboradores: Usuarios y/o interesados.
Tarea: Validar con los usuarios la descripción textual de los casos de uso asociados a la iteración actual.	*Producto*: • Especificación de requerimientos funcionales validada por los Usuarios.
	Responsables: Analistas y Usuarios.
Tarea: Discutir con el Equipo de Desarrollo la descripción textual de los casos de uso asociados a la iteración actual.	*Recomendaciones:* • Dado que la descripción textual de los requerimientos funcionales constituye el insumo principal que se utiliza para llevar a cabo las actividades de las fases de Análisis y Diseño, Codificación y Pruebas, se recomienda la discusión de esta descripción con los demás integrantes del Equipo de Desarrollo. Ello permite a su vez identificar posibles ambigüedades en dichas descripciones.
	Producto: • Especificación de requerimientos funcionales validada por el Equipo de Desarrollo.
	Responsables: Analistas y demás miembros del Equipo de Desarrollo.
Actividad: Especificación de requerimientos no funcionales	
Tarea: Definir con los usuarios los requerimientos no funcionales que debe cumplir el software.	*Recomendaciones:* • Los requerimientos no funcionales representan un insumo muy importante para la definición de la arquitectura del software, pues ellos implican aspectos que debe contemplar el software en términos de seguridad, tiempos de respuesta, interfaz de usuario, entre otros, que son determinantes para seleccionar una arquitectura que permita cumplir con tales requerimientos. • Durante el proceso de Conceptualización de Proyectos de Software Libre se levanta información que puede servir de insumo para la definición formal de requerimientos no funcionales. • Estos requerimientos pueden ser definidos en la primera iteración y pueden ser refinados en iteraciones posteriores.
	Herramienta: • La especificación de los requerimientos no funcionales puede registrarse en el wiki correspondiente a “Especificación de requerimientos no funcionales”, contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Especificación de requerimientos no funcionales.

	Responsables: Analistas y Usuarios.

1.3.2 Fase de Análisis y Diseño

Esta fase comprende la definición de la arquitectura del software, la especificación de los datos persistentes y el diseño de la interfaz de usuario. En el caso de la arquitectura es conveniente destacar que ésta constituye un aspecto a considerar, con mayor importancia, para aplicaciones de software que resulten complejas en términos del número de requerimientos y/o el impacto de los mismos (Hofmeister et al., 2000).

La arquitectura del software, la especificación de datos persistentes y el diseño de interfaz de usuario pueden construirse de forma incremental en cada iteración de desarrollo, con base a los requerimientos funcionales y no funcionales que se aborden en cada iteración.

Para describir las actividades y tareas que se contemplan en esta fase se utiliza la misma estructura presentada en la fase anterior.

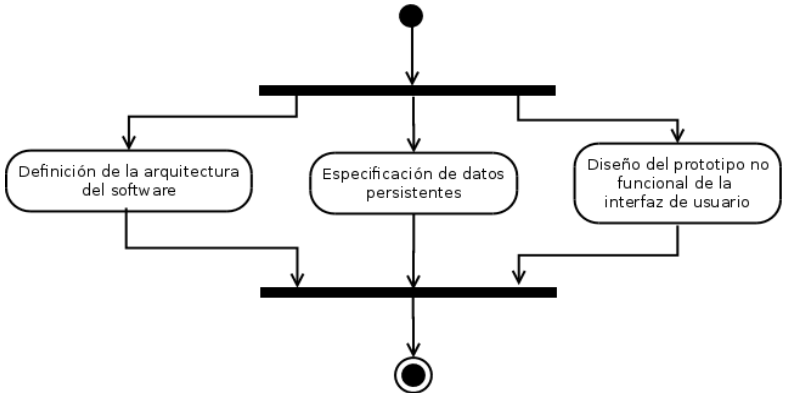

Figura 6. Flujo de trabajo de la fase de Análisis y Diseño

Tabla 5. Tareas que integran las actividades asociadas a la fase de Análisis y Diseño.

Actividad: Especificación de datos persistentes	
Tarea: Modelar los datos persistentes que maneja el software.	*Recomendaciones:* • Para modelar los datos persistentes se debe seleccionar el tipo de modelo en el que se realizará el almacén de datos. Entre los tipos de modelos mas utilizados se encuentran: orientado a objetos, entidad-relación, basado en documentos. Es posible realizar transformaciones entre estos modelos. • Los objetos o entidades que maneja el software se pueden identificar en la descripción textual de los casos de uso.

	• Los diagramas de clases elaborados como parte de la arquitectura del software constituyen en sí un modelo de datos, en caso de utilizar el modelo de datos orientado objetos. • En los casos en los que se trabaje con el modelo de datos relacional se requiere modelar los datos del software en base al modelo entidad-relación. • En la medida de lo posible, se recomienda utilizar tipos básicos para los campos de las relaciones, clases o documentos, de tal manera que la aplicación de software admita el uso de diversos gestores de datos, que se ajusten a las necesidades del despliegue. • Cuando se utilizan herramientas (editores gráficos) para modelar los datos es necesario tener en cuenta algunas pautas tales como configurar la salida de los *scripts* en SQL estándar, y mantener actualizado el modelo cuando ocurren cambios en los *scripts* de creación de la base de datos. • Se debe tener como premisa que la aplicación podrá cambiar de gestor de datos, incluso de modelo de datos, por lo tanto, es conveniente ceñirse a estándares y patrones de base de datos, como por ejemplo, los definidos en *SQL,* en el Modelo-Vista-Controlador o en *ORM* (*Object-Relational Mapping*).
	Herramientas: • Para elaborar diagramas de entidad-relación se pueden utilizar herramientas como: *DIA, ER Visual, BD Designer Fork* (principalmente para trabajar con base de datos *MySQUL*), *Druid*. • El modelo de datos del software puede registrarse en el wiki de "Modelo de datos persistentes", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Modelo de datos persistentes.
	Responsables: Analistas y/o el Arquitecto de software.
Tarea: Especificar los datos a intercambiar con otras aplicaciones de software.	*Recomendaciones:* • La interoperabilidad entre aplicaciones de software constituye un atributo de calidad a considerar en el desarrollo de aplicaciones que requieran el intercambio de datos con otras aplicaciones, razón por la cual es necesario especificar los datos a intercambiar, incluyendo en ello la definición de los formatos a utilizar para realizar dicho intercambio.
	Herramienta: • La especificación de datos a intercambiar puede registrarse en el wiki

	de "Modelo de datos persistentes", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Especificación de datos a intercambiar.
	Responsables: Analistas y/o Arquitecto de software.
	Colaboradores: Equipo de Desarrollo.
Actividad: Definición de la arquitectura del software	
Tarea: Definir la arquitectura del software.	*Recomendaciones*: • Al momento de definir la arquitectura del software, es decir, sus componentes[2] y la interacción (comunicación) entre éstos, se requiere tener en cuenta no solo las funcionalidades del software y las limitaciones tecnológicas existentes, sino también los atributos de calidad asociados al software, pues la arquitectura que se defina puede inhibir o facilitar el cumplimiento de dichos atributos (Bass et al., 1998). Por ejemplo, el atributo de calidad respectivo al desempeño del software depende de los componentes de se definan para éste y de su ubicación en los procesadores, así como de los caminos de comunicación entre estos componentes, etc. • La descripción textual de los casos de uso constituyen el insumo base para identificar los componentes del software. • Para el desarrollo de aplicaciones de software complejas y de gran escala se recomienda utilizar patrones y/o estilos arquitectónicos, así como patrones de diseño, pues éstos permiten mejorar la calidad de estas aplicaciones. Para la selección de patrones y estilos es necesario tener presente que éstos pueden facilitar el cumplimiento de ciertos atributos de calidad en un software, pero a su vez pueden inhibir el cumplimiento de otros atributos. Por ejemplo, el patrón arquitectónico Modelo-Vista-Controlador facilita el cumplimiento de atributos de calidad asociados a la funcionalidad y la mantenibilidad, pero dificulta el cumplimiento de atributos de calidad asociados al desempeño y a la portabilidad del software (Buschmann et al., 1996) . • Es importante que se estudien varias alternativas de arquitecturas, de modo que se pueda seleccionar de éstas aquella que permita cumplir en mayor medida con los requerimientos funcionales y los atributos de calidad establecidos para el software, que facilite futuras modificaciones al mismo y que sea factible conforme las limitaciones tecnológicas que puedan existir.

2 El término componente se utiliza en la metodología propuesta para hacer referencia a los elementos del software con comportamiento funcional definido, tales como: clases, módulos, funciones, métodos y piezas de código.

	• La arquitectura de un software puede ser representada a través de diferentes diagramas o vistas arquitectónicas, las cuales constituyen las diferentes perspectivas del diseño de una aplicación (Kruchten, 1999). Entre los diagramas utilizados para representar vistas arquitectónicas se encuentran: diagramas de clases, diagramas de secuencia, diagramas de estado, diagramas de componentes, entre otros. Cabe destacar que no es obligatorio diseñar todas las vistas arquitectónicas de un software, basta con plantear aquellas que se consideren pertinentes según la complejidad y alcance del software, que permitan contar con una visión de éste que sirva de apoyo en las diferentes fases de construcción del mismo, así como en sus procesos de mantenimiento.
	Herramientas: • Existen varias herramientas para elaborar los diagramas de la arquitectura de un software, entre ellas: *Dia, Umbrello, Bonita, CASEUML, ArgoUML, BOUML*, entre otras. • En el caso de la Fundación Cenditel se plantea utilizar para elaborar los diagramas de arquitectura de un software la herramienta *Platuml* contenida en el plugin de la plataforma *Trac* desarrollado para la metodología. Entre los diagramas que se pueden elaborar con esta herramienta se encuentran: diagramas de clases, de secuencia, de estado, entre otros. • Los diagramas en base a los cuales se represente la arquitectura del software pueden registrarse en el wiki de "Arquitectura del software", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Diagramas que representan la arquitectura del software.
	Responsables: Arquitecto de software.
	Colaboradores: Analistas, otros integrantes del Equipo de Desarrollo.
Actividad: Diseño del prototipo no funcional de la interfaz de usuario	
Tarea: Diseñar las pantallas no funcionales correspondientes a las interfaces gráficas del software.	*Recomendaciones:* • Por lo general toda interfaz gráfica se compone de los siguientes elementos (Baéz, Castañeda, Castañeda, 2005): a) Encabezado, éste se refiere a un logo o imagen de identificación del software. Se recomienda utilizar *frames* para que el encabezado se cargue solo una vez en las pantallas de interfaz del software. b) Menú, en éste se muestra el listado de las funcionalidades que ofrece el software, se recomienda que el menú se ubique, de ser posible, en varios lugares de

	las pantallas de interfaz. c) Zona de contenido, esta muestra las operaciones que puede ejecutar el usuario conformes a las funcionalidades que ofrece el software. d) Zona de mensajes, en esta se muestran los mensajes de tipo informativo, de error y de éxito en una operación del software. • Se recomienda utilizar un diagrama de navegabilidad entre las pantallas diseñadas para representar la interfaz de usuario, con lo cual se podrá mostrar el paso de una pantalla a otra. • Teniendo en cuenta la importancia que adquiere la interfaz de usuario, en términos de usabilidad, se plantean a continuación algunas recomendaciones a considerar para el diseño de la interfaz gráfica: a) La interfaz de las operaciones que ejecuta el software debe mantener un estándar visual. Por ejemplo, de ser posible, las funciones para modificar o eliminar información deben seguir un mismo esquema de presentación para las distintas funciones u operaciones del software en las cuales se requieran éstas. b) La estructura de iconos o botones que sirven de enlace a las funciones que ejecuta el software debe ser lo más sencilla posible, es decir, se debe evitar la ejecución de varios pasos a efectuar en el software para acceder a alguna de sus funciones. c) La interfaz debe mantener una estandarización en relación al formato de los iconos o botones mostrados. d) Los botones o iconos utilizados en la interfaz pueden mostrar textos en los cuales se indique que función cumple cada uno de éstos. Para ello se recomienda hacer uso de los *tool tips*. e) Los tipos y tamaños de las letras utilizadas en las pantallas deben facilitar la visualización de los textos o frases que se presentan en la interfaz. f) Los colores utilizados en cada pantalla deben ser contrastantes entre sí, a fin de facilitar la lectura de la información mostrada en la interfaz . g) La interfaz debe mantener una estandarización en relación al idioma de las personas que usarán el software.
	Herramientas: • La herramienta *Pencil* se puede utilizar para elaborar las pantallas del prototipo no funcional de la interfaz de usuario. • Este prototipo puede registrarse en el wiki “Prototipo no funcional de la interfaz de usuario”, contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Prototipo no funcional de la interfaz de usuario.
	Responsables: Diseñador Gráfico.

	Colaboradores: Equipo de Desarrollo, Usuarios.
Tarea: Validar el prototipo no funcional de la interfaz con los usuarios.	*Producto*: • Prototipo no funcional de la interfaz validada por los Usuarios.
	Responsables: Diseñador Gráfico y Usuarios.

1.3.3. Fase de Codificación

En esta fase se codifican las funcionalidades de la aplicación de software correspondientes a la iteración actual, se construye la interfaz de usuario y la base de datos. De esta manera, en esta fase, con cada iteración de desarrollo, se obtiene una nueva versión de la aplicación, clasificada como versión beta, es decir, una versión sobre la cual se deben realizar un conjunto de pruebas como pruebas funcionales y no funcionales.

Para describir las actividades y tareas que se contemplan en la fase de Codificación se utiliza la misma estructura presentada en la fase anterior.

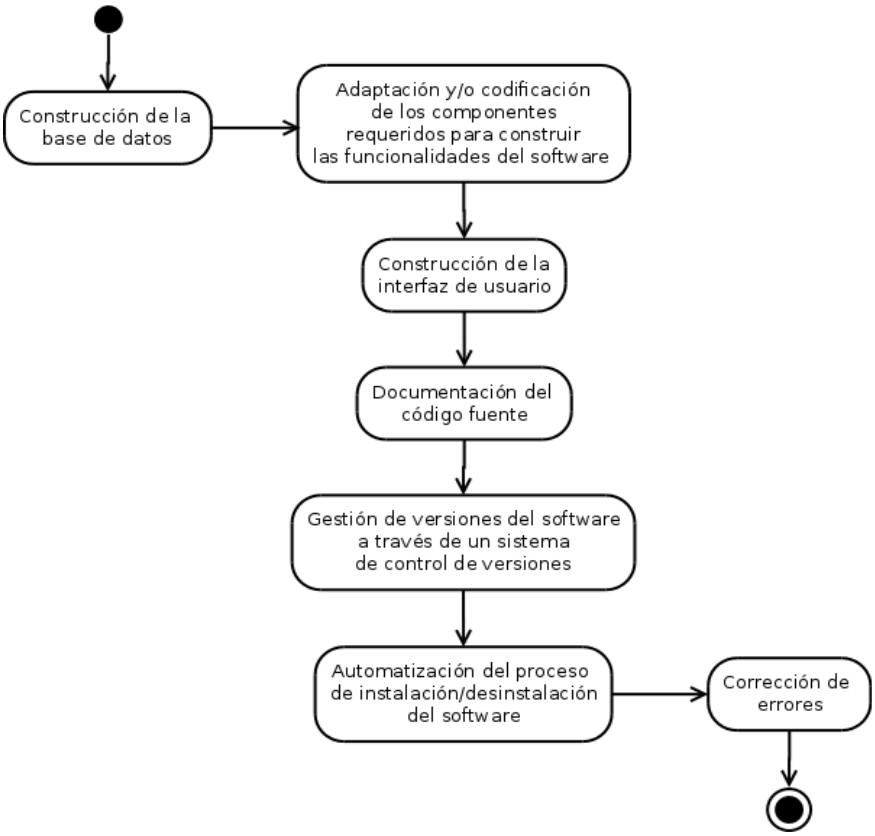

Figura 7. Flujo de trabajo de la fase de Codificación

Tabla 6. Tareas que integran las actividades asociadas a la fase de Codificación.

Actividad: Construcción de la base de datos	
Tarea: Construir la base de datos conforme al modelo de datos persistentes.	*Herramientas*: • Algunas herramientas que permiten generar los *scripts* para base de datos conforme al modelo de datos: *ER Visual, BD Designer Fork* (principalmente para trabajar con base de datos *MySQUL*).
	Producto: • Base de datos.
	Responsables: Programador.
	Colaboradores: Equipo de Desarrollo.
Actividad: Adaptación y/o codificación de los componentes requeridos para construir las funcionalidades del software	
Tarea: Codificar los componentes requeridos para construir las funcionalidades asociadas a la iteración actual.	*Recomendaciones:* • Para agilizar la fase de codificación y mejorar la calidad del software se recomienda el uso de *framework* de desarrollo, así como de patrones de programación y componentes reutilizables que puedan ser adaptados según se requiera. Entre los patrones de programación que se pueden utilizar se encuentran: patrón de acumulación, patrón lectura de datos, patrón de conteo (Mateu, s.f.); patrón *Super Loop*, patrón *Background / Foreground* (Alvarez, s.f.); entre otros.
	Producto: • Componentes implementados, componentes adaptados.
	Responsables: Programadores.
Tarea: Realizar las pruebas unitarias a los componentes adaptados y/o codificados y corregir los errores encontrados.	*Recomendaciones:* • A fin de evitar errores al momento de integrar los componentes del software se recomienda elaborar pruebas unitarias para verificar el funcionamiento, por separado, de cada uno de estos componentes. • Para agilizar las actividades asociadas a la aplicación de pruebas unitarias se recomienda la automatización de éstas.
	Herramientas: • Para aplicar pruebas unitarias se pueden utilizar herramientas como: *PhpUnit* (para lenguaje PHP), *Junit* (para lenguaje Java) , *xUnit* (para distintos lenguajes de programación), entre otras.
	Producto: • Errores corregidos.

	Responsables: Programadores.
Actividad: Construcción de la interfaz de usuario	

Tarea: Construir la interfaz de usuario correspondiente a las funcionalidades asociadas a la iteración actual.	*Herramientas*:
	• Para crear interfaces de usuario se pueden utilizar herramientas como:
	  • *TkInter, wxPython, PyGTK, PyQt,* estas herramientas permiten crear interfaces gráficas en *Python.*
	  • *Codeblocks, CodeLite, Gtkmm,* entre otras.

	Producto: • Interfaz gráfica del software.
	Responsables: Diseñador Gráfico, Programadores.
Actividad: Documentación del código fuente	
Tarea: Documentar el código fuente.	*Recomendaciones:* • La documentación del código fuente representa una actividad fundamental para facilitar los procesos de apropiación del software (mantenibilidad), por lo cual se recomienda realizar la documentación de todos los componentes de éste, tanto los codificados como los reutilizados. • Se sugiere que la documentación de cada componente (función, método, clase), como mínimo, haga referencia a qué hace el componente, qué parámetros hay que pasarle y qué devuelve (Fernández, 2008).
	Herramientas: • Para generar la documentación del código fuente se pueden utilizar herramientas como: *Doxygen*, *phpDocumentor,* entre otras.
	Producto: • Código documentado.
	Responsables: Programadores.
Actividad: Gestión de las versiones del software a través de un sistema de control de versiones	
Tarea: Colocar el código fuente desarrollado en cada iteración en el sistema de control de versiones que se utilice en el proyecto.	*Recomendaciones:* • El uso de sistemas de control de versiones es muy importante para el manejo organizado de las distintas versiones que se desarrollan para un software, pues éstos permiten gestionar los diversos cambios que se realizan sobre estas versiones, facilitando así el trabajo colaborativo y cooperativo en torno al desarrollo de software.
	Herramientas: • Existen muchas herramientas para el control de versiones de software,

	entre ellas tenemos: *CVS, SVK, Subversion*, *SourceSafe*, *Mercurial*, *Bazaar*, *GIT*, *Darcs*. • La mayoría de las plataformas para gestión de proyectos de software contienen sistemas para el control de versiones, tal es el caso de la plataforma *Trac* que se utiliza en la Fundación Cenditel.
	Producto: • Código fuente del software contenido en el sistema de control de versiones.
	Responsables: Programadores.
Actividad: Automatización del proceso de instalación/desinstalación del software	
Tarea: Codificar los pasos requeridos para instalar y desinstalar el software desarrollado.	*Recomendaciones*: • Es importante que durante el proceso de instalación se permita al usuario suspender dicho proceso, y que se muestren a éste mensajes de información sobre el éxito o no de la instalación. De igual manera, para facilitar el uso del software se recomienda que durante el proceso de instalación se creen los iconos para el acceso directo a éste.
	Producto: • Proceso de instalación/desinstalación automatizado.
	Responsables: Programadores.
Actividad: Corrección de errores	
Tarea: Corregir los errores reportados.	*Recomendaciones*: • Al utilizar un sistema de control de errores cada Programador puede tener acceso a los errores que se le asignen para su respectiva corrección, así como también podrán acceder a la información donde se describe el error respectivo (reporte). • Una vez corregido el error se recomienda al Programador registrar en el sistema de control de errores la causa del error y la corrección realizada. Esto permite llevar un historial sobre formas de resolver errores, el cual puede ser utilidad para solventar en el futuro errores iguales o similares que se puedan presentar.
	Producto: • Código corregido. • Registro de correcciones en el sistema de control de errores.
	Responsable: Programadores.

1.3.4. Fase de Pruebas

En esta fase se elaboran y aplican pruebas funcionales y no-funcionales a cada versión del software, así como pruebas de regresión y de instalación, con lo cual se facilita la detección temprana de errores e/o incompatibilidades en el código. Estas pruebas deben ser elaboradas y aplicadas por probadores de software, quienes se recomiendan deben ser personas distintas a quienes codifican la aplicación.

Cada versión del software que se obtiene pasa por la fase de pruebas, de esta manera los planes de prueba evolucionan con cada iteración, pues en cada una de éstas se agregan casos de prueba para verificar el comportamiento de las versiones del software y el cumplimiento de atributos de calidad.

En esta fase no se elaboran pruebas de integración, dado que las pruebas funcionales permiten a su vez verificar la integración entre los componentes o módulos del software, razón por la cual no se considera necesario realizar las pruebas de integración, lo cual permite agilizar el proceso de desarrollo del software.

Para describir las actividades y tareas que se contemplan en la fase de Pruebas se utiliza la misma estructura presentada en la fase anterior.

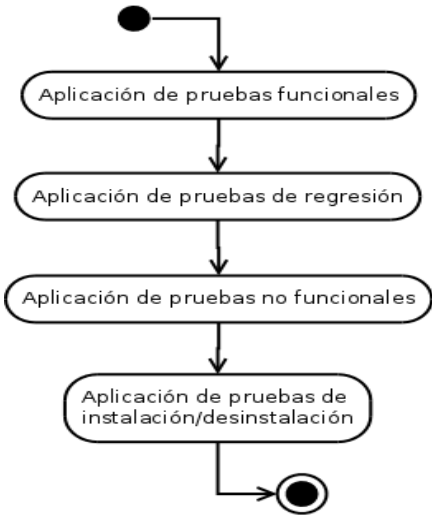

Figura 8. Flujo de trabajo de la fase de Pruebas

Tabla 7. Tareas que integran las actividades asociadas a la fase de Pruebas.

Actividad: Aplicación de pruebas funcionales	
Tarea: Elaborar el plan de pruebas funcionales correspondientes a	*Recomendaciones*: • Las pruebas funcionales se utilizan para verificar que el software ejecute sus funciones según lo establecido en la especificación de requerimientos funcionales. Por esta razón, la elaboración de las

las funcionalidades desarrolladas en la iteración actual.	pruebas funcionales se basa en la descripción textual de los casos de uso definidos para el software. • Un plan de pruebas funcionales se compone de casos de prueba con los cuales se verifica el comportamiento de las funcionalidades del software, en términos de los escenarios indicados en la descripción textual de los casos de uso asociados a dichas funcionalidades, es decir, en términos de flujos básicos, flujos alternativos y requisitos especiales. • Los datos utilizados en los casos de prueba deben ser de tipo válidos e inválidos, a fin de verificar el comportamiento del software ante estos tipos de datos. Por lo general, el comportamiento del software ante datos de entrada inválidos es especificado en los flujos alternativos de los casos de uso especificados para el software. • Un escenario de un caso de uso puede tener asociado varios casos de prueba, conforme a la cantidad y/o complejidad de los datos de entrada al software indicados en el caso de uso. • Para agilizar la elaboración y aplicación de las pruebas funcionales se sugiere, de ser posible, abarcar varios escenarios en un mismo caso de prueba, por ejemplo, se podrían abarcar varios flujos alternativos en un mismo caso de prueba. • Un caso de prueba debe contener: a) Escenario bajo el cual se realiza el caso de prueba, es decir, debe indicar si la prueba se realiza para el flujo básico, para flujos alternativos o para requisitos especiales del caso de uso respectivo. b) Objetivo del caso de prueba, en éste se indica el propósito que se persigue al ingresar u omitir datos de entrada al software para un escenario específico. c) Pre-condición, es un requisito que debe cumplir el software para poder ejecutar el caso de prueba, por lo general este requisito es la misma condición de entrada que se especifica en la descripción textual del caso de uso respectivo. d) Pasos para realizar el caso de prueba, acá se indican los pasos que debe llevar a cabo el probador en el software para ejecutar la prueba, por ejemplo: 1) Entrar a la función “Registrar proyecto”. 2) Ingresar los datos solicitados a excepción del título del proyecto, el cual debe dejarse en blanco. 3) Pulsar la opción “Guardar”. e) Salida esperada, es decir, el resultado que debe emitir el software, según el escenario de prueba, indicado en la descripción del caso de uso respectivo. f) Salida obtenida, acá se indica si el comportamiento del software al aplicar el caso de prueba respectivo es igual o diferente a la salida esperada. Una salida obtenida que sea diferente a la salida esperada se

	considera como un error o falla en el software. • Para agilizar el proceso de elaboración del plan de pruebas funcionales se recomienda no indicar en el mismo, de manera explicita, los datos a ingresar para ejecutar cada caso de prueba. En este sentido, para que el Probador tenga conocimiento acerca de los datos que debe ingresar al software, para ejecutar los casos de prueba, basta con mencionar en el plan el objetivo que se persigue con dichos casos de prueba.
	Herramienta: • Existen varias técnicas de prueba, en el caso de esta metodología se utiliza la técnica de diseño de pruebas Caja Negra (*black-box testing*) para elaborar casos de pruebas funcionales. • El plan de pruebas puede registrarse en el wiki "Plan de pruebas funcionales", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Plan de pruebas funcionales.
	Responsables: Probadores.
Tarea: Aplicar el plan de pruebas funcionales correspondientes a las funcionalidades desarrolladas en la iteración actual.	*Recomendaciones*: • En vista de que las pruebas funcionales pueden utilizarse como pruebas de regresión se recomienda utilizar herramientas que permitan grabar la ejecución de las pruebas funcionales, de modo que éstas puedan ser reproducidas posteriormente para verificar comportamiento del software después de haber modificado funcionalidades, o haber corregido errores. • Las pruebas funcionales al igual que las no funcionales pueden ser aplicadas en forma manual, de forma automatizada o apoyándose en el uso de herramientas de prueba. • La automatización de las pruebas es una actividad que requiere tiempo y esfuerzo, por lo cual ésta se recomienda siempre que se puedan reutilizar los casos de prueba automatizados, y/o en ocasiones en que resulte complicado ejecutar pruebas de forma manual, por ejemplo, cuando se requiere hacer pruebas no funcionales para verificar comportamiento y tiempos de respuesta del software cuando existen muchos usuarios conectados al mismo tiempo. En este caso, por lo general, el número de probadores con los que se puede contar no es suficiente para realizar estos tipos de prueba de forma manual, por lo que la automatización de dichas pruebas es la mejor opción (Oliveira y Gouveia, 2006). • Los errores que se encuentren al aplicar las pruebas funcionales deben

	reportarse al Líder de Proyecto, a fin de que éste gestione la corrección de los mismos. Para reportar un error se debe indicar el caso de prueba en el cual ocurre el error, así como una captura de la pantalla donde se indica el error en el software.
	Herramientas: • Existen diversas herramientas para aplicar pruebas funcionales, entre estas se encuentran: *Selenium* (permiten grabar las pruebas), *Jmeter, Watir* (para pruebas de aplicaciones web en Ruby), *SOLEX* (permiten grabar las pruebas), entre otras. • El reporte de errores se puede hacer por vía correo electrónico, a través de mecanismos de reporte o directamente a través de un sistema para gestión de errores.
	Producto: • Reporte de pruebas funcionales.
	Responsables: • Probadores.
Actividad: Aplicación de pruebas no funcionales	
Tarea: Elaborar el plan de pruebas no funcionales correspondientes a las funcionalidades desarrolladas en la iteración actual.	*Recomendaciones*: • Las pruebas no funcionales se utilizan para verificar en el software el cumplimiento de los atributos de calidad (requerimientos no-funcionales) establecidos para el mismo. Existen varios tipos de pruebas no funcionales, entre éstas se encuentran: pruebas de seguridad, pruebas de rendimiento, pruebas de usabilidad y pruebas de portabilidad. Entre las pruebas no funcionales mas aplicadas se encuentran las pruebas de rendimiento, con las cuales se estudia el comportamiento del software ante cargas máximas o mínimas de entrada de datos, de actividades o de almacenamiento. • Dado los diferentes tipos de pruebas no funcionales que se pueden aplicar, y teniendo en consideración la importancia que adquieren las pruebas de rendimiento para la gran mayoría de los software, en esta metodología solo se especificará, con cierto detalle, la elaboración de planes de pruebas de rendimiento. En este tipo de planes se indica el ambiente de prueba a utilizar y las variables del software a estudiar. En lo referente al ambiente de prueba se indican los recursos lógicos y físicos a utilizar en las pruebas. Los recursos lógicos se refieren a las herramientas para la realización de pruebas, por ejemplo, herramientas automatizadas. Los recursos físicos se corresponden a las características del equipo (hardware) a utilizar para realizar las pruebas, por ejemplo, tipo de computador y su velocidad, tipo de

	memoria, características de disco duro, etc. Las variables del software que se estudian con las pruebas de rendimiento constituyen atributos de calidad, como por ejemplo: tiempos de respuesta, capacidad de almacenamiento de datos, entre otras.
	Herramienta: • El plan de pruebas puede registrarse en el wiki "Plan de pruebas no funcionales", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Plan de pruebas no funcionales.
	Responsable: Probadores.
Tarea: Aplicar el plan de pruebas no funcionales correspondientes a las funcionalidades desarrolladas en la iteración actual.	*Recomendaciones*: • La aplicación de pruebas de rendimiento de forma manual es poco factible, dado que se requiere de una gran cantidad de probadores utilizando el software en un mismo período de tiempo, por lo cual se recomienda el uso de herramientas de prueba y/o la automatización de éstas. • Los resultados de las pruebas de rendimiento deben indicar cuando el software falla para determinados valores de las variables que se estudian en estas pruebas. Por ejemplo, en una prueba de rendimiento en la cual se estudia la variable "número de usuarios conectados al mismo tiempo", se debe poder indicar como resultado de la prueba el número de usuarios que soporta el software conectados al mismo tiempo antes de presentar fallas de conectividad o de operación. • Los responsables de aplicar las pruebas de rendimiento deben contar con conocimientos y habilidades en el área de programación, dado que estas pruebas implica la simulación de un número de usuarios interactuando con las funcionalidades del software. • Los errores del software que se encuentren al aplicar las pruebas no funcionales deben reportarse al Líder de Proyecto, a fin de que éste gestione la corrección de los mismos. Para reportar un error se debe mostrar una captura de la pantalla donde se indica el error en el software. • Posteriormente a la corrección de los errores que se reporten durante este tipo de pruebas se deben repetir éstas para verificar que el software opere correctamente.
	Herramientas: • Para aplicar pruebas de rendimiento se pueden utilizar herramientas

	como: *JMeter* (realizada en Java), *JCrawler*, *SOLEX* (también se puede usar para aplicar pruebas de regresión), entre otras. • Para aplicar pruebas de seguridad se pueden utilizar herramientas como: *Powerfuzzer, Nessus, Netcat, John the Ripper y OpenSSH*. • Para reportar los resultados de las pruebas no funcionales se pueden utilizar las herramientas planteadas para el reporte de errores en pruebas funcionales.
	Producto: • Reporte de pruebas no funcionales.
	Responsables: Probadores.
	Colaboradores: Programadores.
Actividad: Aplicación de pruebas de regresión	
Tarea: Aplicar pruebas de regresión a la versión del software obtenida en la iteración actual.	*Recomendaciones:* • Las pruebas de regresión se utilizan, generalmente, después de la corrección de errores o modificaciones en el código que puedan generar alteraciones considerables en el software, dado que al corregir errores o modificar el código se pueden introducir errores en el software que no existían antes de tales correcciones o modificaciones. En este sentido, el objetivo de las pruebas de regresión es verificar si se han introducido errores en el software después de la corrección de errores o de modificaciones al código, para lo cual se realizan pruebas de tipo funcional (Williams, Succi y Marchesi, 2003). En este sentido, las pruebas de regresión se definen en sí como pruebas funcionales, de allí la recomendación de utilizar herramientas que permitan grabar las pruebas funcionales aplicadas al software, con el fin de poder reproducirlas como pruebas de regresión. • Existen tres tipos de pruebas de regresión: pruebas para todas las funcionalidades del software, pruebas sólo para las funcionalidades que han sido modificadas y pruebas para las funcionalidades o piezas de código que se puedan ver afectadas por las modificaciones realizadas en el software. • Para reportar los errores encontrados en las pruebas de regresión se sugiere utilizar el mismo procedimiento indicado para el caso de reporte de errores en pruebas funcionales.
	Herramientas: • Para reportar los errores encontrados en las pruebas de regresión se pueden utilizar las herramientas planteadas para el reporte de errores en pruebas funcionales.
	Producto:

	• Reporte de pruebas de regresión.
	Responsables: Probadores.
Actividad: Aplicación de pruebas de instalación/desinstalación	
Tarea: Probar el proceso de instalación/ desinstalación de la aplicación de software en los hardware y software para los cuales ésta pueda operar.	*Recomendaciones:* • Las pruebas de instalación/desinstalación básicamente radican en ejecutar los pasos definidos para tales acciones en el manual de usuarios. Con estas pruebas se busca verificar que el proceso de instalación/desinstalación de la aplicación, ya sea éste manual o automatizado, no presente fallas. • Los errores que se encuentren en el proceso de instalación/desinstalación deben reportarse al Líder de Proyecto, a fin de que éste gestione la corrección de los mismos. Para reportar un error se debe presentar una captura de la pantalla donde se indica el error en el software. • Posteriormente a la corrección de los errores que se reporten durante este tipo de pruebas se deben repetir éstas para verificar que el proceso de instalación/desinstalación se lleve a cabo sin presentar fallas.
	Herramientas: • Para reportar las fallas del proceso de instalación/desinstalación se pueden utilizar las herramientas planteadas para el reporte de errores en pruebas funcionales.
	Producto: • Reporte de pruebas de instalación/desinstalación.
	Responsables: Probadores.

1.3.5. Fase de Liberación

En esta fase se llevan a cabo una serie de actividades asociadas a la liberación de versiones del software. Entre estas actividades se encuentran: elaboración de manuales, empaquetado del software, publicación de versiones beta (versiones de prueba) y publicación de versiones estables.

Para describir las actividades y tareas que se contemplan en la fase de Liberación se utiliza la misma estructura presentada en la fase anterior.

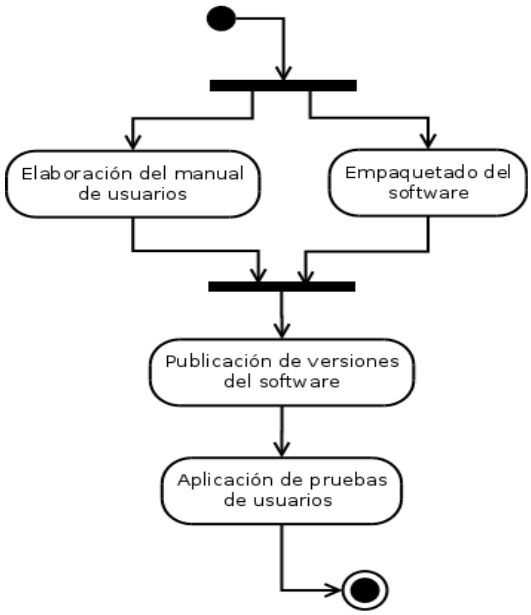

Figura 9. Flujo de trabajo de la fase de Liberación

Tabla 8. Tareas que integran las actividades asociadas a la fase de Liberación.

Actividad: Elaboración del manual de usuarios	
Tarea: Elaborar el manual de usuarios.	*Recomendaciones*: • A fin de que el manual de usuarios sea una herramienta de apoyo para el uso del software se sugiere que éste contenga la siguiente información: a) Índice. b) Descripción general del software y de sus funciones, así como de las tareas que pueden ser ejecutadas utilizando el software. c) Descripción de los símbolos y convenciones presentadas en la interfaz. d) Explicación del proceso de instalación/desinstalación del software. e) Explicación sobre el modo de uso de cada una de las funciones que componen el software, incluyendo imágenes o fotos de la interfaz que el software presenta para cada una de estas funciones. f) Preguntas frecuentes, acá se indican algunas de las preguntas más comunes que podría realizar el usuario con respecto al uso del software. g) Ejemplos para ayudar en la compresión de asuntos determinados. h) Explicación de los mensajes de error, cuando se considere necesario. En el caso del proceso de instalación/desinstalación del software se sugiere indicar la siguiente información: a) Tipo de procesador, memoria *RAM* y dispositivos de entrada y salida de datos (*Cd-Rom*, micrófono, teclado, escáner, *CD* y/o *DVD*, altavoces, auriculares, tarjeta de sonido, impresoras, etc.) requeridos para colocar el software en

	funcionamiento, así como el tamaño del dispositivo de almacenamiento que requiere el software. b) Sistemas operativos en los cuales funciona el software. c) Paquetes de software requeridos para su instalación en los sistemas operativos indicados. d) Información respectiva a la configuración de software en los sistemas operativos indicados. e) Pasos requeridos para instalar y desinstalar el software en los sistemas operativos indicados. f) Indicar si existe un procedimiento automatizado para la instalación y desinstalación del software. • El manual de usuarios puede realizarse de forma incremental con cada iteración de desarrollo, o puede ser elaborado al culminar de desarrollo de todas las funcionalidades del software.
	Herramienta: • Para elaborar manuales de usuarios se puede utilizar la herramienta *Sphinx.* • El manual puede registrarse en el wiki "Manual de Usuarios", contenido en el plugin de la plataforma *Trac* desarrollado para la metodología.
	Producto: • Manual de usuarios.
	Responsables: Documentador.
Tarea: Verificar correspondencia del manual de usuarios con el software desarrollado.	*Recomendaciones*: • La revisión del manual de usuarios debe realizarse a fin de verificar que éste se encuentre en correspondencia con el software desarrollado, así como para verificar que la información presentada en el manual sea clara, precisa y de fácil comprensión para los usuarios.
	Producto: • Observaciones sobre las revisiones realizadas al manual de usuarios.
	Responsables: • Analistas o cualquier miembro del Equipo de desarrollo.
Actividad: Empaquetado del software	
Tarea: Empaquetar el software para diferentes distribuciones de software.	*Recomendaciones*: • A fin de que el software desarrollado pueda ser portable, es decir, operar en varias distribuciones, se recomienda empaquetar el software para más de una distribución. • El empaquetar el software facilita en gran medida el proceso de instalación de éste. • Cada distribución tiene herramientas para empaquetado de software.

	Producto: • Software empaquetado para varias distribuciones.
	Responsables: Programadores.
Actividad: Publicación de versiones del software	
Tarea: Publicar la versión beta del software correspondiente a la iteración actual.	*Recomendaciones:* • La publicación de versiones beta es de fundamental importancia para el proceso de aseguramiento de calidad de la aplicación que se desarrolla, puesto que esta publicación se realiza con la finalidad de aumentar el número de personas que prueben el software, lo cual aumenta la posibilidad de encontrar errores, permitiendo así mejorar la calidad de la aplicación que se desarrolla. • Con la publicación de versiones beta no solo se busca que los usuarios prueben la aplicación a fin de reportar errores, sino también que éstos puedan plantear observaciones respecto a la interfaz de la aplicación.
	Producto: • Versión beta publicada.
	Responsables: Líder del proyecto y Equipo de desarrollo.
Tarea: Publicar la versión estable del software correspondiente a la iteración actual.	*Recomendaciones:* • Las versiones estables se publican una vez se hayan corregido los errores reportados por los usuarios en las versiones beta. • Con la intención de facilitar el proceso de apropiación del software la publicación de la versiones estables debe incluir no solo la publicación del código y los paquetes respectivos, sino también el manual de usuarios y el resto de documentación asociada a la versión publicada.
	Producto: • Versión estable publicada.
	Responsables: Líder del proyecto y Equipo de desarrollo.
Actividad: Aplicación de pruebas de usuarios	
Tarea: Probar la versión beta del software correspondiente a la iteración actual.	*Recomendaciones:* • Las pruebas que realizan los usuarios son conocidas como pruebas de aceptación, en ellas los usuarios verifican el funcionamiento del software y pueden revisar la interfaz a fin de aprobar o desaprobar elementos de la misma. • Es importante que se le indique a los usuarios la herramienta que pueden utilizar para reportar los errores y/u observaciones respecto a la interfaz, así como indicarles que al reportar un error deben presentar una captura de la pantalla donde se indica el error en el software.

	Herramienta: • Para reportar errores los usuarios pueden utilizar la herramienta defina en el proyecto para dicha tarea.
	Producto: • Reporte de pruebas de usuarios.
	Responsables: Usuarios.

Referencias Bibliográficas

Abowd, G., Allen, R., & Garlan, D. (1995). Formalizing Style to Understand Descriptions of Software Architecture. Technical Report. The Software Engineering Institute, Carnegie Mellon University. CMU-CS-95-111. Obtenido el 19-11-2013 de: http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/able/ftp/styleformalism- tosem95/styleformalism-tosem95.pdf

Alvarez, R. (s.f.) .El "Código Espagueti" y Los Patrones Avanzados de Programación. Obtenido el 14-11-2013 de: http://www.tecbolivia.com/index.php/articulos-y-tutoriales-microcontroladores/12-el-qcodigo-espaguetiq-y-los-patrones-avanzados-de-programacion

Bass, L., Clements, P., & Kazman, R. (1998). Software Architecture in practice. Addison-Wesley.

Báez, A., Castañeda, C., Castañeda, D., (2005). Metodología para el diseño y desarrollo de Interfaces de Usuario . Obtenido el 26-11-2013 de: http://pegasus.javeriana.edu.co/~fwj2ee/descargas/metodologia%28v0.1%29.pdf

Bredemeyer, D., & Malan, R. (2002). The Visual Architecting Process. White Paper. Obtenido el 21-11-2013 de: http://www.bredemeyer.com/pdf_files/ VisualArchitectingProcess.PDF

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). The UML Modeling Language User Guide. Adisson-Wesley

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). Pattern – Oriented Software Architecture. A System of Patterns. John Wiley & Sons, Inglaterra.

Consejos para Escribir Buenos Casos de Uso. Traducción de Mónica Navarro – TIS 2009. Obtenido el 19-11-2013 de: http://www.slideshare.net/kaolong/consejos-para-escribir-buenos-casos-de-uso-10382823

Cuesta, S. (2007). Patrones de Casos de Uso. Obtenido el 02-11-2013 de: http://sg.com.mx/content/view/510)

Fernández, F. (2008). phpDocumentor. Obtenido el 14-11-2013 de: http://www.epsilon-eridani.com/cubic/ap/cubic.php/doc/phpDocumentor---documentacion-para-codigo-PHP-246.html

Hofmeister, C.; Nord, R.; Soni D. (2000). Applied Software Architecture. Addison Wesley.

Kruchten, P. (1999). The Rational Unified Process. Reading, MA: Addison Wesley Longman, Inc.

MacIntyre, A. (1985). After Virtue: A Study in Moral Theory, Duckworth and Co., London.

Mateu, L. (s.f.). Patrones de programación. Obtenido el 14-11-2013 de: http://users.dcc.uchile.cl/~lmateu/CC10A/Apuntes/patrones/

Oliveira, J., Gouveia, C. (2006). *A way of Improving Test Automation Cost-Effectiveness.* Artículo publicado en CAST'06, Indianápolis, EE.UU.

Pérez, A. (s.f.). Estructuración y Especificación de Casos de Uso. Obtenido el 02-11-2013 de: https://sites.google.com/site/alfonsoperezr/investigacion/estructuracin-y-especificacin-de-casos-de-uos

Prueba de concepto, Historial de uso, Ejemplos. 2013. Obtenido el 12-11-2013 de: http://centrodeartigos.com/articulos-enciclopedicos/article_93241.html)

Williams, M., Succi, G. y Marchesi, L. (2003). *Traditional and Agile Software Engineering*. Ch 8 - Black Box Testing. Ed. Addison-Wesley.