Metodología para Desarrollo Colaborativo de Software Libre

Versión 2

Cenditel, Agosto 2013

Licencia de Uso

Metodología de Desarrollo Colaborativo de Software Libre

La propuesta que en este documento se presenta constituye una segunda versión de la Metodología de Desarrollo de Software Libre propuesta por la fundación Cenditel. El objetivo de esta segunda versión es mejorar la práctica de desarrollo de software libre contribuyendo a potenciar aspectos determinantes para la apropiación del software dentro de dicha práctica.

Esta metodología esta inspirada en el método ágil *Extreme Programming (Beck, 2004), en The Rational Unified Process (Kruchten, 2000)* así como en patrones, modelos y normas de excelencia que orientan la práctica de desarrollo de software.

En la metodología se combinan características de los estilos de desarrollo Catedral (desarrollo propietario) y Bazar (desarrollo de software libre), en base a las cuales es posible definir procesos centralizados que permiten el desarrollo colaborativo y la liberación frecuente del código fuente. Para explicar en detalle estas características es necesario definir ambos estilos (Catedral y Bazar) de desarrollo. En el estilo Catedral el desarrollo de software esta dirigido de manera centralizada y el proceso de desarrollo esta restringido a un grupo de programadores, quienes trabajan fuertemente en la depuración del código con la finalidad de que los usuarios puedan ver menos errores en cada versión liberada. En contraposición, en el estilo Bazar, el desarrollo de software no es dirigido de manera centralizada, la construcción de la aplicación se realiza con la participación de una comunidad de interesados que libera frecuentemente cada versión desarrollada, con la finalidad de que otros puedan depurar el código (http://biblioweb.sindominio.net/telematica/catedral.html).

Como se observa en las definiciones, ambos estilos de desarrollo presentan características opuestas. Para el caso específico de esta metodología se requiere combinar ambos estilos (Catedral y Bazar), con la intención de desarrollar proyectos de software donde se contemple la dirección centralizada de los proceso involucrados en el desarrollo, la participación comunitaria en la construcción de aplicaciones y la liberación frecuente del código fuente.

La metodología planteada se basa en una estructura organizacional orientada a procesos específicos. Estos procesos son: Conceptualización de Proyectos de Software, Administración de Proyectos de Software y Desarrollo de Aplicaciones de Software. Estos procesos se describen a continuación.

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. El objetivo de este proceso es 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 se presenta en la Figura 1 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. Luego, en la Tabla 1 se describen las tareas correspondientes a cada una de las actividades indicadas en el flujo respectivo.

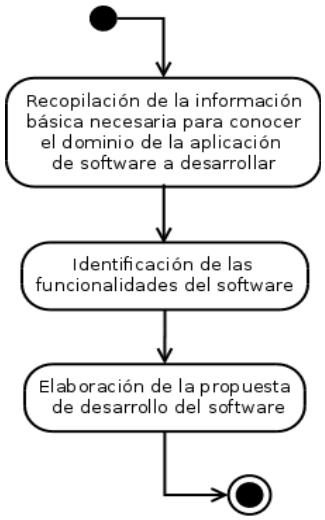

Figura 1. Flujo de Trabajo del proceso de Conceptualización de Proyectos de Software

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

Actividad: Recopilación de la información básica necesaria para conocer el dominio de la aplicación de software a desarrollar	
Tarea: Identificar problemáticas que serán abordadas con el software a desarrollar, así como los procesos automatizar.	*Recomendaciones*: • Para identificar los problemas que serán abordados con el software, así como los procesos que serán automatizados se requiere realizar reuniones entre miembros del Equipo de Desarrollo y los usuarios e interesados en el software. • 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 debe cumplir el software a desarrollar, como por ejemplo, información respectiva al número aproximado de personas que usaran el software para llevar a cabo los procesos a automatizar, 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*, *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, Analista.
	Colaboradores: Usuarios, interesados y cualquier miembro del Equipo de Desarrollo.
Actividad: Identificación de las funcionalidades del software	
Tarea: Elaborar un diagrama de proceso para cada uno de los procesos a automatizar.	*Recomendaciones*: • Cada diagrama debe contener los elementos que describen el proceso, a saber: entradas, productos (salidas), recursos, reglas, objetivos y actores. • Se pueden utilizar los diagramas caja negra para representar los elementos que describen cada proceso.
	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: Analista.
	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: Analista.
	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: Analista.
	Colaboradores: Usuarios y/o interesados.
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: Analista y usuarios.
Tarea: Elaborar los diagramas de casos de uso en los que se represente el alcance	*Recomendaciones*: • Es necesario conocer los procesos que se requieren automatizar y las relaciones entre éstos, a fin de definir los requerimientos funcionales del software que serán indicados en los diagramas de casos de uso.

del software en base a las funcionalidades generales del mismo.	• En los diagramas de casos de uso se deben representar tanto las funciones 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. Dicha herramienta se puede utilizar directamente en el wiki correspondiente al documento "Propuesta de Desarrollo", contenido en el plugin de la plataforma *Trac*.
	Productos: • Diagramas de casos de uso (alcance del software).
	Responsable: Analista.
	Colaboradores: Cualquier miembro del Equipo de Desarrollo.
Tarea: Identificar potenciales actores colaboradores en el desarrollo del software.	*Recomendaciones*: • 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 del software buscar un tipo de articulación que permita fomentar este trabajo colaborativo.
	Producto: • Potenciales actores colaboradores.
	Responsable: Líder del Proyecto.
	Colaboradores: Equipo de Desarrollo.

<table>
<tr><th colspan="2">Actividad: Elaboración de la propuesta de desarrollo del software</th></tr>
<tr><td rowspan="4">*Tarea*: Elaborar la propuesta de desarrollo.</td><td>*Recomendaciones*:
La propuesta de desarrollo debe contener información respectiva a la conceptualización del proyecto, por ejemplo, secciones de información referidas a: a) Necesidades y/o problemáticas que se abordarán con el software a desarrollar; b) Solución que se propone (tipo de software); c) Alcance del software; d) Descripción general de la arquitectura; e) Potenciales actores colaboradores en el desarrollo del software; f) Metodología de desarrollo; g) Plataforma de operación; h) Plataforma de desarrollo; i) 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 facilitará la comprensión del alcance del mismo.</td></tr>
<tr><td>*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.</td></tr>
<tr><td>*Producto*:
Propuesta de desarrollo.</td></tr>
<tr><td>*Responsable*: Líder del Proyecto.</td></tr>
<tr><td></td><td>*Colaboradores*: Analista, Desarrolladores y cualquier otro miembro del Equipo de Desarrollo.</td></tr>
</table>

2. Proceso de Administración de Proyectos de Software Libre

En el proceso de Administración de Proyectos de Software se realizan actividades de planificación, coordinación y seguimientos de las tareas del Equipo de Desarrollo, así como un conjunto de actividades orientadas a facilitar la práctica de desarrollo de software y la apropiación de éste, como también facilitar el desarrollo colaborativo.

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

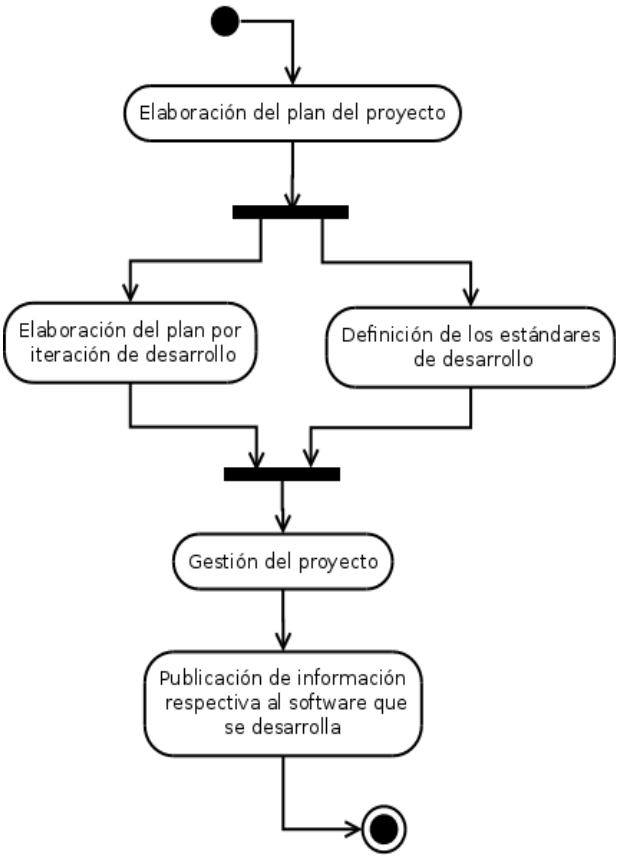

Figura 2. Flujo de Trabajo del proceso de Administración de Proyectos de Software

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

Actividad: Elaboración del plan del proyecto	
Tarea: Definir y	*Recomendaciones*:

priorizar las funcionalidades del software.	• 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, aun después de haber establecido el alcance del software. 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, en otras funcionalidades, 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 se puedan llevar a cabo en función de evitar la ocurrencia de los riesgos identificados. En el caso de riesgos entorno 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 (http://es.wikipedia.org/wiki/Prueba_de_concepto), 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: a) Priorización y orden de dependencia de las funcionalidades del software; b) Estudio de riesgos (priorización y acciones preventivas); c) Priorización de desarrollo de cada funcionalidad (se utiliza la formula indicada en el wiki del "Plan del proyecto" para calcular dicha priorización), d) Cronograma de desarrollo del proyecto. • En el cronograma de desarrollo del proyecto se debe indicar 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 desarrollo.	*Recomendaciones*: • En relación a los estándares de desarrollo es importante tener presente que existe una cantidad de estándares publicados que pueden ser

	reutilizados en el proyecto, a los cuales se les podrían hacer algunas adecuaciones según se considere pertinente. • Los estándares de desarrollo no solo abarca estándares de codificación, 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 la iteración actual, según se plantea en el plan del proyecto. En el plan por iteración 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 acciones.
	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 utilizar el 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 fundamentales para facilitar 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*, entre otras. 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 a utilizar.	*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.
	Producto: • *Framework* de desarrollo
	Responsable: Líder del proyecto.

	Colaboradores: Equipo de desarrollo.
Tarea: Realizar reuniones periódicas entre el equipo de desarrollo del proyecto.	*Recomendaciones*: • Se recomienda tener reuniones semanales para discutir asuntos del proyecto y para verificar el avance del mismo. • 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 *Trac*. • 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 un seguimiento de las tareas asignadas al equipo de desarrollo.	*Recomendaciones*: • Se sugiere que el seguimiento de las tareas que realiza el equipo de desarrollo 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 llevar un 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 asignar a miembros del Equipo de Desarrollo la corrección de los mismos.
	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: Publicación de información respectiva al software que se desarrolla	
Tarea: Publicar las versiones del software y su documentación.	*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 colocar en el sitio de publicación algún acceso a la plataforma de desarrollo del software, a fin de que las personas que visiten el sitio puedan acceder a dicha plataforma, en caso de requerir

	mayor información sobre el proyecto.
	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 el proyecto.
	Responsables: Líder del Proyecto y miembros del Equipo de Desarrollo.

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. Este conjunto de actividades se agrupan en las siguientes fases: Especificación de Requerimientos, Análisis y Diseño, Codificación, Pruebas, Elaboración de Manuales y Liberación.

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

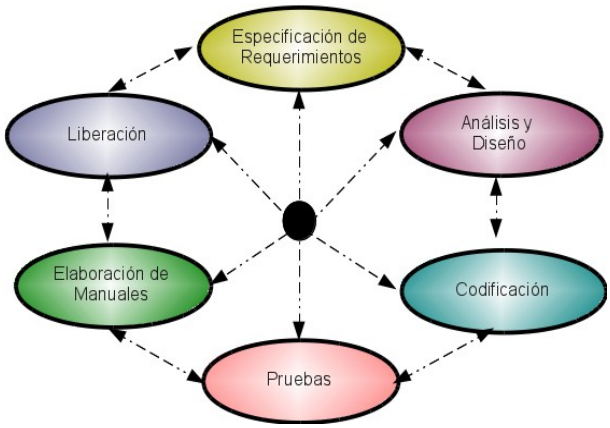

Figura 3. Gráfico de relación entre las fases que componen el proceso de Construcción de Aplicaciones de Software Libre

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 3, o por lo menos su mayoría. Por lo general, una iteración de desarrollo comienza por la fase de Especificación de Requerimientos y culmina en la fase de Elaboración de Manuales. En el caso de la fase de Empaquetado ésta debería ejecutarse una vez terminada la construcción y pruebas de las funcionalidades del software establecidas en el alcance del proyecto. Esta última fase incluye actividades referentes al empaquetado de la versión estable del software en diferentes distribuciones, con lo cual se busca la operabilidad de la aplicación en varios entornos o ambientes de software.

En los proyectos de software es común los cambios en los requerimientos, así como las actualizaciones en su documentación, razón por la cual se plantea en la Figura 3 un proceso de construcción lo bastante flexible, en el cual se puede pasar de una fase a otra sin importar la secuencia entre éstas.

Las actividades que contempla cada una de las fases mostradas en la Figura 3 se detallan a continuación. Para describir cada fase se utiliza un flujograma de trabajo que 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.

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.

En lo que respecta a la especificación de requerimientos funcionales cabe destacar que la actividad de elaboración de diagramas de casos de uso, la cual se realiza como parte de esta especificación, es llevada a cabo en el primer proceso de la metodología (Conceptualización de Proyectos de Software Libre), dado que es requerida allí para definir el alcance de la aplicación que se plantea en la propuesta de desarrollo. En este sentido, en esta fase, en lo que respecta a la especificación de

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

requerimientos funcionales, sólo se describirán las actividades referidas a la descripción textual de los casos de uso y a sus respectivas validaciones.

La especificación de requerimientos no funcionales corresponde a la definición de característica o atributos de calidad del software, como por ejemplo:

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

En la Figura 4 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.

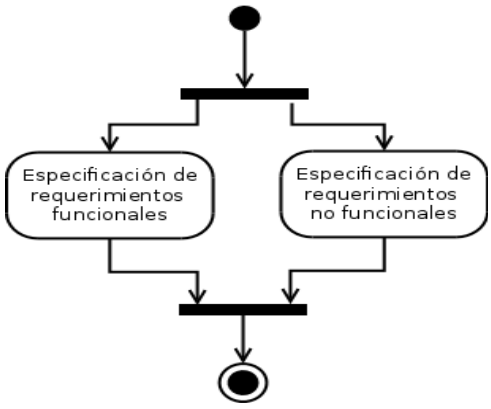

Figura 4. Flujo de Trabajo de la fase de Especificación de Requerimientos

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

Actividad: Especificación de requerimientos funcionales	
Tarea: Describir textualmente los casos de uso correspondientes a la iteración actual.	*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: 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 ejecutan 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/o el software en situaciones no comunes, por ejemplo, cuando ocurren errores en la interacción que se describe en el flujo básico. 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 son requerimientos funcionales, no 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 en un caso de uso (http://www.slideshare.net/kaolong/consejos-para-escribir-buenos-casos-de-uso-10382823). • La descripción del caso de uso debe ser hecha en forma detallada, clara y precisa, de manera que el arquitecto de software, el diseñador de interfaz, los programadores, los probadores y los documentadores no deban leer ningún otro documento para realizar su trabajo en relación con el software a construir (https://sites.google.com/site/alfonsoperezr/investigacion/estructuracin-y-especificacin-de-casos-de-uos). • 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, el cual se podría denominar "Gestionar ..." o "Administrar ...". 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 (http://sg.com.mx/content/view/510).

	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: Analista.
	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: Analista 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. Con ello, se busca explicar las descripciones realizadas de los casos de uso, a fin de apoyar la compresión de las mismas por parte del Equipo de Desarrollo, además de poder identificar ambigüedades que se puedan presentar en dichas descripciones.
	Producto: • Especificación de requerimientos funcionales validada por el Equipo de Desarrollo.
	Responsables: Analista y demás miembros del Equipo de Desarrollo.
Actividad: Especificación de requerimientos no funcionales	
Tarea: Definir con los usuarios los requerimientos no funcionales que debe	*Recomendaciones:* • Los requerimientos no funcionales representan un insumo muy importante para la definición de la arquitectura del software, pues en ellos se definen aspectos que debe contemplar el software en términos

cumplir el software.	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 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 las siguientes iteraciones.
	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: Analista y usuarios.

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 la misma constituye un aspecto que debe considerarse para procesos de desarrollo de 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 la fase de Análisis y Diseño se utilizará la misma estructura presentada en la fase anterior.

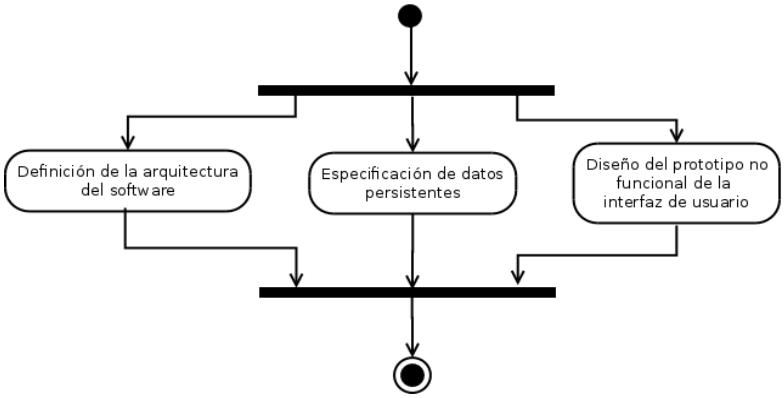

Figura 6. Flujo de Trabajo de la fase de Análisis y Diseño

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 de un software es necesario identificar los objetos o entidades que maneja el software a desarrollar, para lo cual se utiliza como insumo la descripción textual de los casos de uso. • Si el software se desarrolla bajo el enfoque de programación orientada a objetos los datos persistentes que éste maneja se modelan en los diagramas de clases elaborados como parte de la arquitectura del software. • 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.
	Herramientas: • Para elaborar diagramas de clases se pueden utilizar las mismas herramientas indicadas para representar la arquitectura del software. • 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, entre otras. • Los diagramas de entidad-relación pueden 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: Analista 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 muy importante a considerar en el desarrollo de aplicaciones que manejen datos de interés para otras aplicaciones, razón por la cual se requiere una especificación de los datos a intercambiar que incluya la identificación de dichos datos y la definición de los formatos a utilizar para realizar este 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: Analista 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 un patrón o estilo puede facilitar el cumplimiento de

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 funcional.

	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. • 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 de un software 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 los procesos de mantenimiento del software.
	Herramientas: • Existen varias herramientas para elaborar los diagramas a través de los cuales se pueden representar 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 correspondientes a la 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. Dicha herramienta se puede utilizar directamente en el wiki correspondiente al documento de “Arquitectura del software”, contenido en el plugin de la plataforma *Trac.*
	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 de la interfaz de usuario. • Teniendo en cuenta que el tema de la interfaz de usuario es fundamental para facilitar el uso del software, 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 en la interfaz 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 y entendible 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 para acceder a las funciones u operaciones del software 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. • El prototipo no funcional de la interfaz 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.

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 cada iteración de desarrollo en esta fase 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 utilizará la misma estructura presentada en la fase anterior.

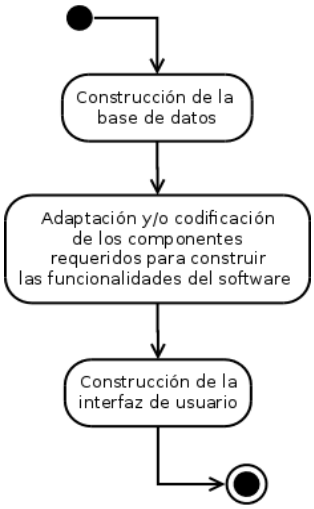

Figura 7. Flujo de Trabajo de la fase de Codificación

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	*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.

asociadas a la iteración actual.	Entre los patrones de programación que se pueden utilizar se encuentran: patrón de acumulación, patrón lectura de datos, patrón de conteo (http://users.dcc.uchile.cl/~lmateu/CC10A/Apuntes/patrones/); patrón *Super Loop*, patrón *Background / Foreground* (http://www.tecbolivia.com/index.php/articulos-y-tutoriales-microcontroladores/12-el-qcodigo-espaguetiq-y-los-patrones-avanzados-de-programacion); entre otros.
	Producto: • Componentes implementados, componentes adaptados.
	Responsables: Programadores.
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 recomienda 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 (http://www.epsilon-eridani.com/cubic/ap/cubic.php/doc/phpDocumentor---documentacion-para-codigo-PHP-246.html).
	Herramientas: • Para generar la documentación del código fuente se pueden utilizar herramientas como: *Doxygen*, *phpDocumentor,* entre otras.
	Producto: • Código documentado.
	Responsables: Programadores
Tarea: Realizar las pruebas unitarias a los componentes adaptados y/o codificados.	*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 estas pruebas.
	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.

	Responsables: Programadores.
Tarea: Colocar el código fuente desarrollado en cada iteración en el sistema de control de versiones que se utilice en el proyecto.	*Producto*: • Código fuente contenido en el sistema de control de versiones.
	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: Programador.

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

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

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

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

Addison-Wesley.

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

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

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

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