Centro Nacional de Desarrollo e Investigación en Tecnologías Libres (CENDITEL)
Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
Número 8. Año 5. ISSN: 2244-7423

Experiencia en Aseguramiento de Calidad en el Ámbito de Software Libre

Johanna Alvarez[1], Solazver Solé[2]
Fundación Centro Nacional de Desarrollo e Investigación en Tecnologías Libres (CENDITEL)[1]
Mérida, Venezuela
jalvarez@cenditel.gob.ve, ssole@cenditel.gob.ve

Resumen—En este artículo se tratan aspectos característicos de la práctica de desarrollo de software libre y sus productos, en función de los cuales se hace posible el ejercicio de las cuatro (4) libertades del software libre, y la generación de conocimiento a través de procesos colaborativos de enseñanza-aprendizaje. Por tales razones, se considera esencial la consideración de estos aspectos en el ámbito de aseguramiento de calidad de software libre. En este sentido, en el artículo se presenta una propuesta de evaluación de la calidad en práctica y producto de software, que nace como resultado de una experiencia en aseguramiento de calidad que se fundamenta en estándares, patrones, normas y modelos de calidad para el área de desarrollo de software.

Palabras Clave: calidad, desarrollo, software libre, estándares, patrones, modelos, evaluación.

1 Introducción

La experiencia que se presenta en el área de aseguramiento de calidad, desde el ámbito de software libre, busca consolidar un proceso de evaluación tanto para productos de software como para la práctica de desarrollo. Este proceso surge de una iniciativa planteada por la Fundación Centro Nacional de Desarrollo e Investigación de Tecnologías Libres (CENDITEL), a partir de la cual se plantean dos procesos de evaluación. El primer proceso basado en la norma ISO/IEC TR 9126-3 [1], dirigido a evaluar atributos de calidad en aplicaciones de software. El segundo proceso está orientado a evaluar la práctica de desarrollo, tomando como base para ello la metodología de desarrollo [2] propuesta por la Fundación.

Para determinar la pertinencia de los dos procesos señalados anteriormente, se procedió a la validación de ambos en dos (2) proyectos de desarrollo de software de CENDITEL. Durante la realización de esta validación surgieron algunas eventualidades que dificultan la puesta en práctica de muchos de los métodos de evaluación contemplados en estos procesos, debido a algunos aspectos de tipo organizacional presentes en la práctica de desarrollo de CENDITEL, así como también a la falta de adecuación de estos procesos de evaluación al contexto de desarrollo de software libre.

Las eventualidades descritas dan origen a la necesidad de adecuación y mejora de los procesos de evaluación mencionados, para lo cual se requiere mejoras a nivel de la metodología de desarrollo. Estas eventualidades y mejoras se describen a continuación como parte de la experiencia narrada.

2 Aseguramiento de Calidad en Aplicaciones de Software Libre Desarrolladas en la Fundación CENDITEL

Con el propósito de iniciar una línea de investigación en torno a la búsqueda de la ejecución de prácticas virtuosas en el desarrollo de tecnologías libres, en particular en el área de desarrollo de software, desde CENDITEL se propone un Modelo de Aseguramiento de Calidad en el Desarrollo de Software Libre [3], el cual se estructura en dos procesos: Evaluación del Proceso de Desarrollo de Software Libre y Evaluación de Calidad en Aplicaciones de Software Libre. Con ello, se persigue sentar las bases que contribuyan a la mejora continua de la práctica de desarrollo de software libre y de las aplicaciones que de ésta se deriven.

Para estudiar la pertinencia o aplicabilidad del Modelo de Aseguramiento de Calidad en el Desarrollo de Software Libre se plantea, durante el año 2012, la validación de éste evaluando dos (2) proyectos de software desarrollados en CENDITEL. La evaluación de Calidad en Aplicaciones de Software Libre se aplicó en el “Sistema de Gestión de Eventos en Línea”, conocido como SOFI. La evaluación del Proceso de Desarrollo de Software Libre se aplicó en el proyecto “Sistema Integral de Planificación Estratégico Situacional para la Administración Pública Nacional”, conocido como SIPES- APN.

Para llevar a cabo esta validación se organiza un equipo

de trabajo integrado por 4 (cuatro) personas con experiencia en el desarrollo de software libre, los cuales tuvieron como objetivo llevar a la práctica los métodos de evaluación descritos en el modelo propuesto.

2.1. Evaluación de Calidad en Aplicaciones de Software Libre

El proceso diseñado para evaluar atributos de calidad en las aplicaciones de software se basa en los métodos de evaluación de métricas descritos en la norma ISO/IEC TR 9126-3 [1], así como en algunas listas de chequeo para evaluar la calidad técnica de la documentación del software. Durante la validación de este proceso se generaron una serie de dificultades en torno a la ejecución de los mismos. Estas dificultades se indican a continuación:

- La aplicación SOFI no cuenta con la documentación técnica requerida para evaluar la mayoría de las métricas de funcionalidad, fiabilidad, eficiencia, mantenibilidad y portabilidad contempladas en el proceso. Para evaluar una aplicación de software en base a la norma ISO/IEC TR 9126-3, se requiere contar con la documentación técnica de ésta, en base a la cual se puedan identificar muchas de las variables contempladas en los métodos de evaluación de métricas, descritos en la respectiva norma. Por ejemplo, para evaluar la mayoría de las sub-métricas que integran las métricas de funcionalidad, mantenibilidad y portabilidad se necesita información contenida en la especificación de requerimientos y en los reportes de pruebas del software. Tal es el caso de las sub-métricas que se describen a continuación.

  Para evaluar la sub-métrica *integridad en la implementación funcional*, la cual corresponde al tipo de métrica funcionalidad, se establece el siguiente método:

  $$X=1- A/B$$

  donde la variable X representa la integridad en la implementación funcional, la variable A el número de funciones omitidas en el software con respecto al número de funciones descritas en la especificación de requerimientos, y la variable B el número de funciones descritas en la especificación de requerimientos. En este caso para obtener el valor de la variable B se requiere el documento de especificación de requerimientos.

  Para evaluar la sub-métrica *eliminación de fallas*, la cual corresponde al tipo de métrica fiabilidad, se establece el siguiente método:

  $$X=A/B$$

  donde la variable X representa la cantidad de fallas corregidas, la variable A el número de fallas corregidas en el diseño y/o codificación, y la variable B el número de fallas detectadas en las pruebas funcionales. En este caso para obtener el valor de la variable B se requiere el reporte de aplicación de pruebas funcionales.

  Cabe destacar que la mayoría de la aplicaciones desarrolladas en CENDITEL, hasta el momento de la validación del Modelo de Aseguramiento de Calidad en el Desarrollo de Software Libre, no contaban con el documento de especificación de requerimientos, pues de las nueve (9) aplicaciones que se han desarrollado sólo una tienen descritos sus casos de uso y estaba desactualizado. Esta falta de documentación técnica dificulta la aplicación de procesos de evaluación de calidad como los descritos en la norma ISO/IEC TR 9126-3. Según la opinión de la mayoría de los programadores que participaron en el desarrollo de estas aplicaciones, la falta de documentación técnica de las aplicaciones se debe a la carencia de personal dedicado a la documentación y pruebas de software.

- Para evaluar la mayoría de las métricas indicadas en la propuesta de evaluación de calidad en aplicaciones de software libre, se requiere llevar a cabo una serie de actividades que demandan tiempo y esfuerzo por parte de quienes tengan la responsabilidad de realizarlas. Esta responsabilidad en el área de la ingeniería de software es asignada a los denominados equipos de aseguramiento de calidad. En el caso de CENDITEL, la ejecución de actividades de aseguramiento de calidad se torna complicada, dado que en el área de desarrollo de software sólo se cuenta con un pequeño equipo de trabajo centrado en la programación de aplicaciones, y el resto de equipo de trabajo es multidisciplinario, pero ninguno de los perfiles se dedica a la documentación de software.

  Entre las distintas actividades que se requieren para llevar a cabo la evaluación de métricas de calidad se encuentran: revisión de documentación del software, elaboración de pruebas de usuarios, revisión del software, comparación entre el software y su documentación, entre otras. De allí,

la necesidad de contar con un equipo de aseguramiento de calidad encargado de estas actividades.

- No se contempla la evaluación de atributos asociados a características de calidad como mantenibilidad y portabilidad, entre ellos documentación y publicación del código fuente, empaquetado del software en diferentes distribuciones y su publicación en repositorios oficiales (como los repositorios de Debian, Ubuntu, entre otros), los cuales representan elementos de gran importancia en el ámbito de software libre. La importancia de estos atributos radica en su papel para garantizar las cuatro (4) libertades del software libre[1], puesto que un software documentado es mucho más fácil de entender y modificar por terceros, y si el mismo se encuentra publicado en repositorios oficiales, se facilita el acceso y uso de éste por parte de los usuarios e interesados en el mismo.

  Cabe destacar que los atributos mencionados contribuyen en el tránsito hacia un estadio más amplio que el de las cuatro (4) libertades del software libre, en tanto que, representan una condición de posibilidad de procesos de enseñanza-aprendizaje en la práctica de desarrollo. Estos procesos se nutren de los aportes de todas aquellas personas que colaboran en el desarrollo del software, pues al publicar los aportes se está generando la posibilidad de que otros aprendan del conocimiento publicado, y, a su vez, se genere conocimiento a partir de éste.

2.2. Evaluación de la Práctica de Desarrollo de Software Libre

La propuesta de Evaluación de la Práctica de Desarrollo de Software Libre toma como referencia la primera versión de la Metodología para el Desarrollo Colaborativo de Software Libre [2], planteada por CENDITEL. En específico, esta propuesta de evaluación se fundamenta en un conjunto de preguntas asociadas a cada una de las actividades que componen los procesos de la referida metodología. Con dichas preguntas se busca determinar cuándo las actividades que conforman la práctica o el proceso de desarrollo evaluado se realizan de forma deficiente o no satisfactoria, según lo establecido en la mencionada metodología.

Al validar esta propuesta de evaluación en la práctica de desarrollo del proyecto “Sistema Integral de Planificación Estratégico Situacional para la Administración Pública Nacional”, se presentaron una serie de dificultades que se exponen a continuación:

- La propuesta no contempla la evaluación de algunas actividades primordiales en toda práctica de desarrollo de software libre, en tanto que, tributan a las libertades de uso, copia, distribución, estudio y modificación del software. Entre las actividades mencionadas se encuentran, por ejemplo, aquellas respectivas al empaquetado del software en diferentes distribuciones, a la publicación en repositorios oficiales, al uso de sistemas de control de versiones, en entre otras.
- Tampoco evalúa actividades concernientes al diseño de la interfaz gráfica del software, ni a la aplicación de pruebas de regresión, de instalación y desinstalación.
- Muchas de las preguntas que se proponen para evaluar algunas de las actividades que integran la práctica de desarrollo, no permiten una evaluación detallada de las mismas, donde se aprecie la ejecución de aspectos fundamentales en dichas actividades. Por ejemplo:
  - En la evaluación de la actividad referida a la *elaboración del documento de la propuesta de desarrollo*, no se plantean preguntas concernientes a la forma en la cual se define el alcance del software y su arquitectura.

    El alcance del software representa un elemento base de toda propuesta de desarrollo, dado que permite identificar las funcionalidades que debe ofrecer el software en términos de los procesos a automatizar y/o de las necesidades a cubrir. En este sentido, es prioritario evaluar si en la práctica de desarrollo se realiza un estudio de los procesos a

1 Según la *Free Software Foundation* un programa es software libre si se dan la siguientes libertades para el uso del mismo:

- Libertad de ejecutar el programa con cualquier propósito (libertad 0).
- Libertad de estudiar el funcionamiento del programa, y cambiarlo según se requiera (libertad 1).
- Libertad de redistribuir copias del programa (libertad 2).
- Libertad de distribuir copias de versiones modificadas del programa (libertad 3).

automatizar y/o de las necesidades de los usuarios, así como evaluar la forma en la cual se realiza dicho estudio.

La definición de la arquitectura de un software implica la consideración de un conjunto de aspectos, entre ellos las funcionalidades del software, las limitaciones tecnológicas existentes y los atributos de calidad asociados al software, a los cuales queda supeditado el cumplimiento de los requerimientos de usuarios [4]. Es por ello que, debe plantearse un método de evaluación para la definición de la arquitectura que considere los aspectos mencionados.

- En lo referido a la gestión de incidentes, no se evalúa la forma en la cual se reportan y gestionan los errores del software y/o sugerencias indicados por los usuarios. En el ámbito del software libre la gestión de incidentes constituye un elemento básico en el desarrollo colaborativo y en la mejora continua del software, pues éste permite gestionar de manera organizada el reporte y la corrección de errores, de allí la necesidad de considerarlo en la evaluación de la práctica de desarrollo.
- La evaluación de la especificación de los requerimientos funcionales, no contempla una evaluación detallada de la forma en que se describen textualmente los casos de uso del software. Por ejemplo, no se evalúa si en esta descripción se consideran elementos de interfaz, se incluye condiciones de entrada y salida, se indican los actores que interactúan con cada caso de uso, entre otras. Cabe destacar que la descripción de los casos de uso representa un documento esencial tanto para la fase de Codificación como para la fase de Pruebas del software, pues en éste se especifica qué debe hacer el software ante las solicitudes del usuario, por tanto, los casos de uso deben ser descritos a detalle y deben contener toda la información necesaria para que sirvan de apoyo en las fases mencionadas.
- En lo que respecta a la evaluación del modelado de datos persistentes, no se considera la evaluación de algunos aspectos que podrían dificultar el ajuste de los datos a las necesidades de despliegue. Entre estos aspectos se encuentra, por ejemplo, el uso de diversos gestores de datos. Por otro lado, la evaluación que se plantea para el modelado de datos no contempla una evaluación de la especificación de datos a intercambiar, en caso de que el software deba interoperar con otras aplicaciones.
- En la evaluación de las actividades concernientes a la codificación, no se toman en cuenta elementos como la reutilización de código y el uso de patrones de programación, los cuales resultan de gran utilidad en la práctica de desarrollo, puesto que agilizan la construcción del software y mejoran la calidad del mismo.
- La forma en que se documenta el código fuente no es evaluada, dado que no se realizan preguntas orientadas a conocer cómo se lleva a cabo tal documentación. En el ámbito del software libre un código bien documentado facilita y promueve el trabajo colaborativo en torno al desarrollo del mismo, de allí la importancia de evaluar la forma en que se documenta el código y el tipo de documentación que se indica sobre éste.
- No se evalúa el tipo de información que deberían contener los manuales de uso de software.

3 Mejoras en los Procesos Involucrados en el Aseguramiento de Calidad de Software

En vista de las dificultades observadas durante la validación del modelo de aseguramiento de calidad, surge la necesidad de adecuarlo a fin de brindar una mayor correspondencia de éste con el contexto de desarrollo de software de CENDITEL, así como con otros contextos de desarrollo que sean similares al mismo. En función de ello, se requiere una reformulación de la Metodología de Desarrollo Colaborativo de Software Libre [2] propuesta por CENDITEL, en vista de que ésta constituye una guía de orientación de la práctica de desarrollo que se lleva a

cabo en CENDITEL, y, a su vez, representa el modelo de referencia sobre el cual se fundamenta el método propuesto para la evaluación de la práctica de desarrollo.
A continuación, se describen las mejoras en las cuales se trabaja actualmente tanto a nivel de la metodología de desarrollo como a nivel del proceso de aseguramiento de calidad.

3.1. Adecuaciones en la Metodología de Desarrollo Colaborativo de Software Libre

En cuanto a la reformulación de la metodología de desarrollo se añade un conjunto de actividades por cada proceso que compone esta metodología, a saber: Conceptualización de Proyectos de Software Libre, Administración de Proyectos de Software Libre y Construcción de Aplicaciones de Software Libre. Por otro lado, se agrega una serie de recomendaciones por cada tarea de cada uno de los procesos mencionados, con el objetivo de mejorar la forma de realizar la práctica de desarrollo en base a modelos y patrones, así como en base a experiencias adquiridas en la práctica de desarrollo que se da en CENDITEL. Seguidamente se presentan las adecuaciones planteadas para cada uno de estos procesos.

3.1.1. Conceptualización de Proyectos de Software Libre

Las sugerencias para mejorar el proceso de Conceptualización de Proyectos se indican a continuación:

- Incorporar actividades referidas a la recopilación y análisis de información de los procesos que se requieren automatizar en una aplicación de software, con el fin de entender el dominio de la aplicación, así como los problemas y/o necesidades en relación a dichos procesos, todo ello con el propósito de definir el alcance del software conforme a los requerimientos de usuarios. Esta actividad implica el modelado de procesos y la elaboración de los diagramas de casos de uso.
- Agregar la selección de un entorno de desarrollo (*framework*) como una actividad de este proceso. La utilización de este tipo de entornos permite la reutilización de librerías, funciones, y otras ventajas que éstos ofrecen para la construcción de código.
- Indicar recomendaciones basadas en buenas prácticas y patrones que brinden orientación en la realización de actividades como el modelado de procesos y casos de uso, definición de la arquitectura y selección de lenguajes de programación. Por ejemplo, entre estas recomendaciones se encuentran:
    - En el caso del modelado de procesos, plantear sugerencias orientadas al tipo de información que deben contener los diagramas asociados a este tipo de modelado, con el fin de que esta información sirva a cualquiera de los integrantes del equipo de desarrollo para entender con claridad los procesos a automatizar.
    - En lo que respecta a la definición de la arquitectura de software, resulta pertinente proponer una serie de criterios para apoyar dicha definición, enfocados al cumplimiento de requerimientos de calidad y a facilitar futuras modificaciones en el software. Uno de estos criterios podría ser el uso de patrones arquitectónicos como el Modelo-Vista-Controlador, teniendo en cuenta que éste facilita el cumplimiento de atributos de calidad asociados a la funcionalidad y la mantenibilidad, pero dificulta el cumplimiento de algunos otros atributos asociados al desempeño y a la portabilidad del software [5].

3.1.2. Administración de Proyectos de Software Libre

Para potenciar las labores de planificación, coordinación y seguimiento, tanto de tareas como de resultados de los proyectos de software, se sugieren algunas actividades y recomendaciones:

- Incorporar como actividad la realización de reuniones semanales entre el equipo de desarrollo, a fin de que las mismas sirvan como medio para generar procesos de enseñanza-aprendizaje entre los integrantes de este equipo, pues en ellas, por ejemplo, se pueden exponer las dificultades que se puedan presentar durante el desarrollo del software, así como las lecciones aprendidas al momento de abordar dichas dificultades.
- Incluir la gestión de incidencias como parte de la administración de proyectos de software, dado la importancia de esta actividad para la mejora del software. En este sentido, para

facilitar y promover estas tareas, es necesario brindar a los usuarios y al equipo de desarrollo mecanismos que permitan reportar y atender de forma oportuna los incidentes que se puedan presentar.

- Plantear recomendaciones fundamentadas en buenas prácticas y patrones que guíen la ejecución de actividades como la gestión de incidentes, seguimiento de tareas a distancia, priorización de desarrollo de funcionalidades, entre otras. En el caso de la gestión de incidencias se sugiere recomendar que los mecanismos de reporte se encuentren en el sitio web del proyecto y que puedan ser distinguidos con claridad, de manera que sea fácil acceder a los mismos. Para la priorización de desarrollo de funcionalidades del software se aconseja mencionar criterios para realizarla. Por ejemplo, como criterios se podrían indicar: la identificación de riesgos de desarrollo, la importancia o prioridad que el usuario asigna a cada funcionalidad y la dependencia entre funcionalidades.

3.1.3. Construcción de Aplicaciones de Software Libre

Para brindar mayor detalle en las fases que componen la construcción de una aplicación de software, a fin de que mejore la ejecución de las actividades que integran estas fases, así como los productos que de ellas se deriven, se plantea como necesario agregar algunas actividades y recomendaciones en estas fases:

- Fase de Especificación de Requerimientos:
    - Plantear algunas recomendaciones con respecto a la descripción textual de los casos de uso, en función de las cuales se pueda conducir una práctica de especificación de requerimientos para reflejar con claridad las indicaciones de lo que se requiere construir (requerimientos funcionales). Ello permite que el arquitecto de software, el diseñador gráfico, los programadores, los probadores y los documentadores no tengan necesidad de leer ningún otro documento para realizar su trabajo en torno al desarrollo de software [6]. Para ello es importante que se mencione el tipo de información que debe contener cada caso de uso especificado, por ejemplo, actores que interactúan con el caso de uso, condiciones de entra y salida, flujos básico y alternativos, entre otros.
    - Agregar una actividad relacionada a la discusión de la especificación de requerimientos con el equipo de desarrollo, 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. La discusión de la especificación de requerimientos facilita el identificar posibles ambigüedades en ésta, así como el agregado de información de utilidad que no haya sido tomado en cuenta por el Analista que realiza la especificación.
- Fase de Análisis y Diseño:
    - Exponer recomendaciones en torno al modelado de datos persistentes, con el propósito de facilitar la identificación de estos datos y su descripción a través de un modelo respectivo, así como con el propósito de considerar posibles cambios a nivel de gestor de datos y/o del modelo de datos.
    - Incorporar una actividad que contemple la especificación de datos a intercambiar con otras aplicaciones de software, así como una serie de recomendaciones en torno a ésta. La interoperabilidad es un elemento clave a tener en consideración durante la fase de Análisis y Diseño, siempre y cuando el software a construir lo requiera, pues en esta fase se definen los mecanismos con los cuales se realizará el intercambio de datos.
    - Platear algunos criterios de apoyo al diseño de la interfaz de usuario, basados en patrones o estándares de diseño que aporten características de calidad al software en términos de usabilidad. Por ejemplo, se podría recomendar el considerar estándares como los planteados en las ISO 9421 [7] e ISO 14915 [8].
- Fase de Codificación:

- Incluir actividades referidas a la documentación del código fuente, empaquetado del software y gestión de versiones.

  La documentación del código fuente facilita los procesos de apropiación del software (mantenibilidad), dado que permite entender el código con mayor facilidad y en menor tiempo.

  Para ofrecer características de calidad referidas a portabilidad, como por ejemplo, el uso del software en varias distribuciones, lo cual incide en el aumento del número de usuarios probando el software, se requiere contar con actividades destinadas al empaquetado del software en diferentes distribuciones.

  Para el manejo organizado de las versiones del código resulta pertinente el uso de sistemas de control de versiones, pues éstos permiten llevar un control de los cambios que se realizan sobre las versiones del software, facilitando así el trabajo colaborativo en torno al desarrollo del mismo.

- Proponer recomendaciones en base a buenas prácticas de documentación, empaquetado y gestión de versiones. Por ejemplo, dar sugerencias respecto al tipo de información básica que debe contener la documentación del código fuente, a saber, información sobre qué hace cada componente (función, clase, método, etc.), qué parámetros necesita y qué datos devuelve [9].

- Fase de Pruebas:
  - Agregar una actividad para la aplicación de pruebas de regresión, debido a que éstas son de gran utilidad para verificar si se han introducido errores en el software después de la corrección de errores o de modificaciones al código.
  - Exponer recomendaciones sobre formas o métodos para la formulación y aplicación de planes de pruebas, que permitan orientar esta práctica a fin de alcanzar los objetivos de toda prueba, a saber, encontrar errores.

    Entre las recomendaciones se podría sugerir el uso de técnicas de prueba, como por ejemplo, la técnica Caja Negra (*black-box testing*) [10] basada en casos de uso. Por otro lado, se podría proponer el uso de herramientas para grabar pruebas funcionales, de modo que éstas puedan ser reproducidas posteriormente como pruebas de regresión. En cuanto a la aplicación de pruebas, resulta pertinente recomendar que se indique qué tipo de información se debe presentar al momento de reportar un error, con el objetivo de que se facilite la identificación y corrección del mismo. En estos casos se podría sugerir, dependiendo del tipo de aplicación de software, presentar capturas de pantalla o sesión donde se indica el error en el software, así como presentar archivos de bitácoras donde se registran los eventos con niveles configurables de detalle.
- Fase de Liberación:
  - Proponer recomendaciones en torno a la publicación del software, como por ejemplo, la publicación en repositorios oficiales como Debian, Ubuntu, Fedora, entre otros.
  - Plantear sugerencias respecto al tipo de información que debe contener un manual de usuario, a fin de que éste pueda facilitar tanto el uso del software como la instalación del mismo.

3.2. Adecuaciones en el Proceso de Aseguramiento de Calidad en el Desarrollo de Software Libre

Este proceso requiere mejoras, tanto a nivel del método de evaluación de la práctica de de desarrollo como a nivel de los métodos planteados para evaluar la calidad en aplicaciones de software. Estas mejoras o adecuaciones se esbozan a continuación.

3.2.1. Evaluación de la Práctica de Desarrollo de Software Libre

En este caso es necesario adecuar las preguntas planteadas para evaluar la práctica de desarrollo, según las mejoras propuestas en la sección 3.1. Para ilustrar la adecuación que se propone, se presenta, en la Tabla 2, un resumen del

método de evaluación sugerido para el proceso de Conceptualización de Proyectos de Software Libre, pero con antelación a ello se muestra, en la Tabla 1, una breve descripción de las actividades planteadas para dicho proceso, según lo indicado en la sección 3.1.1. Ello permite entender el tipo de preguntas que se formulan para evaluar el proceso, puesto que la descripción de éste representa el modelo de referencia sobre el cual se basa su respectiva evaluación.

Tabla 1. Propuesta de actividades que componen el 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 los procesos que se quieren automatizar, así como las problemáticas y/o necesidades en torno a éstos.	*Recomendaciones*: • Para identificar los problemas y/o necesidades en torno a los procesos que se desean automatizar, se requiere realizar reuniones entre los 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 entenderlos en función de las actividades que los integran. • Es importante que la información que se recopilará sobre los procesos a automatizar, contribuya a 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.
	Productos: • Minutas.
Actividad: Identificación de las funcionalidades del software	
Tarea: Elaborar los diagramas de procesos a automatizar.	*Recomendaciones*: • Un diagrama de procesos constituye una representación gráfica que capta la estructura y la dinámica de un proceso, lo cual permite describirlo especificando los datos (entrada y salida), las actividades, los roles y las reglas que regulan dicho proceso. • Cada diagrama de proceso 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 procesos [11].
	Producto: • Diagramas de procesos.
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 uno de éstos y que se requieren como insumos (entradas y/o recursos) en otros procesos [11].
	Producto: • Diagrama de relación entre procesos.
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 [11] de cada uno de ellos con el objetivo de identificar inconsistencias y conflictos en el flujo de ejecución de éstos, así como identificar si es necesario agregar o eliminar actividades a los mismos. De este análisis, se pueden generar nuevos diagramas de actividades en los que se propongan mejoras en la ejecución de los respectivos procesos.

	Productos: • Diagramas de actividades por proceso.
Tarea: Validar con los usuarios los diagramas de procesos y de actividades.	*Recomendaciones*: • La validación de los diagramas de procesos y actividades por parte de los usuarios es determinante para el desarrollo de software, pues los usuarios son quienes conocen con detalle cada uno los procesos que se requiere automatizar. De allí, la relevancia de que sean éstos quienes revisen los diagramas en los cuales se modelan los procesos y en base a los cuales se diseñará el respectivo software.
	Productos: • Diagramas de procesos validados. • Diagramas de relación entre procesos validados. • Diagramas de actividades validados.
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*: • Los casos de uso [11] del software se identifican en base a los problemas y/o necesidades que se planteen en torno a los procesos que se desean automatizar, así como en base a los diagramas de procesos y actividades. • Se recomienda no superar más de tres niveles de relación entre casos de uso (ya sean relaciones de inclusión, extensión o generalización), para facilitar la lectura de estos diagramas.
	Productos: • Diagramas de casos de uso (alcance del software).
Actividad: Descripción general de la arquitectura	
Tarea: Describir a grandes rasgos el tipo de arquitectura del software a desarrollar.	*Recomendaciones*: • Al momento de definir la arquitectura del software, es decir, sus componentes (clases, módulos, funciones, métodos y piezas de código) y la interacción (comunicación) entre éstos, se requiere tener en cuenta no sólo las funcionalidades del software y las limitaciones tecnológicas existentes, sino también los atributos de calidad asociados al software, ya que la arquitectura que se defina puede inhibir o facilitar el cumplimiento de dichos atributos [4]. Por ejemplo, el atributo de calidad respectivo al desempeño del software depende de los componentes que se definan para éste y de su ubicación en los procesadores, así como de los caminos de comunicación entre dichos componentes, etc. Otro elemento a considerar para la definición de la arquitectura es la disponibilidad de componentes que faciliten generar bitácoras de funcionamiento con niveles de detalle configurables, las cuales son de gran utilidad para la detección y corrección de errores. Similarmente, otro elemento de interés para dicha definición es la disponibilidad de funcionalidades que posibiliten la realización de pruebas de perfilamiento, las cuales ayudan a detectar componentes para optimización de desempeño. • 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, debido a que 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 [5]. • Es importante que se estudien varias alternativas de la arquitectura, 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. • Para describir de manera general la arquitectura del software se pueden utilizar diagramas sencillos, donde se representen a grandes rasgos los componentes del software. Los diagramas más detallados de la arquitectura pueden ser elaborados en la fase de Análisis y Diseño, contenida en el proceso de Construcción de Aplicaciones de Software Libre, pues en esta etapa se tiene mayor conocimiento de los componentes del software, así como de sus interacciones.
	Productos: • Diagrama de la arquitectura del software.
Tarea: Fundamentar el uso de la arquitectura seleccionada.	*Recomendaciones*: • A fin de hacer explicito las decisiones de diseño se sugiere elaborar un documento en el que se fundamenten las razones a las cuales obedece la selección de la arquitectura del software, en función de los criterios utilizados. Ello facilita no sólo el entender la arquitectura sino también facilita futuros procesos de mantenimiento del software.
	Producto: • Documento de fundamentación de diseño arquitectónico.
Actividad: Selección del entorno de desarrollo (*framework)*	
Tarea: Seleccionar un entorno de desarrollo (*framework).*	*Recomendaciones:* • Dado las bondades que ofrecen los entornos de desarrollo, se recomienda el uso de estas herramientas a fin de reutilizar librerías, funciones y otras ventajas que éstas ofrecen en relación a la construcción de código. Existen numerosos entornos de desarrollo, entre ellos: *Django*, *Ruby on Rails*, *Symfony*. • Para seleccionar el entorno de desarrollo se recomiendan algunos criterios [12] como: a) Popularidad, es decir, el entorno de desarrollo debe ser reconocido entre la comunidad de usuarios, lo cual indica calidad y completitud de sus componentes. b) Sostenibilidad, ello implica mantenimiento actualizado del entorno. c) Soporte, implica la facilidad de encontrar respuestas a las preguntas que puedan surgir sobre el entorno de desarrollo. En este caso es necesario identificar los responsables de brindar dicho soporte, así como los medios que se utilicen para ello (listas de correo, IRC, entre otros). d) Uso de patrones de diseño, lo cual permite la reutilización de código estándar para efectuar tareas particulares dentro del software. e) Seguridad, el entorno de desarrollo debe ofrecer funciones que minimicen casos de vulnerabilidad en el software. f) Documentación que facilite el uso del entorno de desarrollo. g) Licencia de uso, este criterio es muy importante dado que la utilización de un entorno de desarrollo implica automáticamente la aplicación de la licencia de éste sobre el software desarrollado. h) Prueba del entorno, ello permite constatar la aplicabilidad de éste al tipo de desarrollo a realizar. i) Facilidad de aprendizaje, es decir, se debe requerir poco tiempo para aprender a utilizar el entorno de desarrollo. j) Interoperabilidad, ello implica que las funciones y librerías que ofrece el entorno puedan interactuar con otros software. • Además de los criterios mencionados en el ítem anterior debe considerarse también el tipo de arquitectura del software, ya que los entornos brindan elementos como los patrones de diseño que pueden ser determinantes para la arquitectura.
	Productos: • Entorno de desarrollo seleccionado y la fundamentación de dicha selección.

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, indicando el manejador de base de datos, los lenguajes de programación, el entorno de desarrollo, entre otros; 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.
	Producto: • Propuesta de desarrollo.

Tabla 2. Propuesta del método de evaluación para el proceso de Conceptualización de Proyectos de Software Libre

Actividad	**Preguntas de evaluación por actividad**	**Opciones de respuesta**
Identificación de las funcionalidades del software	¿Se han elaborado los diagramas de procesos a automatizar?	Si No
	¿En los diagramas de proceso se describen las entradas, productos (salidas), recursos, reglas, objetivos y actores de cada proceso?	Nada Poco Medianamente Ampliamente Completamente
	¿Se ha elaborado el diagrama de relación entre procesos?	Si No
	¿En el diagrama de relación entre procesos se indican los productos que se generan en cada proceso y que se requieren como insumos (entradas y/o recursos) en otros procesos?	Nada Poco Medianamente Ampliamente Completamente
	¿Se han elaborado los diagramas de actividades por proceso?	Si No
	¿Se han validado con los usuarios los diagramas de procesos y actividades?	Si No
	¿Se ha establecido el alcance del software en función de los casos de uso del mismo?	Si No
	¿En los diagramas de casos de uso no se superan más de tres niveles de relación?	Si No
Descripción general de la arquitectura	¿Se ha descrito el tipo de arquitectura del software a desarrollar?	Si No

	¿Se han contemplado aspectos como las funcionalidades del software, las limitaciones tecnológicas existentes y los atributos de calidad asociados al software para la definición de la arquitectura?	Nada Poco Medianamente Ampliamente Completamente
	¿La arquitectura del software contempla patrones y/o estilos arquitectónicos, así como patrones de diseño, que sean de utilidad para efectuar tareas dentro del software?	Nada Poco Medianamente Ampliamente Completamente
	¿Se ha elaborado un documento donde se fundamenten las razones a las cuales obedece la selección de la arquitectura del software?	Si No
Selección del entorno de desarrollo (*framework)*	¿En el proyecto se utiliza un entorno de desarrollo?	
	¿El entorno de desarrollo ofrece soporte y mantenimiento actualizado?	
	¿El entorno de desarrollo contempla patrones de diseño que permitan efectuar tareas particulares dentro del software?	Si No
	¿La licencia de uso del entorno de desarrollo establece el uso libre del mismo?	
Elaboración de la propuesta de desarrollo del software	¿El proyecto cuenta con una propuesta de desarrollo?	Si No
	¿La propuesta de desarrollo contiene información como: necesidades y/o problemáticas que se abordarán con el software a desarrollar, solución que se propone, alcance del software, descripción general de la arquitectura, potenciales actores colaboradores en el desarrollo del software, metodología de desarrollo, plataformas de operación y desarrollo, licencias de código y documentación?	Nada Poco Medianamente Ampliamente Completamente

Las actividades que sean evaluadas con alguna de las opciones “No”, “Nada”, “Poco”, “Medianamente” y “Ampliamente” requieren ser mejoradas en su ejecución, en mayor o menor grado, por lo cual el evaluador debe reportar al equipo de desarrollo el tipo de mejoras que se requieren en estas actividades.

3.2.2. Evaluación de la Calidad en Aplicaciones de Software Libre

La falta de un equipo de aseguramiento de calidad y la escasa documentación de las aplicaciones desarrolladas en CENDITEL, constituyen dos (2) de las tres (3) dificultades observadas durante la validación del proceso de Evaluación de Calidad en Aplicaciones de Software

Libre. La tercera dificultad esta referida a la carencia, en el respectivo proceso, de métodos de evaluación de características de calidad asociadas a mantenibilidad y portabilidad, entre ellas documentación y publicación del código fuente, empaquetado del software en diferentes distribuciones y su publicación en repositorios oficiales.

Las dos primeras dificultades no se asocian directamente a necesidades de modificación en el proceso mencionado, sin embargo, las mismas repercuten en la posibilidad de su aplicación. Para tratar estas dificultades se requiere contar con personal destinado a las labores de documentación y pruebas de software, así como a las actividades de aseguramiento de calidad.

El resolver la tercera dificultad, implica una serie de adecuaciones en torno a la incorporación, en este proceso, de la evaluación de características de calidad como documentación del código fuente, gestión de versiones, empaquetado del software en diferentes distribuciones y su publicación en repositorios oficiales.

En la Tabla 3, se presenta un ejemplo del tipo de evaluaciones que se requieren incorporar al proceso de Evaluación de la Calidad en Aplicaciones de Software Libre, a fin de considerar en éste las características de calidad descritas en el párrafo anterior.

Tabla 3. Propuesta de evaluación para características de calidad de software asociadas a mantenibilidad y portabilidad

Característica de calidad	Preguntas para evaluar característica	Opciones de respuesta
Documentación del código fuente	¿Se encuentran documentados todos los componentes (funciones, métodos y/o clases) del software?	Nada Poco Medianamente Ampliamente Completamente
	¿En la documentación de los componentes se incluye información sobre qué hace cada componente, qué parámetros necesita y qué datos devuelve?	
Documentación técnica del software	¿El software cuenta con especificación de requerimientos?	Nada Poco Medianamente Ampliamente Completamente
	¿La especificación de requerimientos se encuentra actualizada en función de los cambios realizados al software?	
Gestión de versiones del software	¿Se lleva un control de los cambios realizados al software?	Nada Poco Medianamente Ampliamente Completamente
Empaquetado del software	¿El software está empaquetado para más de una distribución Linux?	Si No
Publicación en repositorios	¿Está el software publicado en algún repositorio oficial, como el de Debian, Ubuntu, Fedora, etc?	

Conclusiones

El mostrar nuestra experiencia en torno al aseguramiento de calidad, en el desarrollo de aplicaciones de software libre, nos permite tratar aspectos característicos de este tipo de aplicaciones no contemplados en estándares de calidad como la norma ISO/IEC TR 9126-3, así como en mostrar particularidades de la práctica de desarrollo de software libre que tributan a la generación de conocimiento, a través de procesos colaborativos de enseñanza aprendizaje que permiten la mejora continua de ésta, razón por la cual merecen especial atención en el área de aseguramiento de calidad de software.

Con la validación del proceso de Evaluación de Calidad en Aplicaciones de Software Libre, basado en la norma ISO/IEC TR 9126-3, se identificaron algunas necesidades particulares del área de desarrollo de la Fundación CENDITEL, las cuales se asocian a problemáticas a nivel organizacional, como por ejemplo, la falta de personal

encargado de tareas de documentación y pruebas de software. Adicionalmente, esta validación permitió distinguir algunos elementos asociados a calidad de software, no contemplados en la mencionada norma, que tienen una significativa importancia para la práctica de desarrollo de software libre, dado que repercuten de manera significativa en el uso y mejoras de las aplicaciones de software libre, razón por la cual se plantea la consideración de estos elementos en el proceso mencionado.

Durante la validación del proceso de Evaluación de la Práctica de Desarrollo de Software Libre se reconocieron algunas debilidades en dicho proceso. Estas debilidades reflejan la necesidad de agregar la evaluación de actividades primordiales en la práctica de desarrollo de software libre, entre ellas la gestión de versiones, el empaquetado de software y su publicación en repositorios oficiales (como por ejemplo, Debian y Ubuntu). Así mismo, es necesario llevar un seguimiento más detallado de las actividades que integran la práctica de desarrollo, a fin de buscar la mejora de las mismas, para lo cual el seguimiento a estándares y buenas prácticas de desarrollo resulta esencial.

A consideración de lo expuesto, se estima pertinente plantear un método de evaluación para la práctica de desarrollo que evalúe no sólo el hecho de si se realizan o no las actividades que conforman esta práctica, sino también dirigido, con especial interés, a la evaluación de la forma en la cual estas actividades se llevan a cabo. Para ello, se requiere que el método de evaluación tome como referencia un modelo de procesos en el que se indiquen mejores formas de realizar las actividades de la práctica de desarrollo, tomando como base para ello patrones, estándares y buenas prácticas derivadas de las experiencias en el desarrollo de aplicaciones, tanto a nivel privativo como libre. Esto implica una mejora de la metodología de desarrollo de CENDITEL, dado que ésta constituye el modelo de referencia que se utiliza en el proceso de evaluación de la práctica de desarrollo. Esta mejora en la metodología estaría orientada al agregado de las actividades mencionadas como primordiales dentro de la práctica de desarrollo, así como al agregado de recomendaciones o sugerencias en torno a la ejecución de las actividades que conforman esta práctica.

REFERENCIAS

[1] Norma ISO/IEC TR 9126-3. Software engineering. Product quality. Part 3: Internal metrics. 2002.

[2] J. Alvarez, J. Aguilar y O. Terán. Metodología para el Desarrollo Colaborativo de Software Libre (Versión I). Mérida, Venezuela, 2007. Disponible en: http://calidad-sl.cenditel.gob.ve/55-2/

[3] J. Alvarez, R. Briceño, S. Solé, M. Venegas y L. Rojas. Modelo para el Aseguramiento de Calidad en el Desarrollo de Software Libre. Mérida, Venezuela, 2011. Disponible en http://calidad-sl.cenditel.gob.ve

[4] L. Bass, P. Clements y R. Kazman. *Software Architecture in practice. Addison-Wesley.* 1998.

[5] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad y M. Stal. *Pattern – Oriented Software Architecture. A System of Patterns. John Wiley & Sons.* Inglaterra, 1996.

[6] A. Pérez. Estructuración y Especificación de Casos de Uso. (s. f.). Disponible en: https://sites.google.com/site/alfonsoperezr/investigacion/estructuracin-y-especificacin-de-casos-de-uos

[7] Norma ISO 9421: *Ergonomic requirements for office work with visual display terminals (VDT)*

[8] Norma ISO 14915: *Software ergonomics for multimedia user interface.*

[9] F. Fernández. PhpDocumentor. 2008. Disponible en: http://www.epsilon-eridani.com/cubic/ap/cubic.php/doc/phpDocumentor---documentacion-para-codigo-PHP-246.html

[10] R. Pressman. Ingeniería del Software: Un enfoque práctico. McGraw-Hill. Madrid, 2002.

[11] J. Montilva, I. Besembel y F. Zerpa. Modelado de Sistemas Usando UML 2.0. Centro de Excelencia en Ingeniería del Software. Mérida, Venezuela, 2007.

[12] J. Melgoza. 10 Criterios para Elegir el Framework Correcto. 2014. Disponible en:

http://jonathanmelgoza.com/blog/10-criterios-para-elegir-el-framework-correcto