Impacto Tecnológico En México: El uso de estándares IEEE para el desarrollo de proyectos de calidad en la Ingeniería de Software

Technological Impact in Mexico: The use of IEEE standards for the development of quality projects in Software Engineering

DÁVILA-NICANOR, L.[1]& BENHUMEA-PEÑA, A.

Universidad Autónoma del Estado de México. Centro Universitario UAEM Valle de México. Blvd. Universitario s/n Predio San Javier, Atizapán de Zaragoza, México, C.P. 54500

ID 1° Autor: Leticia Dávila-Nicanor/ **ORC ID**: 0000-0002-4691-4997, **Researcher ID Thomson**: E-1397-2018, **arXiv ID**: ldavilan, **CVU CONACYT-ID**: 162561

8 Introducción

Frecuentemente los ingenieros de software se encuentran con problemas de concordancia y satisfacción de las expectativas del usuario o cliente al momento de realizar proyectos de aplicación de software, esta problemática se percibe con mayor énfasis en el planteamiento de proyectos donde participa una comunidad multidisciplinaria de personas para el desarrollo de un proyecto de software debido a la inexistencia de marcos teóricos comunes que pueden ser utilizados en el entendimiento de los requisitos funcionales y no funcionales de software por todos los participantes. Galvis (1996) reconoce la necesidad dentro de la ingeniería de software de crear un marco teórico de referencia que tenga como objetivo principal la satisfacción de los requerimientos funcionales y no funcionales de un proyecto de software en sus diversas fases de desarrollo, todo esto mediante la construcción de un material didáctico informatizado. Por tal motivo este documento tiene como objetivo, presentar de una forma sencilla y directa las herramientas, los métodos y los procedimientos que la ingeniería de software utiliza para proporcionar desde el punto de vista técnico un producto eficiente y de calidad, donde el conocimiento y aplicación de estas teorías permitan el desarrollo en la ejecución de nuevos proyectos.

El documento se estructura en dos grandes bloques. El primero de carácter esencialmente descriptivo, donde se aborda el estado del arte de la ingeniería de software y la calidad, tomando como referencia normas internacionales y su relación con la satisfacción del usuario. En el segundo bloque se realiza la síntesis de los tres modelos del ciclo de vida de un software utilizados a nivel desarrollo, y se incluye la elaboración de un caso de estudio bajo la aplicación de un modelo de desarrollo y el análisis del proyecto bajo estándares internacionales de la IEEE. Por último, tras las conclusiones que a modo de resumen sintetizan el contenido del documento se incluyen tanto las referencias bibliográficas empleadas en la elaboración del mismo, así, como un conjunto de recomendaciones de mejora continua que se derivan del análisis efectuado.

La ingeniería de software

Existen diferentes conceptos y definiciones alrededor de lo que es y lo que debería ser la ingeniería de software, para las finalidades de este documento se retoma una de las primeras definiciones formales propuesta por Frits Bauner en 1969 quien define a la ingeniería de software como: “el establecimiento y uso de principios de ingeniería robustos, orientados a obtener software económico y que funcione de manera eficiente sobre máquinas reales”. Pressman (1993) señala que la ingeniería del software surge a partir de las ingenierías de sistemas y de hardware, considerando tres elementos clave para el desarrollo de nuevos productos de software: que son los métodos, las herramientas y los procedimientos que facilitan el control para el proceso de desarrollo de software, los cuales brindan a los desarrolladores una base de calidad para el perfeccionamiento de productos.

A estos modelos que contienen métodos, procedimientos y herramientas se les denomina paradigmas de la ingeniería del software y la elección de alguno de estos paradigmas se realiza con base en la naturaleza del proyecto y su aplicación, teniendo en cuenta los controles y las entregas de información a realizar por el software. Para la construcción de un sistema de software existen una serie de pasos a desarrollar, a los cuales se les denomina proceso, el cual se describe como la obtención de los requisitos funcionales del sistema, el diseño, la codificación de los algoritmos a utilizar y las pruebas del sistema tomando en cuenta el mantenimiento y actualizaciones. J. Juzgado (1996) sostiene que el nombre dado a este proceso inicialmente como “ciclo de vida” ha sido relegado en los últimos años, utilizando en su lugar “proceso de software”, lo que cambia totalmente la perspectiva de producto a proceso.

Para el desarrollo de un proyecto de ingeniería de software es necesario establecer un enfoque sistemático y disciplinario, a través del uso de metodologías de desarrollo, las cuales influyen directamente en el proceso de construcción y se elaboran a partir de un marco definido por uno o más ciclos de vida. Según Piattini (1996), no hay una definición universalmente aceptada para el concepto de metodología, pero existe un acuerdo donde se considera a la metodología como “un conjunto de pasos y procedimientos que deben seguirse para el desarrollo del software”. Por su parte Maddison (1983) define a la metodología como “un conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas y documentación como aspectos de formación para los desarrolladores de sistemas de información”. Generalmente las metodologías persiguen tres aspectos principales:

- Mejores aplicaciones, tendencia a mejorar la calidad, aunque a veces no es suficiente.
- Un proceso de desarrollo controlado, que asegure el buen uso de recursos y costos asignados.
- Un proceso estándar en la organización, donde no repercutan los cambios de personal.

De acuerdo a Somerville (2017), el proceso de desarrollo de software se define como un conjunto de actividades para especificar, diseñar, implementar y evaluar un sistema de software. Estas actividades o etapas, tienen una secuencia lógica porque sus salidas, son las entradas de las siguientes. En el caso de la primera etapa, la especificación de requerimientos se tiene como objetivo resolver el ¿qué servicios debe tener el sistema de software?, mientras que en el Diseño se resuelve el ¿cómo? resolver cada uno de los servicios planteados en la especificación del sistema. En la implementación se utilizan lenguajes de programación en función del diseño planteado. Tomando en cuenta la distribución de esfuerzos, si las primeras etapas se hacen con orden y completitud, los recursos invertidos de todo el proceso de desarrollo es del 20%.

Mientras que en la etapa de evaluación se concentran la mayoría de los recursos del proyecto de software porque se ocupan alrededor del 40%, esto es debido a que se tienen que realizar las pruebas necesarias para evaluar la mayoría de caminos de funcionalidad entre variables y componentes de la arquitectura del sistema, con la finalidad de asegurar que el producto final se encuentra libre de errores potenciales. Cuando esta última etapa no se realiza de forma adecuada y los sistemas fallan, además de tener pérdidas económicas que ascienden a millones o trillones de dólares, tomando en cuenta a Cass (2014), también las organizaciones que los respaldan pierden la confianza y credibilidad de sus clientes, esta situación implica una pérdida aún mayor.

La realización de cada una de las etapas del proceso implica una serie de técnicas, procedimientos y estándares que se pueden utilizar. El uso de las técnicas y herramientas que nos ofrece la Ingeniería de Software depende de la problemática que se aborda, en este caso no implica el mismo esfuerzo y recursos el diseño de un sitio Web que la realización de un sistema de software que controle una sonda espacial. Incluso tomando el sitio Web como referencia, la complejidad de su desarrollo depende de las metas del negocio. Este es un proceso que en la actualidad es tan importante en el marco de los procesos de producción tecnológica de los países, que se han generado estándares y normas que obligan a llevar esta secuencia y orden, esto es con la finalidad de llevar el mismo flujo de actividades y cuya compresión pueda extenderse a cualquier lugar en donde se realiza cualquier tipo de desarrollo de sistemas de software de media y gran escala. Así un sistema de software que se desarrolla en México siguiendo los estándares internacionales debe comprenderse en cualquier otro país (Estados Unidos de América, la Unión Europea, Asia, etcétera).

Calidad en la ingeniería de software

El concepto de calidad generalmente resulta ser un problema abstracto, basado en la satisfacción de las necesidades de los usuarios o bien la descripción de procesos organizados que se plasman en manuales de operación, lo que en teoría debe garantizar el buen funcionamiento y la satisfacción del producto o servicio para los usuarios.

Dentro de la ingeniería de software la calidad es una necesidad que se aborda con el desarrollo de modelos matemáticos e instrumentos de evaluación, ya que los atributos de calidad que puede poseer un sistema de software dependen en gran medida de su funcionalidad y su contexto de operación.

De acuerdo a Fenton (1991) el proceso de la calidad se da mediante la asignación de números o símbolos a los diferentes atributos y entidades en un contexto real, por lo que se deben describir claramente a través de reglas establecidas, para lo cual existen diferentes normas internacionales a tomar en cuenta, es el caso de la norma internacional ISO 9126 que define a la calidad en términos relacionados con el interés de los usuarios por el uso del software o producto, tomando en cuenta diferentes atributos internos y externos; lo que representa la piedra angular de la definición de un proceso para evaluar la calidad de un software. Barry Boehm y McCall desarrollan un concepto de calidad basado en la propiedad de medir y evaluar; MacCall articula su concepto de calidad basado en los tres usos relevantes de un producto de software por el usuario:

- Las características de operación.

- La capacidad para ser modificado.
- La adaptabilidad a entornos nuevos.

Cada una de estas características están compuestas por una serie de factores como: la fiabilidad, facilidad de uso, integridad, facilidad de prueba, flexibilidad, portabilidad, interoperabilidad, fácil mantenimiento y reusabilidad. Estos factores a su vez se complementan de propiedades intrínsecas de software, tales como: la comunicación, la facilidad de operación, el control de acceso, el aprendizaje, la ejecución, almacenamiento, eficiencia, precisión, tolerancia de fallos, capacidad de expansión, eficacia, compatibilidad e independencia entre sistema y hardware. McCall desarrolla su principio de calidad bajo la siguiente función, FC:

$$FC = c_1 X m_1 + c_2 X m_2 + ... + c_n X m_n \quad (1)$$

Dónde:

- $c_1 \ldots c_n$ son los coeficientes de regresión y
- $m_1 ... m_n$ son las métricas que repercuten en el factor de la calidad. (MacCall, J. 1996)

Dichos criterios se pueden evaluar a través de métricas que se pueden calcular observando directamente el software. Para estos criterios, MacCall propone una serie de métricas que desafortunadamente sólo se pueden medir de forma subjetiva. Dichas métricas se pueden considerar como propiedades de comprobación para obtener los atributos específicos del software, basados en un esquema de graduación que va desde cero (considerado como bajo) a diez (considerado como alto) utilizando como medidas las propiedades intrínsecas del software enunciadas anteriormente.

El modelo de McCall también llamado Factor Criteria Metric (FCM) establece que cada factor de la calidad está compuesto por cierto número de criterios, que es lo que realmente se mide, ya que resultan ser más fáciles de comprender por los usuarios. En este modelo se describe el grado de pertinencia que tienen las relaciones entre factores. Boehm y McCall definen los atributos externos del modelo como: la confiablidad, usabilidad (utilidad) y al mantenimiento, eficiencia y testeabilidad como atributos internos; dentro de estos mismos, es pertinente tomar en cuenta a la estructurabilidad y a la modularidad ya que estos se reflejan sobre los atributos externos. A su vez la norma internacional IEEE 1061 y la norma internacional ISO 9126 proponen un modelo propio de medición de la calidad de software, basados en la medición de factores internos y externos similares a los que propone McCall.

En la década de los ochenta, el sector empresarial comenzó a crear modelos particulares de evaluación de la calidad por cada proyecto de software desarrollado, lo que dio origen al concepto de "calidad relativa". Para lo cual Gilb (1988) propone crear una especificación de los requisitos de calidad redactados por el usuario y por los analistas conjuntamente, con la finalidad de crear una lista de características que definan la calidad en cada aplicación. Dicho enfoque es asociado a la filosofía del Despliegue de la Función de la Calidad "Quality Function Deployment" (QFD), que es aplicada en el ámbito de la gestión de la calidad industrial. Otros modelos de evaluación de la calidad de software son los propuestos por Grady y Caswell (1987) que presentan otro enfoque de medición de la calidad, inspirado en el control estadístico de los procesos, este modelo es aplicado a la industria de fabricación, tomando en cuenta la calidad basada en la ausencia de fallos, defectos o errores en el funcionamiento del software.

Para evaluar la calidad de un software, habitualmente nos referimos a las medidas del producto, y dejamos de lado las medidas del proceso de desarrollo. Por métrica se puede definir a "una asignación de un valor a un atributo de una entidad de software, ya sea un producto o un proceso" (Fenton, N. 1991). En repetidas ocasiones las métricas representan medidas indirectas de la calidad ya que sólo se miden los resultados de esta. En los diferentes tipos de métricas de software podemos encontrar métricas basadas en el texto y extensión del código y métricas basadas en la estructura de control del código.

Métricas basadas en el texto y extensión del código. Estas métricas se basan en la cantidad de líneas de código que se toman como un indicador de tamaño, el número de líneas comentadas como un indicador de la documentación interna, el número de instrucciones, el porcentaje de líneas de código de documentación, etc. Halstead (1975), propone sus métricas basadas en este tipo de criterios, denominándolas: "Ciencia del software".

Métricas basadas en la estructura de control del código. En este contexto pueden tomarse dos subtipos de medidas; las relacionadas con el control intramodular del software, que se basan en el grafo de control y las relacionadas con la arquitectura, que a su vez se basan en el grafo de llamadas o en el diagrama de estructuras del código. Las métricas de McCabe (1976) pertenecen a las métricas del primer tipo y constituyen un indicador del número de caminos independientes basados en los conceptos matemáticos que existen en un grafo. Piattini (1996) sostiene que los resultados indican los mejores valores de métricas implicadas en la reducción de un mantenimiento posterior basado en el menor número de fallos. Es entonces donde surge la pregunta ¿en qué momento utilizar un modelo predefinido o definir un modelo propio? Ya que todos los anteriores se consideran modelos fijos típicos, lo que podría resultar una salida para crear y utilizar un modelo por cada desarrollo de software, así como lo sostiene Kitchenham y Walter (1989) en el "Constructive Quality Model" (COQUAMO).

Ciclo de vida del software

Un software pasa por diferentes etapas durante su desarrollo a las cuales se denomina ciclo de vida o ciclo de desarrollo, donde resulta pertinente definir las actividades, los procesos, y las tareas a desarrollar durante todo el proceso de creación. La norma internacional IEEE 1074 define al ciclo de vida del software como el "marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software abarcando la vida del sistema desde la definición de requisitos hasta la finalización de su uso" y la norma internacional ISO 12207-1 lo define como "Una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software". Estas normas tienen en común considerar a una actividad como un conjunto de tareas y a una tarea como una acción que transforma las entradas en salidas (Piattini, M. 1996).

Es así que la elección adecuada de un ciclo de vida para la elaboración de un desarrollo está basada en las características y funcionalidades del producto a obtener. Derivado del tipo de proyecto a realizar dependerá el uso del proceso, el plan de actividades y el ciclo de vida utilizado. A partir de este plan de actividades se puede estimar el tiempo y costo de cada una de las actividades que contempla el proyecto global, también se puede generar una estimación de los recursos necesarios para llevar a cabo cada una de estas actividades tomando en cuenta que algunas se repiten en diferentes etapas del proyecto y otras solo se realizan una vez.

El Institute of Electrical and Electronics Engineers (IEEE) y la International Standards Organization/International Electrochemical Commission (ISO/IEC) han publicado normas relacionadas con el tema del ciclo de vida como "Standard for Developing Software Life Cycle Processes" estándar de la IEEE para el desarrollo de los procesos durante el ciclo de vida del software (IEEE. 1991) y la "Software life-cycle process" que propone y evalúa el proceso del ciclo de vida del software (ISO. 1994). Charles Sigwart denomina ciclo de vida del software "a toda la vida del software, comenzando con su concepción y finalizando con la desinstalación del mismo" y ciclo de desarrollo según Sigwart se le denomina en ocasiones "al subconjunto del ciclo de vida que comienza con el análisis y concluye con la entrega del software o producto" (Sigwart, et al. 1990). Por lo tanto el ciclo de vida determina el orden de las fases del proceso de software y los criterios necesarios para avanzar a la siguiente fase del desarrollo.

El modelo en cascada. Es el modelo metodológico más antiguo, presentado por Winstone Royce en 1970, aunque son más conocidos los refinamientos realizados por Barry Boehm en 1980, por Ian Sommerville en 1985 y Sigwart en 1990. Este modelo es utilizado frecuentemente para proyectos pequeños, donde se tienen bien claros los requerimientos del sistema por todos los miembros que pertenecen al desarrollo del proyecto, el producto se desarrolla a través de una secuencia rigurosa de fases ordenadas en forma lineal descendente, de tal forma que el inicio de cada etapa depende de la culminación de la etapa anterior, permitiendo la interacción con el estado anterior. El número de etapas suele variar, pero en general suelen ser las siguientes:

- Definición de los requisitos del sistema.
- Diseño del sistema y del software.
- Implementación y pruebas de unidades.

- Integración y pruebas del sistema.
- Operación y mantenimiento.(Somerville, I. 2000)

Por su parte Mc Cracken y Jackson (1982) difieren con el uso de este modelo, ya que sostienen que en la práctica los proyectos reales rara vez siguen una secuencia tan rigurosa como esta, y que casi siempre hay más de una interacción que depende de la etapa anterior. Además, una deficiencia notable es que como el sistema no entra en funcionamiento sino hasta finalizar el proyecto, el cliente, recibe la primera versión del producto al momento de haber consumido casi la totalidad de los recursos. De esta forma cualquier error dentro del diseño detectado en la etapa de pruebas conduce necesariamente a un re-diseño y quizá a la re-programación del código, ocasionando un aumento en los costos del proyecto.

El modelo del prototipado evolutivo. El uso del modelo del prototipado evolutivo se centra en la idea de coadyuvar a comprender los requisitos que plantea el usuario en el desarrollo de software, sobre todo si este no tiene una idea clara de lo que desea. También se puede utilizar cuando el ingeniero de software tiene dudas sobre la arquitectura o algoritmos a utilizar dentro de la solución propuesta. Con este modelo se generan signos visibles de progreso especialmente cuando existe premura del desarrollo por parte de la entidad de financiamiento.

El modelo del prototipado evolutivo proporciona una versión preliminar de lo que será el producto y sus funciones acotadas en un principio y que podrán evolucionar paulatinamente a través de refinamientos sucesivos basados en las especificaciones del sistema hasta llegar a la versión final. Al hacer uso de prototipos, las etapas del ciclo de vida quedan de la siguiente manera:

- Esbozo de la descripción.
- Diseño, desarrollo e implementación del prototipo.
- Prueba del prototipo.
- Refinamiento iterativo del prototipo.
- Refinamiento de las especificaciones del prototipo.
- Diseño e implementación del sistema final.
- Explotación (u operación) y mantenimiento.

El modelo de prototipos evolutivos es uno de los más usados ya que se puede modificar y ampliar fácilmente, “es recomendable usar prototipos desechables para esclarecer los aspectos del sistema que no se comprenden adecuadamente” (J. Juzgado, 1996).

El modelo en espiral de Boehm. En 1988 Barry Boehm propone el modelo en espiral con la finalidad de superar algunas limitaciones de su modelo de cascada propuesto en 1980. Este modelo de espiral se forma a partir de una serie de ciclos de desarrollo que van evolucionando. Los ciclos internos de la espiral denotan análisis y prototipado que se conjuntan con los espirales externos del modelo clásico. Esta dimensión radial contempla los costos acumulativos y la dimensión angular representa el progreso obtenido en cada etapa del desarrollo. Cada ciclo comienza identificando los objetivos, las alternativas y las restricciones del proyecto, es decir, evalúa las alternativas de solución con base en los objetivos planteados, tomando en cuenta sus restricciones en cada caso, en ese momento se puede llevar a cabo la siguiente fase de desarrollo, una vez finalizado el ciclo anterior, comienza el planteamiento del nuevo ciclo.

Durante cada ciclo de la espiral se contempla el análisis de riesgos, identificando las situaciones que pueden poner en riesgo el proyecto o incrementar su costo. El análisis de riesgos representa la misma cantidad de desplazamiento angular en cada etapa y el volumen transportado denota el incremento de los niveles de esfuerzo requeridos para el análisis de riesgo. Algunas de las bondades del modelo en espiral se mencionan a continuación:

- Se hacen explicitas las posibles alternativas para el logro de los objetivos.
- Se identifican los riesgos potenciales de cada alternativa y su posible solución.
- El modelo se puede adaptar a desarrollos de software y de mantenimiento del sistema sin establecer diferencias.

Al inicio de un nuevo proyecto de software la funcionalidad del sistema se proyecta mediante la especificación de requerimientos. Dicha especificación es la antesala de la fase del diseño, es por ello que para el desarrollo adecuado de un sistema de software la combinación de estos dos factores resulta ser la parte central que responde al propósito del sistema. Para dichos efectos el estándar de Especificación de Requerimientos IEEE 830, es considerado como la forma en que la abstracción de la funcionalidad del sistema se encuentra descrita en términos concretos para elaborar un adecuado documento de requisitos de una forma objetiva.

En 1996 la IEEE a través de su Subcomité de Normas de Ingeniería de Software (SESC) realizo una encuesta a través de internet contestada por 148 organizaciones al rededor del mundo basada en el uso de los estándares de la IEEE para la elaboración y desarrollo de proyectos de software.

Entre los resultados se destaca que el 37.14% de los encuestados respondió que las normas IEEE no están disponibles en su organización, otro 37.14% respondió que usan diferentes normas, un 11.42% dice que las normas IEEE no se ajustan a sus necesidades, el 5.72% respondió que el uso de las normas no se encuentra estipulado en el contrato de trabajo, otro 5.72% declararon que las normas no cubren lo que necesitan y el 2.86% menciona que el estándar es de mala calidad.

A su vez los encuestados mencionan el uso de diferentes normas de estandarización en sus organizaciones tales como las normas IEC/ISO, las ISO 9001, DOD Std 2167A y MIL-STD-498 (Land, S. 1997). Es claro que las organizaciones tienden a utilizar algún tipo de estándar internacional para la elaboración de proyectos de ingeniería de software aunque muchos de estos se enfoquen en el plano administrativo como es el caso de uso de las ISO 9001.

Por otro lado en la encuesta se les pide que mencionen el campo de aplicación en el que se encuentra inmersa la organización de los encuestados, obteniendo como resultado que un 22.39% está inmerso en el área de las tecnologías de la información, un 14.93% en el ejército y en un porcentaje menor 10.45% el sector aeroespacial, sin dejar de lado las áreas sociales, económicas y de educación.

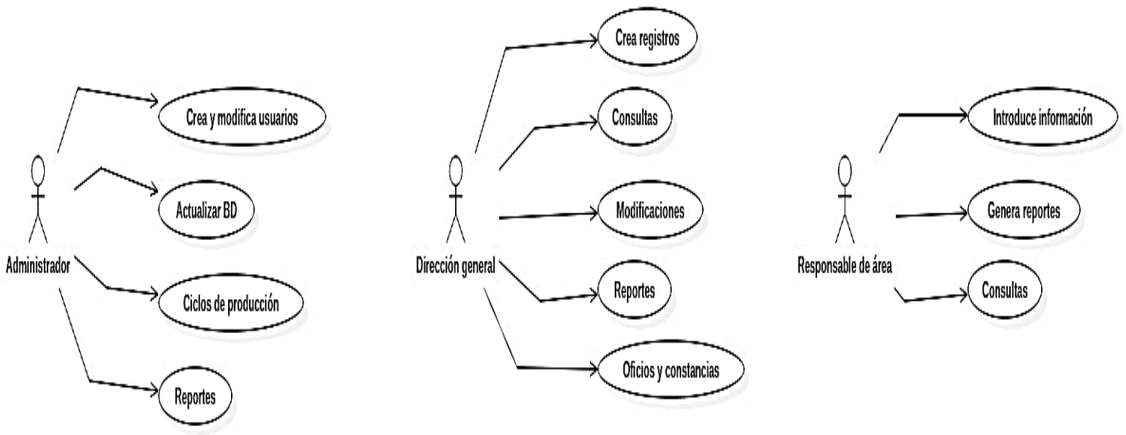
8.1 Caso de estudio

La propuesta de este artículo de Investigación se basa en el desarrollo y análisis de un caso de estudio de un Sistema de software para el Seguimiento y Control de las Actividades Desarrolladas por los Coordinadores (SiSCADC) de las diversas licenciaturas que se ofertan en una organización, el cual tiene como objetivo optimizar los procesos de recepción, registro y seguimiento de las actividades realizadas por parte de los responsables de ciertas áreas administrativas de la organización, ya que hasta el momento la gestión de los recursos necesarios para realizar las actividades en la planeación semestral y asegurarse del cumplimiento de planes y programas para el logro de metas institucionales es llevada a cabo por la dirección general, en dicha área no se cuenta con procedimientos estandarizados de recepción, revisión y retroalimentación de los planes y programas realizados por la administración.

De esta manera el desarrollo de software del SiSCADC se basa en el modelo de cascada y el estándar internacional para especificación de requerimientos IEEE830 con la finalidad de ubicar los elementos más importantes para el desarrollo del sistema, tener documentación que permita efectuar un mantenimiento preventivo y correctivo eficiente y proyectar los recursos asignados por la organización de forma eficaz.

El SiSCADC cuenta con tres perfiles operacionales, ya que por la naturaleza de la información y documentación es personalizado el acceso:

Figura 8.1 Perfiles operacionales



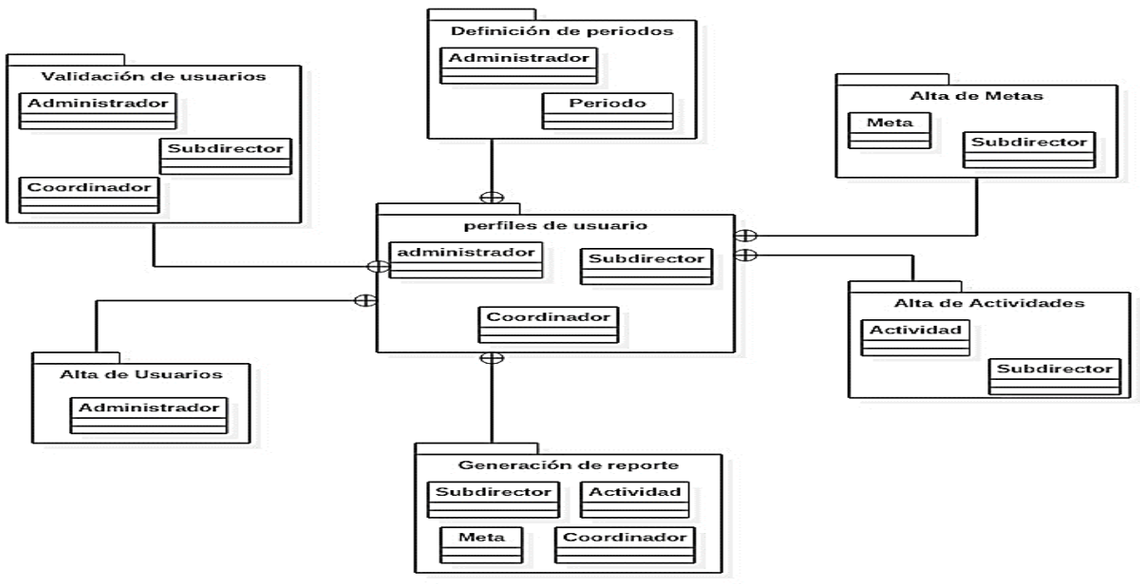
Fuente: Elaboración propia.

En el desarrollo del SiSCADC, el propósito del sistema es generar un instrumento colaborativo para el control y seguimiento de las actividades planeadas en cada ciclo de producción de la organización. En el apartado inicial de la plantilla del estándar IEEE830 quedan especificadas las problemáticas que se tienen dentro del proceso manual y las ventajas de la automatización de dicho proceso mediante el SiSCADC. En el segundo apartado de la plantilla se establece con claridad la funcionalidad de las perspectivas del sistema y las funciones de los perfiles operacionales, así como sus restricciones y dependencias con otros sistemas. En el tercer apartado se realizó un documento de especificación de requerimientos más concreto y objetivo, con el apoyo de los apartados anteriores.

Con la elaboración de la especificación de requerimientos, tomando como base el estándar IEEE 830 se tiene una entrada objetiva y clara en la fase de diseño, ya que al tener un soporte documental que avala los procesos y procedimientos del sistema, quedan explícitos los detalles para avanzar a la etapa siguiente, esto permite detectar errores antes de llegar a la etapa de implementación del sistema.

En la figura 8.1 es posible observar la arquitectura del sistema que ha sido modelada tomando en cuenta las funciones y perspectivas del producto, de acuerdo al apartado dos de la plantilla del estándar IEEE830. En la figura 8.2, el apartado tres, guiado por el apartado dos de la plantilla del estándar IEEE830 nos marca con claridad las relaciones entre los perfiles operacionales y las funciones del sistema.

Figura 8.2 *Diagrama de paquetes del SiSCADC*

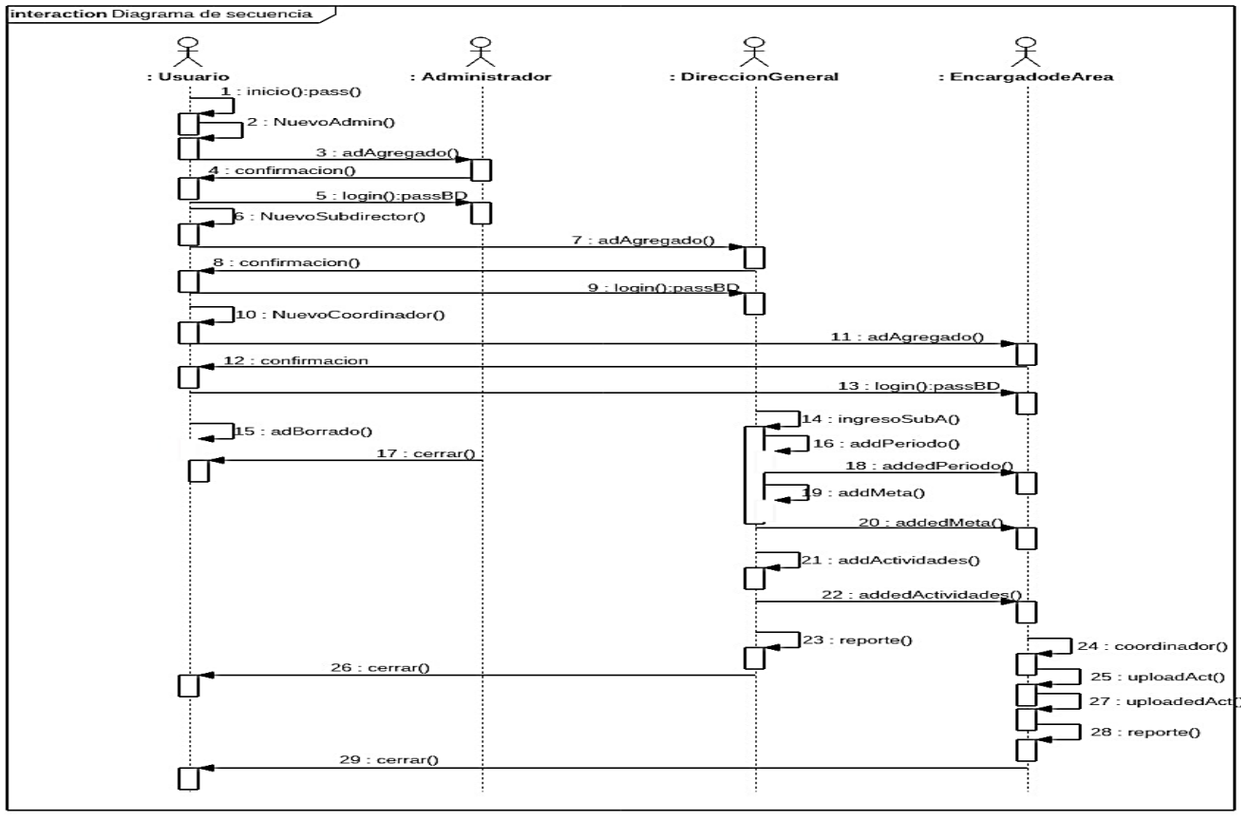


Fuente: Elaboración propia.

En la figura 8.2 es posible observar la arquitectura del sistema que ha sido modelada tomando en cuenta las funciones y perspectivas del producto, de acuerdo al apartado dos de la plantilla del estándar IEEE830. En la figura 8.3, el apartado tres, guiado por el apartado dos de la plantilla del estándar IEEE830 nos marca con claridad las relaciones entre los perfiles operacionales y las funciones del sistema.

De esta manera se puede observar que el uso de estándares internacionales abalados por instituciones como la IEEE que proporcionan calidad al momento de elaborar un proyecto de software, lo cual redunda en un adecuado uso de los recursos económicos y humanos de la organización, ya que las fases previas del proceso de desarrollo son formuladas exitosamente.

Figura 8.3 Diagrama de secuencias del SiSCADC



Fuente: Elaboración propia.

8.2 Trabajo a futuro

Realizar la fase de pruebas mediante el estándar IEEE 1059 que nos proporciona una guía para realizar planes de validación y verificación de las funciones del sistema en términos de trazabilidad. La Trazabilidad es la propiedad directamente relacionada con la funcionalidad del sistema y el diseño. La Trazabilidad es aquella propiedad que nos permite enlazar las especificaciones de los requerimientos en relación con la arquitectura y la implementación del sistema.

8.3 Conclusiones

DeMarco afirma que “no se puede controlar lo que no se puede medir” (DeMarco, T. 1982), lo cual nos da una idea acabada de la necesidad primordial para efectuar mediciones sobre un proceso determinado, en un proyecto de ingeniería de software. La ingeniería de software hace hincapié en la calidad y en la mejora del proceso productivo, en la confianza de que un buen proceso asegura un buen producto. Cada acción que pretenda medir algo, debe estar motivada por un objetivo particular o necesidad definida claramente. Dicho razonamiento se suma al principio de Gilb citado por Fenton: “los proyectos que no tienen objetivos claros, no arriban a metas claras" (Gilb, T. 1988). En este trabajo queda explicita la necesidad de contar con una metodología que nos permita adecuar, lo que se pretende medir al tipo de software que se pretende desarrollar dentro de un caso específico.

Es en este momento que se da una posible respuesta a la cuestión ¿Para qué estandarizar? Pues bien, el uso de estándares nos permite comprender lo que sucede en el desarrollo de un proyecto, también nos permite analizar y controlar acciones durante la ejecución de un programa y por último, el uso de estándares nos sirve para mejorar los procesos en el desarrollo de proyectos dentro de la ingeniería de software.

Como se mencionó anteriormente la especificación de riesgos es un paso esencial dentro del desarrollo de todo sistema de software que permite el entendimiento claro y objetivo del resto de las etapas del proyecto.

La existencia de un estándar como IEEE830 permite la coherencia en la especificación de requisitos y la adecuada redacción de los casos de uso, lo cual permite identificar de forma eficiente las necesidades del sistema. Recordemos que en los proyectos de software empresarial se tienen en cuenta métricas de esfuerzo, costo, capacidad y productividad, conocidos también como técnicas del COCOMO y COCOMO II, donde se debería considerar adicionalmente la calidad y la confiabilidad, tomando como base la satisfacción del usuario para el desarrollo de nuevos proyectos de ingeniería de software.

8.4 Referencias

Boehm B. (1988). A spiral model of software development and enhancement, Computer 1988 IEEE, 61-72.

Cass (2014). Top 10 Programming Languages. IEEE Spectrum.

DeMarco T. (1979). Structured analysis and systems specifications. Prentice Hall.

Fenton N. (1991), Software Metrics. A rigorous and practical approach, PWS Publishing Company. Boston.

Galvis A. (1996). Software educativo multimedia aspectos críticos no seu ciclo de vida. Revista Brasileira de Informática no Educaçao, Sociedad Brasileira de Computaçao. Disponible en http://www.janus.ufse.br:1085/revista/nr1/galvis_p.htm.

IEEE (1991). Standard for Developing Software Life Cycle Process. IEEE Std. 1074-1991 Nueva York. IEEE Computer Society.

ISO (1994). ISO/IEC 12701-1. Software Life-cycle Process.

ISO (1995) 12207-1. Information Technology-Software Life Cycle Processes. International Standard Organization. Suiza.

J. Juzgado, N. (1996). Procesos de construcción del software y ciclos de vida. España: Universidad Politécnica de Madrid.

Ktcheman y Walter (1989). Citado en Fenton (1991). Software Metrics. A rigurous and practical approach, PWS Publishing Company, Boston.

Maddison R. N. (1983). Information System methodologies. Wiley Henden.

Mc Cracken D. y Jackson A. (1982). Lifecycle concepts considered harmful: ACM, Sigsoft Software Engineering Notes. 7 (2), 29-32, citado en Piattini (1996).

McCall J. (1977). Factors in software quality. Vols. I, II y III. NTIS; Roma, citado en Piattini (1996).

Naur P. Y Randell B. [eds.] (1969). Software engineering: A report on a Conference sponsored by the NATO Science Committee. Citado en Pressman (1993).

Piattini M. (1996). Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión. Madrid: Rama.

Pressman R. (1993). Ingeniería de Software. Un enfoque práctico. Mc Graw Hill.

Land S. (1997). Results of the IEEE Survey of Software Engineering Standards Users. IEEE Computer Society.

Sigwart C. et al. (1990). Software Engineering: a project oriented approach. Franklin, Beedle y Associates, Inc., Irvine, California, citado en Piattini (1996).

Sommerville, I. (2011). Ingeniería de Software (Novena ed.). Pearson Education.