Universidad Autónoma del Estado de México

Centro Universitario UAEM Valle de Chalco

Importancia de la Calidad de Software en la Fase de Análisis

ENSAYO

QUE PARA OBTENER EL TÍTULO

INGENIERO EN COMPUTACIÓN

P R E S E N T A

Heriberto Santiago López

ASESOR: M.E Horacio Jesus Tacubeño Cruz

Revisora: Dra. Maria de Lourdes Lopez Garcia

Revisor: Mtro. Rodolfo Melgarejo Salgado

VALLE DE CHALCO SOLIDARIDAD, MÉXICO 27 SEPTIEMBRE 2021

CUVCH

Importancia de la Calidad de Software en la Fase de Análisis

Contenido

Índice de tablas

Índice de Figuras

1. Introducción

Al encontrar errores en un software, el usuario suele comúnmente llamarlos “bugs”, lo que significa que el resultado puede ser el no deseado. En el ámbito de la ingeniería de software, un defecto o error realmente abarca todo un proceso productivo en el desarrollo de dicho software. Normalmente, los errores comienzan en las etapas tempranas del desarrollo, por lo que debe existir una estructura robusta de control en las diferentes fases para identificar a tiempo un error y corregirlo, de tal forma que no se propague a las fases siguientes. En este sentido, es importante conocer esta área tan importante que forma parte de la Ingeniería de Software.

Según Sommerville, Ingeniería de Software es una disciplina que comprende todos los aspectos del desarrollo de software, incluyendo los requisitos de las actividades de ingeniería, modelos de proceso y de técnicas de estimación (Sommerville I. , 2005).

A través de la Ingeniería de Software se buscan dos factores esenciales:

1. Disciplina de la ingeniería. En este punto se aplican teorías, métodos y herramientas para resolver los problemas de un software.

2. Producción de software. No solo comprende los procesos técnicos, sino también actividades como la administración del proyecto de software.

Dice Pressman que una aplicación de software se desarrolla a través de un proceso. No es algo que se fabrica a partir de materias primas, o se monta a partir de piezas más pequeñas, el software muestra esta característica especial en la comparación con otros tipos de productos, a saber, que no se fabrica en el sentido clásico, pero se desarrollan a través de un proceso de ingeniería (Pressman R. S., 2005).

Campos hace referencia que la Ingeniería de Software ofrece enfoques sólidos para aumentar las posibilidades de que los objetivos de negocio se cumplan en términos de tiempo calidad y características. Las organizaciones de hoy en día tienen el reto de llevar a cabo sus actividades de manera productiva con la calidad y el cumplimiento de la planificación estratégica (Calidad en el Desarrollo de Sotfware, 2007).

De tal forma que la Ingeniería de Software garantizará, si se sigue correctamente el proceso, un producto (un software) de calidad, en los tiempos establecidos. Preguntas importantes al respecto son ¿las fases que comprende el desarrollo de software garantizan que éste sea de calidad?, ¿cómo se puede medir al software de tal forma que sea calificado como un buen producto y si se pudiera medir?, en las fases del desarrollo, ¿cómo se puede evaluar el software para garantizar que el tiempo establecido para desarrollarlo se cumpla de acuerdo con los recursos humanos y tecnológicos?, entre otras.

La manera de comprobar la calidad y la eficiencia de un software desarrollado es a través de las métricas de software. La medición del software empieza en los años 70's en Estados Unidos y Canadá. En el primero, los grupos y actividades de medición de software fueron establecidos desde la mitad de los años 70's. La medida de software más antigua es el *Loc medida* (Número de Líneas de Código), que contiende y se utiliza hasta hoy. En 1983, Boehm hizo uno de los primeros intentos de medir formalmente la productividad usando líneas de código. Propuso medir la productividad con el parámetro de hombre-mes y sugirió las tasas de códigos típicos para realizar comparaciones (Boehm B. , 1978).

Por su parte, Halstead se basa en la medición de código fuente. Propone que la estimación del esfuerzo y tiempo de programador se puede expresar como una función de conteo del operador y número de operando. El método de

Halstead ha sido utilizado por muchas organizaciones, entre ellas IBM y General Electry Company, entre otras (Las Metricas de Halstead, 2013).

Las dos métricas mencionadas usan líneas de código, es decir, ya que el diseño ha sido implementado. Sin embargo, surge también la necesidad de medir la fase de diseño, las cuales fueron propuestas por McCabe.

En los años 70's McCabe deriva una medida de complejidad del software en la teoría de grafos, usando la definición del número ciclomático. El cual es conocido como el número mínimo de caminos en el software. Dependiendo del número mínimo de caminos determina la complejidad ciclomática a través de su diagrama de flujo (McCabe, Richards, & Walters, 1970).

En 1986, en el Reino Unido se inició un proyecto de investigación (Alvey-Project SE / 069) titulado "Software-Estructurado Basado Medición". En este se pretende construir el modelado formal, el análisis y la medición de la estructura del software.

Por otra parte, las medidas para determinar la calidad de software tienen una gran importancia, ya que consiste en asignar un valor a una entidad de software, para esto existen las métricas que miden complejidad ciclomática, líneas de código (LOC) y la complejidad de Halstead las cuales ya fueron mencionadas y su principal función.

En este contexto, nace la necesidad de perfeccionar la estimación de la calidad de un software. Así, surge el software Quality Observatory for Open Source Software (SQO-OSS), en el cual, se desarrolló un conjunto de herramientas de evaluación de software con las que se puede analizar y obtener la calidad del código fuente y probar su capacidad para su despliegue empresarial. Estas herramientas estiman la calidad del producto (Informe Final - SQO-OSS, 2010).

Dada la importancia de la medición del software durante y al final de su desarrollo, en este trabajo se realiza una investigación respecto a la calidad del software, para lo cual, se presenta en la sección del desarrollo, definición, antecedentes y clasificación del software, así como los procesos y modelos en la Ingeniería de software. Además, se presentan las diferentes métricas que se pueden realizar al software y se profundiza en las métricas de calidad, que es la parte más importante en esta investigación. Termina el documento con una discusión sobre el tema y el compromiso que se debe tener para desarrollar un software con alta calidad.

2. Desarrollo

En esta sección se presentan las definiciones necesarias para entender el funcionamiento del software hoy en día y la evolución desde sus inicios, para entender la importancia que tiene una métrica de calidad en el software.

2.1 Antecedentes del software

La historia del software, como primera teoría fue propuesta por Alan Turing en su ensayo en 1935, sobre los números computables. El termino software fue utilizado por primera vez por John W. Tukey en 1958 (Tukey, 1957)

El campo del software era tan nuevo que no se tenía un tiempo estimado de la duración de cada proyecto. Por otro lado, en cuestión de hardware en esos años era muy específico para la aplicación, los lenguajes en los cuales se realizaba el desarrollo de software eran principalmente, los siguientes:

Fortran: Es un lenguaje de programación de alto nivel, el cual está especialmente adaptado al cálculo numérico y la computación científica de desarrollo especialmente por IMB en 1956 (McCraken, 1997).

Cobol: fue creado en el año de 1959 con el objetivo de tener un lenguaje de programación universal que pudiera ser utilizado desde cualquier ordenador y ser compatibles entre sí mismos (Ceballos Sierra, 1997).

Durante los primeros años de la historia del software, el desarrollo se realizaba virtualmente sin ninguna planificación, los programadores trataban de realizar las según las necesidades y con gran esfuerzo la mayoría del software era realizado por la misma personas u organización la cual escribía, ejecutaba y si fallaba lo depuraba, es decir, sobre tareas en específico.

El desarrollo de software se generó en los inicios de 1960, los cuales buscaban nuevas formas de desarrollo y métodos que las grandes empresas exigían en la relación a la demanda de sus clientes. A lo largo de la historia, las etapas del software se pueden dividir en las siguientes (Braude, 2003).

1. Primera etapa (1950-1965): el software se encontraba en su infancia, en los primeros años el propósito general es el hardware, por otra parte, el software se diseñaba a medida de cada aplicación. Un punto muy importante en la historia del software es la crisis del software la cual se vivió de 1960 a 1980, que identifica muchos de los problemas del desarrollo de software. Muchos proyectos de software sobrepasaron el presupuesto y el tiempo estimado. Algunos proyectos causaron daños en la productividad, hoy en día se utilizan términos de crisis de software a su incapacidad de contratar programadores suficientemente calificados.
2. Segunda etapa (1965-1975): se implementa la multiprogramación y los sistemas multiusuario introdujeron conceptos de interacción hombre – máquina, esta etapa se caracteriza por el establecimiento del software la cual se desarrollaba para obtener una amplia distribución en el mercado del software.
3. Tercera etapa (1975-1985): se caracteriza por la implementación de microprocesadores. El microprocesador ha producido un extenso grupo de productos inteligentes, en esta etapa comienza el desarrollo de software para redes y comunicaciones.
4. Cuarta Etapa (1985-2000): se implementa la tecnología orientada a objetos la cual se enfoca en el desarrollo de software más convencional en las áreas de las aplicaciones.

2.1.1 Definiciones del software

Diversos autores han definido al software, por lo que, se presentan algunas de ellas a continuación:

Definición 1. Software

Conjunto de los programas informáticos, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación. (Radatz Chairperson, Glossary of Software Engineering Terminology, 1993)

Definición 2. Software

Toda la información procesada por los sistemas informáticos programas y datos (Tukey, 1957).

Definición 3. Software

Programas de cómputo que cuando se ejecutan proporcionan las características, función y desempeños buscados, estructuras que permiten que los programas manipulen en forma adecuada la información (Pressman R. , 1993).

Definición 4. Software

Programas de cómputo y documentación asociada. Los productos de software se desarrollan para un cliente en particular o para un mercado en general (Sommerville I. , 2005).

Por lo que se puede deducir de las diferentes definiciones que el software es un conjunto de procedimientos y modelos que permiten llegar a un objetivo, de tal forma que puede realizarse una clasificación dependiendo de la finalidad para la cual fue creado.

2.1.2 Clasificación del software

Dependiendo de la finalidad del software, éste ha sido clasificado como sigue:

Software de sistema: es el programa encargado de la ejecución de todas las aplicaciones necesarias para que un sistema opere correctamente.

Consiste en un software que funciona como base para controlar e interactuar con el hardware y otros programas tenemos los siguientes ejemplos de sistemas de software:

- Sistema operativo Windows: de los más populares que existen, la compañía fue fundada en 1975 por William H. Gates III y Paul Allen, inicia con su sistema operativo MS-DOS posteriormente de varias versiones hoy en día se tiene en operación Windows 10 con una interfaz gráfica amigable al usuario (M & Flynn, 2011).

- Sistema operativo Linux: es un conjunto de sistemas operativos libres de multiplataforma, multiusuario, y multitarea basados en Unix, se considera software libre fue creado por Richard Stallman en 1983.Linux puede trabajar tanto de forma gráfica como de modo consola normalmente el modo consola se utiliza para los servidores (Gedda, 2004).

- Sistema operativo Mac (Catalina 10.15) su significado (Sistema Operativo de Macintosh) el cual fue creado por Apple para su línea de computadoras Macintosh. Fue desarrollado por Bill Atkinson, Jef

Raskin y Andy Hertzfeld, la primera versión fue lanzada en el año de 1985 con su versión System 7.5.1. (M & Flynn, 2011).

Como es bien sabido, hoy en día estos son los sistemas operativos más conocidos para una computadora, pero también se tienen sistemas operativos móviles, a continuación, mencionaremos algunos más comunes:

- Sistema Operativo móvil (Android): Fue desarrollado por Google basado en el kernel de Linux, el cual fue diseñado para dispositivos móviles con pantalla táctil como tabletas y relojes. Android fue presentado en los años 2007, según un estudio realizado por Symantec se demuestra la comparación con IOS, donde indica que Android es sistema explícitamente menos vulnerable, la versión más actual es Android 12.

- Sistema Operativo móvil (iOS): Es desarrollado exclusivamente para dispositivos como iPhone, iPod touch y iPad. Este tipo de sistema operativo no permite la instalación IOS en hardware de terceros, fue lanzado en año 2007 la actualización es mediante iTunes hoy en día contamos con la versión iOS 14.5.

Software de programación: son las herramientas que permiten el desarrollo de programas informáticos utilizando diferentes lenguajes programación o bases de datos. Algunos ejemplos de este tipo de software son los siguientes:

- Microsoft Visual Studio: Es un entorno de desarrollo integrado (IDE) el cual es compatible con los sistemas operativos Windows, Linux, Visual Studio permite a los desarrolladores crear sitios y aplicaciones web es compatible con la plataforma .NET, su fecha de lanzamiento fue en año de 1997 fue desarrollado por Microsoft hoy en día la última versión es la 16.4.3 (Flores, 2010).

- Eclipse: fue desarrollado por IBM, está compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar “Aplicaciones de Cliente Enriquecido”, también cuenta con (IDE) el cual proporciona la funcionalidad y módulos. Eclipse provee al programador con frameworks muy ricos para el desarrollo de aplicaciones gráficas, software, aplicaciones web (DÉLÉCHAMP, 2018)

- Notepad++: Es un editor de texto y se considera como software de programación ya que es capaz de soportar varios lenguajes de programación. Entre sus características se encuentran que tiene un coloreado para cada expresión o sintaxis, dependiendo el lenguaje de programación.

- CPython: Funciona como software de programación de tipo interprete de byteCode el cual se encarga de analizar y ejecutar otros programas. Tiene una interfaz de funciones foráneas para varios lenguajes como por ejemplo C, C++ (Braude, 2003).

Software de aplicación: es un conjunto de programas el cual tiene como propósito facilitar las tareas específicas para cualquier medio informático. A continuación, se mencionarán algunos ejemplos.

- Google Chrome: Es unos de los aplicativos más utilizados, su principal función es ser un navegador web de que fue desarrollado por Google, se encuentra disponible gratuitamente y es utilizado en los diferentes sistemas operativos ya antes mencionados (Saiz, 1998).

- Adobe Photoshop: es un software de aplicación de tipo editor de gráficos el cual fue creado por Adobe Systems Incorpotated. Es usado principalmente para el retoque de fotografías y gráficos (Álvarez, 2019).

- Microsoft Word: Esta orientado al proceso de textos, fue creado por Microsoft, utiliza un formato muy nativo el cual se llama DOC, este formato se ha convertido en el estándar con el que se pueden transferir documentos de textos (NORO, 2019).

- Skype: Es un software de aplicación su principal objetivo es establecer comunicaciones telefónicas o videoconferencias empleando la transmisión de datos de internet (NORO, 2019)

- Media Monkey: se trata de un reproductor de multimedia el cual maneja archivos de audio y video, permite organizar, etiquetar y reproducir archivos, es compatible con diversos sistemas operativos y fue desarrollado por Ventis Media Inc.

Software malicioso: También conocido como Malware, el cual funciona instalando programas indeseados en tu computadora o dispositivo móvil. Estos programas instalados afectan al funcionamiento y en algunos casos extraen información, mencionaremos algunos ejemplos de software malicioso:

- Troyano: Es un software malicioso que se presenta como un programa legitimo e inofensivo, pero al ejecutarlo brinda un atacante acceso remoto al equipo infectado (Soto, 2017).

- Gusanos: Los gusanos informáticos se propagan de computadora en computadora, tiene la capacidad de propagarse sin la ayuda de una persona. Lo más peligroso de los gusanos es su capacidad de replicarse en el sistema operativo por lo que la computadora podría enviar cientos de miles de copias de sí mismo (Soto, 2017).

- Keylogger: se encarga de registrar las pulsaciones que se realizan en el teclado y memorizarlas en un fichero o en su caso enviarlas a través de Internet (Soto, 2017).

- Ransomware: Restringen el acceso a determinadas partes o archivos del sistema infectado, algunos tipos de ransomware cifran los archivos del sistema operativo (Soto, 2017).

- Phishing: se caracteriza por intentar adquirir información confidencial de forma fraudulenta (como puede ser una contraseña, información detallada sobre tarjetas de crédito a alguna información bancaria) estos tipos de software son enviados comúnmente por correo electrónico (Soto, 2017).

Los puntos anteriores tienen suma importancia ya que es necesario entender el significado de software sus antecedentes y su clasificación, hoy en día existe una disciplina la cual se encarga de mencionar los puntos más importantes del software y que en la siguiente sección se presentara a detalle y los procesos establecidos para el desarrollo.

2.2 Ingeniería de Software

La ingeniería de software es considerada una disciplina, que ofrece métodos y técnicas para desarrollar y mantener software de calidad, se enfoca desde la primera etapa de especificación del sistema hasta la fase del mantenimiento para resolver problemas de todo tipo. Se relaciona con diferentes áreas de la informática y ciencias de la computación, tales como construcción de compiladores, sistemas operativos o desarrollos en Internet.

La ingeniería de Software es importante por dos razones (Sommerville I. , 2005):

1. Actualmente la sociedad y los individuos se apoyan en sistemas de software, para lo cual requieren una producción económica y rápida de sistemas confiables.

2. El uso de métodos y técnicas de ingeniería de software resulta ser más barato a largo plazo, que solo diseñar los programas como si fueran un proyecto de programación personal.

El software en la actualidad va en crecimiento se puede notar en la aplicaciones y sistemas que utilizamos hoy en día, pero sus características y funciones son diferentes a los requerimientos del usuario final.

En la actualidad grandes equipos de personas trabajan en la creación de programas por la complejidad de estos nuevos sistemas, se sabe que los negocios empresas y gobiernos dependen más del software las cuales les ayudan a tomar decisiones estratégicas y tácticas, sin el software se tendrían fallas catastróficas.

2.2.1 El proceso del software

Se entiende el proceso como los pasos a seguir para alcanzar el objetivo deseado y en esta etapa del software permite mantener el control, estabilidad y organización para las actividades realizadas, el resultado es un software de calidad. El proceso de software efectivo da la oportunidad de incrementar la productividad al desarrollar el software, para lo cual, es importante distinguir los siguientes conceptos:

- Proceso de Software: Conjunto de actividades y resultados asociados que conducen a la creación de un producto software (Sommerville I. , 2005).

- Ciclo de Vida del Software: Aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software (Delgado, 2018).

- Modelo de ciclo de vida: 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 los requisitos hasta la finalización de su uso (ISO/IEC/IEEE, 2017)

Los flujos del proceso que se utilizan en el desarrollo de software describen cómo se van a ejecutar las actividades y la duración de cada una de ellas:

Flujo de Proceso Lineal: Para este punto la característica principal es que empieza desde la comunicación y termina hasta el despliegue, como se muestra en la figura 1.

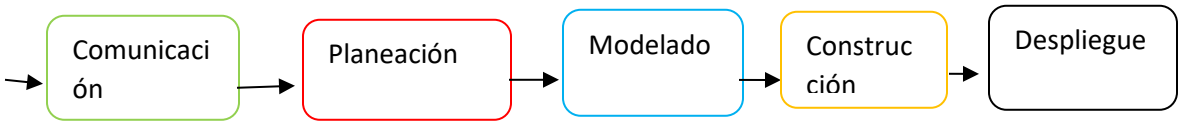

Figura 1. Proceso lineal (Elaboración propia)

Flujo de proceso iterativo: La característica principal de este proceso es que se repiten las actividades una y otra vez mientras sea necesario para alcanzar la siguiente actividad.

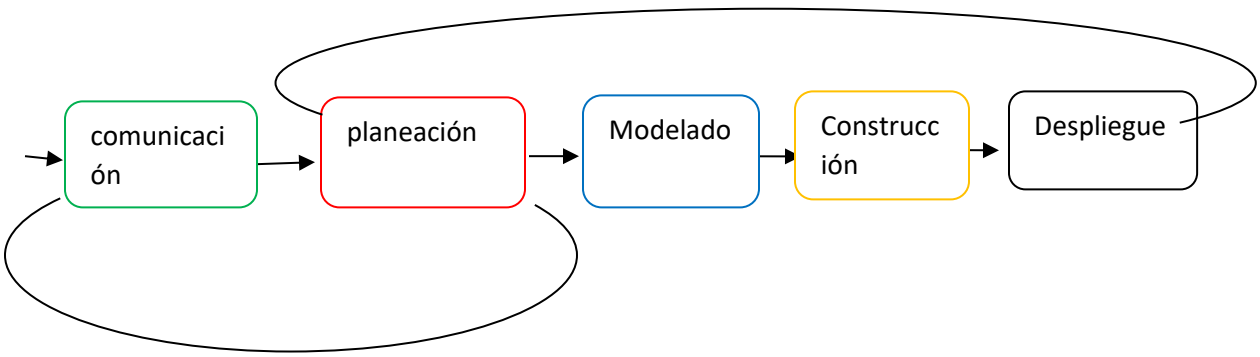

Figura 2. Proceso Iterativo (Elaboración Propia)

Flujo de proceso Evolutivo: En este proceso (Figura 3.) las actividades se realizan de forma circular y en cada círculo significa la mejora del producto.

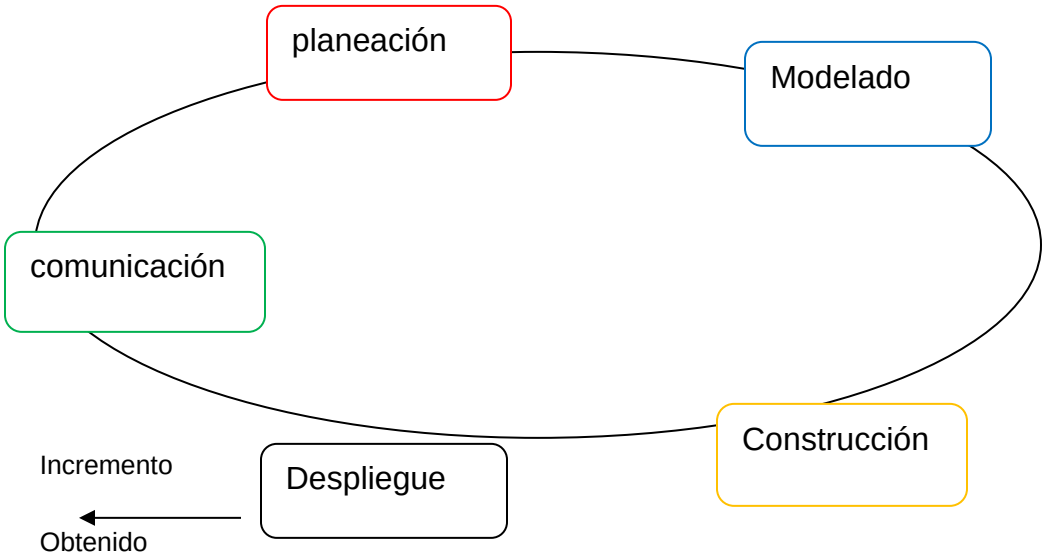

Figura 3. Proceso Evolutivo (Elaboración Propia)

Flujo de proceso en paralelo: Se ejecuta una o dos actividades en paralelo, es decir al mismo tiempo que se ejecuta otra actividad (Figura 4.).

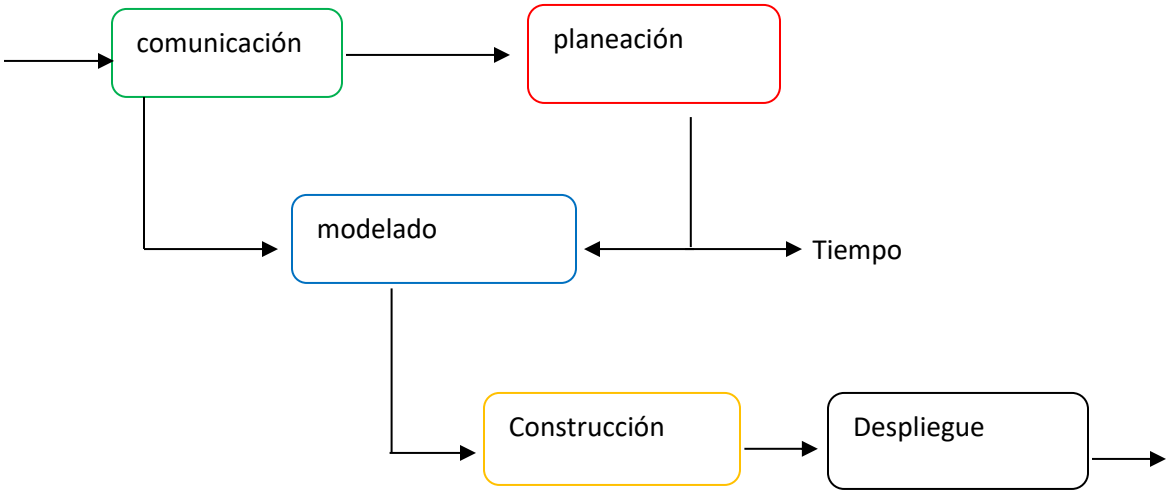

Figura 4. Proceso en Paralelo (Elaboración Propia)

El principal objetivo del flujo de proceso es representar de forma gráfica el estado actual del proceso para obtener un enfoque que facilitara su optimización. Para que de esta forma sea más eficiente, a continuación, se mencionaran algunos ejemplos más detallados de los modelos que actualmente son más utilizados para el proceso del software.

2.2.2 Modelos del Proceso de Software

Un modelo de proceso es un conjunto de actividades o tareas las cuales se realizan con la finalidad de alcanzar el desarrollo completo de un proyecto de software, se mencionarán algunos ejemplos de modelos que son más utilizados

Modelo de Cascada: este modelo fue creado por Bennington 1956 y modificado por Royce en 1970. Pressman lo presenta como ciclo de vida clásico como se ilustra en la Figura 5. Se denomina modelo en cascada porque su característica principal es que no se comienza con un paso hasta que no se ha terminado el anterior. El modelo en Cascada establece que el software debe ser construido, rigurosamente, a través de una transformación sucesiva de documentos, siguiendo una estrategia lineal de desarrollo. Primero saber qué se quiere y después, cuando se conozca todo lo que se quiere, empezar a construirlo (Braude, 2003).

Modelo de Espiral: este modelo fue propuesto originalmente por Boehm en 1988, es un modelo de proceso de software evolutivo que ha sido desarrollado para cubrir las mejores características tanto del ciclo de vida clásico, como de la creación de prototipos, añadiendo al mismo tiempo un nuevo elemento: el análisis de riesgo, véase Figura 6. Proporciona el potencial para el desarrollo rápido de versiones incrementales del software. En este modelo el software se desarrolla en una serie de versiones incrementales. Durante las primeras iteraciones, la versión incremental podría ser un modelo en papel o un prototipo. En las últimas

iteraciones, se producen versiones cada vez más completas del sistema diseñado (Braude, 2003).

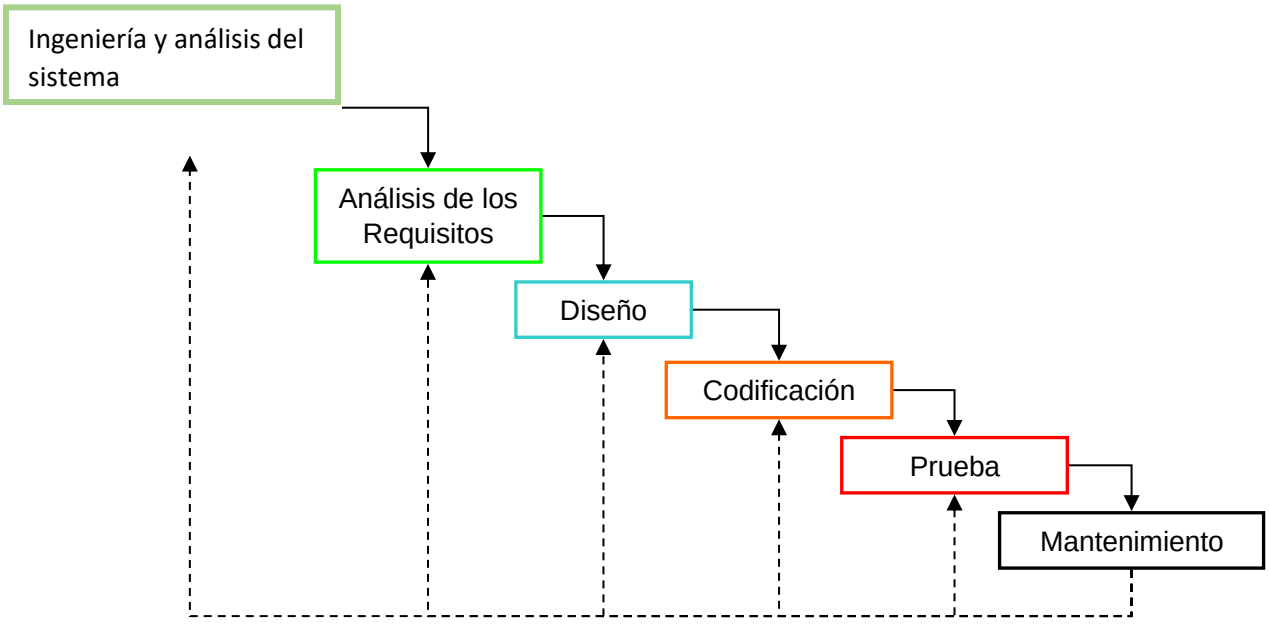

Figura 5. Modelo de Cascada (Elaboración Propia)

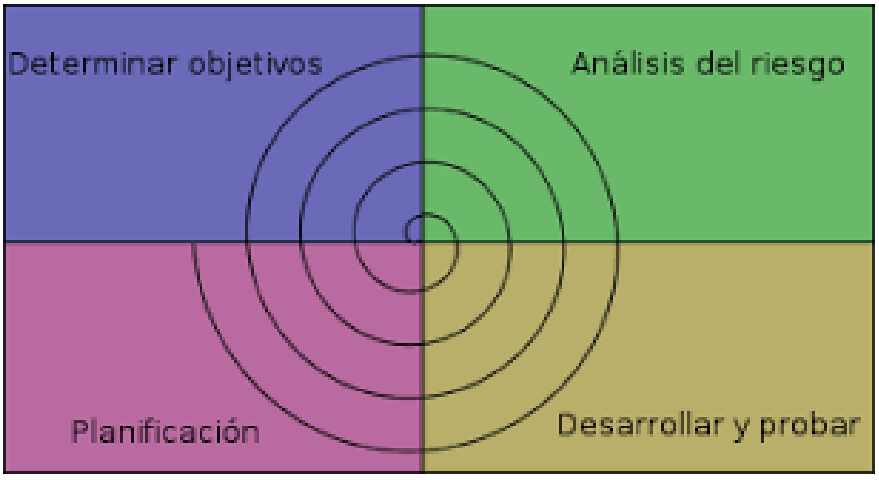

Figura 6. Modelo Espiral (Pressman R. , 1993)

Modelo Scrum: este modelo es un proceso de gestión que ayuda a reducir la complejidad en el desarrollo de productos para satisfacer las necesidades de los clientes. La gerencia y los equipos de scrum trabajan alrededor de requisitos y tecnologías para entregar productos funcionando de manera incremental. Como se puede observar en la Figura 7., esta modelo considera reuniones para retroalimentar el proceso que garantice identificar el trabajo desarrollado bajo lapsos cortos de tiempo y ofrecer entregas parciales del producto (Pressman R. S., 2005).

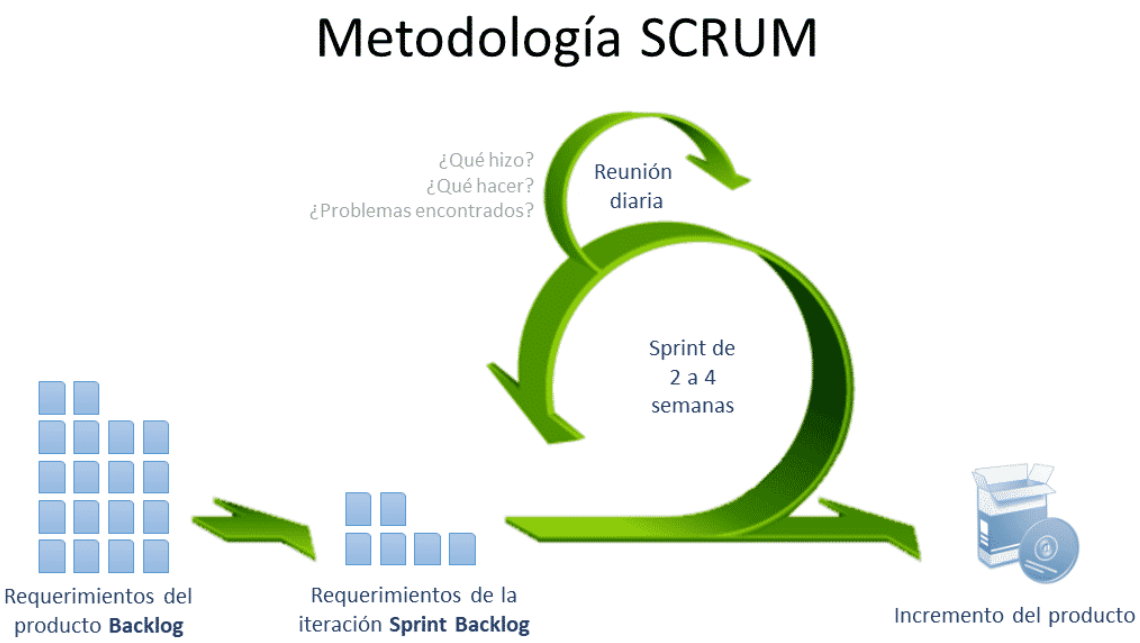

Figura 7. Modelo Espiral (Fillottrani, Pablo R., 2007)

Dado lo anterior, la ingeniería de software es una disciplina la cual integra procesos, métodos y herramientas para el desarrollo de software y que es muy importante hoy en día tener en cuenta el punto de las métricas de software.

2.3 Métricas de software

Se define métricas a los diferentes tipos de medidas que son aplicables en todo el ciclo de vida del desarrollo de software, desde que inicia se debe estimar costos las cuales son aplicadas por el ingeniero de software. También se debe de tener en cuenta que las métricas de software no son absolutas ni son comprobaciones científicas, sin embargo, a través de una medición se puede establecer un criterio en el desarrollo del software. Se define como una medida cualitativa, del grado en que un sistema, componente o proceso posee un atributo determinado (Delgado, 2018).

Pressman define a las métricas de software como:

> "La aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo del software y sus productos para suministrar información relevante a tiempo, así el administrador junto con el empleo de estas técnicas mejorara el proceso y sus productos".

En general la medición persigue tres objetivos fundamentales, ayudar al desarrollador a entender que ocurre durante el desarrollo y mantenimiento, permitir el control de lo que ocurre en los proyectos y poder mejorar los procesos y los productos (Fenton N. E., 1997).

2.3.1 Antecedentes de las métricas de software

Las métricas de software realmente fueron utilizadas a principios de los años ochenta con el trabajo utilizado por dos académicos de la universidad de Iowa, Kafura y Sally Henry. Ellos trataron de investigar el diseño del sistema métrico que podría ser utilizado para predecir factores tales con la facilidad del mantenimiento del software.

Entre las razones para medir un producto se pueden mencionar las siguiente:

- Indicar la calidad del producto.
- Evaluar la productividad del ingeniero de software.
- Evaluar los beneficios de productividad y calidad derivados de uso de nuevos métodos empleados en ingeniería de software.
- Establecer una línea de base para la estimación del producto.
- Ayudar a justificar el uso de nuevas herramientas o de formación adicional.

2.3.2 Clasificación de las Métricas

Las métricas de software de clasifican según sea el criterio con el cual son medidas, se pueden clasificar en medidas directas y medidas indirectas. En las primeras se incluyen las medidas de costo y esfuerzo aplicados, las líneas de código (LOC), la velocidad de ejecución y el tamaño de memoria. Mientras que en las primeras se clasifican las métricas de funcionalidad, calidad, complejidad, eficiencia fiabilidad y facilidad de mantenimiento.

A continuación de mencionaran algunos ejemplos de la clasificación de métricas directas.

a) Métricas orientadas al tamaño

COCOMO: Este modelo fue desarrollado por Barry W. Boehm a finales de los años 70, es un modelo matemático el cual se basa en la estimación de costos de software, su objetivo principal es medir el tamaño del proyecto en líneas

de código (Boehm B. , 1978). La Figura 8., ilustra el flujo del proceso para esta métrica.

Figura 8. Modelo de COCOMOC (Elaboración Propia)

Algunos puntos por los cuales se caracteriza la métrica de COCOMO son las siguientes:

- Se miden los costos del producto de acuerdo con su tamaño y otras características, pero no a la productividad.
- La medición por líneas de código no es válida para orientación a objetos.
- Utilizar este tipo de método puede resultar un poco complicado, en comparación con otros métodos orientadas al tamaño.

LOC: (Líneas de Código) es una métrica estándar que se utiliza para tratar de determinar el tamaño del desarrollo del software el cual se basa en medir líneas de código. Un atributo de esta métrica es que nos proporcionan una idea muy buena del tamaño del desarrollo del software y pude ser interesante para muchas cosas (Pressman R. S., 2010).

Métrica Bang fue desarrollada por De Marcola, la métrica bang se encarga de medir el tamaño del sistema. Para calcularla, el desarrollador de software debe de evaluar primero un conjunto de primitivas. Las primitivas se denominan evaluando el modelo de análisis y desarrollando cuentas para los siguientes elementos (Pressman R. S., 2005).

- Primitivas Funcionales (PFu): aparecen en el nivel inferior de un diagrama de flujo de datos.

- Elementos de Datos (ED): Los atributos de un objeto de datos, los elementos de datos son datos no compuestos y aparecen en el diccionario de datos.

- Objetos (OB): Objetos de datos.

- Relaciones (RE): Las conexiones entre objetos de datos.

- Estados (ES): El número de estados observables por el usuario el diagrama de transición de estados.

b) Métricas basadas en la función

El principal objetivo, de este tipo de métricas, está basado en predecir el tamaño y la complejidad de un sistema, que se va a obtener de un modelo de análisis el cual se representa a través de un diagrama de flujo de datos. Se necesitan los siguientes puntos para poder calcular este tipo de métricas basadas en la función (referencia):

- Entradas de usuario: son entradas que nos proporcionan diferentes datos de la aplicación.

- Salidas de usuario: son reportes, pantallas o mensajes de error que proporcionan información. Los elementos de un reporte, no se cuentan de forma separada.

- Peticiones de usuario: se refiere a una entrada interactiva que produce la generación de alguna respuesta del software en forma de salida.

- Archivos: son componentes que pueden ser parte de una base de datos o independientes.

- Interfaces externas: son los archivos que se usan para transmitir información a otro sistema.

Las métricas antes mencionadas, se clasifican de tipo directa el objetivo principal es medir los costos, el esfuerzo aplicado, líneas de código producidas, velocidad de ejecución y el tamaño de memoria en un determinado periodo en el desarrollo de software.

A continuación, se mencionará ejemplos de métricas indirectas:

a) Métricas de productividad

El principal objetivo de las métricas de productividad es el rendimiento de las funciones del desarrollo de software para poder obtener:

- Tamaño de la cartera de aplicación
- Tendencias de crecimiento de dicha cartera
- Ratio de productividad de un proyecto
- Ratio de duración del desarrollo de un proyecto
- Productividad global de una empresa

b) Métricas de Complejidad

Comúnmente conocida como complejidad ciclomática que originalmente fue desarrollada por Thomas McCabe el principal objetivo de esta métrica es determinar la fiabilidad y mantenimiento de los sistemas de software, las métricas de complejidad también nos ayudan en la parte de del diseño, en las pruebas y mantenimiento del software las ventajas de reducir la complejidad son las siguientes (Braude, 2003).

- Facilitar el mantenimiento del código
- Simplificar la refactorización
- Verificar la calidad del sistema desarrollado
- Hacer más fácil las etapas del proyecto

En esta clasificación son todas las métricas de software que tienen como parámetro especifico la estimación de la complejidad, los puntos que se evalúan son los siguientes:

- Volumen
- Tamaño
- Costo (Estimación)
- Diseño de Software

c) Métricas de Calidad

Son todas las métricas que se relacionan en el punto de calidad del software tales como (Braude, 2003).

- Exactitud
- Estructuración
- Pruebas
- Mantenimiento
- Reusabilidad

d) Métricas de Competencia

Son todas las métricas que miden las actividades de los programadores con respecto a los siguientes puntos (Braude, 2003).

- Certeza
- Rapidez
- Eficiencia
- Competencia

d) Métricas de Desempeño

Son todas las métricas que miden la parte de la conducta de los módulos y sistemas de un software, generalmente tienen que ver con la eficiencia de ejecución, tiempo, almacenamiento, complejidad de algoritmos computacionales (Braude, 2003)

En este trabajo se tienen como principal interés las métricas de calidad, ya que utilizarlas implica obtener un software con las características deseables para definirlo como un “buen software”, por lo cual, en la siguiente sección se detallará sobre las métricas para la calidad de software.

2.4 Métricas para la Calidad del Software

La calidad es la suma de todos aquellos aspectos o características de un producto o servicio, que influyen en su capacidad para satisfacer las necesidades de los usuarios. La satisfacción del usuario está determinada por la diferencia entre la calidad percibida y la calidad esperada, cuando este hace uso de un producto o servicio.

El principal objetivo de la ingeniería de software es producir un sistema aplicación o producto de calidad, por lo cual se convierte en un punto muy importante que implica la ejecución de métodos efectivos y herramientas que los desarrolladores deben de aplicar, el resultado final es obtener software de calidad.

La calidad de un sistema, aplicación o producto debe de ser tan bueno como los requisitos que se describe en la documentación del desarrollo, un punto muy importante para poder llegar al objetivo es en las pruebas que se realizan para detectar errores. Un buen ingeniero de software utiliza mediciones que evalúan la calidad del análisis y los modelos de diseño, el código fuente y los casos de prueba que se ha creado para aplicar la ingeniería de software.

Según Pressman la calidad del software no es algo en lo que se empieza a pensar una vez que se ha generado el código. El aseguramiento de la calidad es una actividad de protección que se aplica a lo largo de todo el proceso de ingeniería de software y engloba:

- Un enfoque de gestión de la calidad.
- Tecnología de ingeniería del software o del conocimiento efectivo (métodos y herramientas).
- Revisión de técnicas formales que se aplican durante cada paso de una ingeniería del software o del conocimiento. Apoyando en una estrategia de pruebas por niveles.
- Procesos de gestión de la configuración. El control de la documentación del software y de los cambios.
- Un procedimiento que asegure un ajuste a los estándares de desarrollo del software.

- Mecanismos de medición y de información.

Así se debe de garantizar la calidad de cada uno de los subproductos del desarrollo y con ello que los aplicativos cumplan los requisitos (Pressman R. S., 2005).

Existen modelos de calidad de producto, destacando entre ellos la ISO 9126 (ISO, 2001), que especifica diferentes dimensiones de la calidad de producto, la cual recae en el uso de las métricas de software.

Como decía Glass, “no hay que olvidar que las personas son las que hacen el software. Las herramientas ayudan, las técnicas también, los procesos, etcétera, pero sobre todo las personas” (Glass, 2003). McConnell, por su parte conjeturaba que “las personas son las que tienen más potencial para recortar el tiempo de un proyecto, y quienes han trabajado en software han observado las enormes diferencias que hay en los resultados que producen entre desarrolladores medios, mediocres y geniales” (McConnell, 1996), mientras que Boehm después de analizar 69 proyectos comprobó que los mejores equipos de desarrolladores eran hasta 4 veces más productivos que los peores (Boehm B. , 1981).

Conforme a lo mencionado se puede considerar que la gente que participa en un desarrollo de software, no son como obreros en un vínculo de acoplamiento. No son tan fácilmente intercambiables, y el trabajo no es tan repetible como en estos otros trabajos.

Un punto importante que se debe considerar es que la calidad del software no es lo mismo que el testing o las pruebas de funcionalidad, que el software funcione no significa que el producto este bien hecho, si está mal hecho se tiene que analizar la parte del código fuente, el diseño, y validar que este programado de manera correcta, es decir, se tienen que analizar la parte de la calidad del producto.

La importancia del uso de un modelo radica principalmente en el hecho de que es precisamente lo que permite comprender cuales son los elementos específicos de una organización, a la vez ayuda a formular y hablar de que es lo que se debe de mejorar dentro de la misma y como se pueden lograr dichas mejoras, las ventajas son las siguientes:

- Permite que los usuarios puedan enfocarse específicamente en la mejora, ya que ayudan a que no se pierda la idea global.
- Ayudan a mejorar la satisfacción del cliente.
- Permite producir productos y servicios de alta calidad.

El modelo CMMI (Capability Maturity Model) fue inicialmente desarrollado para los procesos relativos al desarrollo e implementación del software por la Carnegie-Mellon University en el año de 1987. El propósito de un modelo CMMI es la administración de riesgos también se considera un modelo de calidad, y que a su vez indica la capacidad que tiene una determinada organización para administrar esos riesgos. Tener un sello de CMMI no siempre asegura un producto de software de calidad. El sello es una evidencia indirecta de calidad, la calidad del producto de software es evidencia directa el modelo CMMI se basa en las siguientes características: (Estada, 2006).

- Las auditorias CMMI no miran la calidad del producto de software, solo se basan si cumplen con buenas prácticas del proceso, si se gestionan requisitos, si se planifican los proyectos.
- Como cliente se contrata al personal certificado en el modelo CMMI no significa que la entrega será en el tiempo indicado.

2.4.1 Antecedentes de la Calidad de Software

Se hablaba de la calidad de software desde el año 1940 como un factor muy importante que ha estado evolucionando constantemente, la calidad se relacionaba con la revisión en los productos con el propósito de detectar los errores, en los años 80´s se toma en cuenta la gestión de la calidad total el cual buscaba garantizar la calidad por medio de la planificación y la creación de modelos de calidad permanente.

Por lo cual, es importante responder a la pregunta de qué es la calidad de software. Es el proceso eficaz de software que se aplica de manera que se crea un producto útil, el cual determina un producto de calidad los cuales cumple con los siguientes requerimientos:

- Nivel de satisfacción: En este punto es importante para los clientes y usuarios que perciben que el producto de software cumple sus necesidades.

- Valor del producto: Son los beneficios que el producto del software tiene como resultado para los usuarios

- Atributos de calidad: En este punto se definen las métricas de calidad que debe tener el software para obtener su buen funcionamiento.

- Defectos: Para este punto se dan a conocer las fallas que presenta el software al utilizarse.

- Calidad del Proceso: En este punto es muy importante tener en cuenta que método de proceso se ocupó para el desarrollo del software.

La definición de la calidad de software según la IEEE, Std. 610-1990 es el grado con el que el sistema, componente o proceso cumple los requerimientos específicos y las necesidades o expectativas del cliente o usuario.

Existen muchas medidas de calidad de software, entre ellas están la corrección, facilidad de mantenimiento, integridad y facilidad de uso de indicadores útiles en cada proyecto, explicadas a continuación:

- Corrección: Para este punto es el grado en el que el software lleva a cabo su objetivo principal. Las medidas a emplear son facilidad de mantenimiento, integridad y facilidad de uso del software (Estada, 2006).

- Facilidad de mantenimiento: Es la facilidad con la que se puede corregir un programa si se encuentra un error, o si el cliente requiere adaptar o cambiar el entorno del software. Una simple métrica orientada al tiempo medio de cambio (TMC) (Estada, 2006).

- El mantenimiento es una de las características más demandadas hoy en día por los clientes de software, que piden que el producto de software que se desarrolle pueda ser de fácil mantenimiento por ellos mismos o incluso por un tercero (Fillottrani, Pablo R., 2007).

- Integridad: En este punto se mide la capacidad de un sistema para resistir ataques contra su seguridad. El ataque se puede realizar en cualquiera de los tres componentes del software: programas, datos y documentos (Fillottrani, Pablo R., 2007).

- Facilidad de uso: Este punto es muy amigable con el usuario y se puede medir en las siguientes características. Habilidad intelectual se requiere para poder aprender el sistema. El tiempo requerido para llegar a ser moderadamente eficiente en el uso del sistema (Fillottrani, Pablo R., 2007).

Se ha identificado un conjunto de propiedades de calidad, que obtienen su valor a partir de métricas de código fuente, y se ha establecido la relación que existe

con las subcaracterísticas antes mencionadas. El objetivo al identificar estas propiedades de calidad y métricas las propiedades son las siguientes:

- Incumplimiento de reglas de programación
- Complejidad ciclomática
- Estructura de paquetes y clases
- Tamaño de unidades
- Código duplicado
- Documentación de código
- Acoplamiento y cohesión
- Ciclos de dependencia

2.4.2 Medidas de calidad formales e informales

Durante los estudios formales de ingeniería, existen asignaturas dedicadas a que los alumnos comprendan cual es el concepto de calidad como medirla y asegurarla. En la ingeniería tradicional la calidad se puede medir aplicando una serie de métricas, número de piezas defectuosas, umbrales de estrés o presión de estas.

En el caso del software, también se utilizan variables que permiten medir la calidad del software utilizando:

- Número de líneas de código
- Acoplamiento del diseño
- Complejidad ciclomática

- Algoritmos
- Impacto de memoria bajo determinadas circunstancias
- Tiempos de respuesta

Los puntos mencionados se tratan de las medidas de calidad formales que permiten medir y comprobar el resultado de un proceso del desarrollo de software.

Existe otro tipo de calidad: la cual se percibe cuando se intenta leer y seguir el código que forma parte de la aplicación. Las medidas informales que son medidas de la calidad no menos importante que todas aquellas medidas formales.

Otro punto frecuente que reduce la calidad de los desarrolladores es el código innecesario, redundante, duplicado o código muerto (es el código que nunca llega a ejecutarse). En algunos casos para poder llegar al objetivo deben ser evitados los siguientes puntos para no contar con código innecesario:

- Tratamiento temprano de excepciones: cuando las excepciones se capturan en puntos en los que no pueden ser tratadas con éxito, acaban generando código que simplemente relanza la excepción.

- El recubrimiento directo de APIs: si ya hay un API publica, o una librería común que ofrece una funcionabilidad, puede utilizarse directamente.

- El intento de excesiva optimización del código: Dependiendo del entorno de ejecución, este intento de optimización de código puede ser contraproducente.

- La refactorización y simplificación progresiva del código, que puede provocar que código auxiliar inicial, haya dejado de ser necesario.

A continuación, se mencionarán algunos ejemplos de normas y modelos los cuales tienen como objetivo medir la calidad en el software.

ISO/IEC 25000 conocida como SQUARE (Software Product Quality Requirements and Evaluation), que tiene como objetivo la creación de un marco de trabajo para evaluar la calidad del producto de software, el modelo definido para para el mantenimiento de software es el siguiente:

- Análisis: Se define como la facilidad para identificar las partes de un sistema que se deben de modificar debido a deficiencias o fallos, o la capacidad de evaluar el impacto que puede provocar un cambio en el sistema.

- Modularidad: Se define como el grado, en el que un sistema se encuentra dividido en módulos de forma que el impacto que causa una modificación en un módulo sea mínimo para el resto.

- Capacidad de ser modificado: se define como el grado en el que se pueden realizar cambios en un producto de software de forma efectiva y eficiente, sin introducir defectos ni degradar su rendimiento.

- Capacidad de ser reutilizado: Se define como el grado en que un activo (modulo, paquete, clase) puede ser usado en más de un sistema o en la construcción de otros activos.

- Capacidad de ser aprobado: Se define como la facilidad para establecer criterios de prueba para un sistema y realizar las pruebas que permitan comprobar que se cumplen los criterios.

Quality Assurance (QA): el objetivo principal del grupo de Quality Assurance, es garantizar la calidad del producto del software, así como los subproductos necesarios para su evolución y su ciclo de vida. Sus criterios son los siguientes:

- Certificar la calidad de los productos aplicando un plan de aseguramiento de la calidad (SQAP).
- Entregar productos estables y viables.
- Controlar la calidad de los entregables de los equipos de desarrollo.
- Obtener informes y estadísticas de la calidad para monitorizar el proceso.
- Asegurar:
    - Que el producto de software cumple con los requisitos especificados por el usuario.
    - Que el producto funcionará de acuerdo con lo esperado de manera correcta y eficiente.
    - Que la operación del producto no afectará al resto de los sistemas existentes.
    - Que la evaluación del producto a lo largo de su vida será viable y sencilla garantizando la calidad de toda la información técnica para su evolución.

2.4.3 El costo de la calidad

Por desgracia estimar los costos de un proyecto supone intentar anticiparse a los acontecimientos y predecir, normalmente con pocos elementos de juicio, el comportamiento del proyecto a lo largo del tiempo. Como se puede suponer, no se trata de una tarea fácil, pero si una labor totalmente imprescindible para la buena marcha del negocio. Para este punto es muy importante saber el tener en cuenta los siguientes puntos para definir el costo de la calidad de software, en lo que respecta a:

Costos de Prevención: Se refiere de todos aquellos costos que implican esfuerzo de la calidad de software, el objetivo principal es prevenir los defectos en todas las fases del desarrollo del software.

Costos de la evaluación: para este punto es muy importante calificar el esfuerzo aplicado en el software, esto se refiere a la revisión específica de los requerimientos, diseño y componentes del software.

2.5 Modelos de Calidad

Analizando la información que anteriormente se describe se puede determinar que el factor más importante para un cliente es la calidad del producto y saber que espera del resultado del software. Por lo cual, es importante mencionar algunos modelos y métricas de calidad y los puntos que evalúan.

Modelo McCall

Este modelo fue propuesto por Jim McCall en 1977 fue destinado a ser utilizado durante el proceso de la calidad de sistemas, los factores desarrollados según el modelo de McCall se centran en tres, aspectos importantes para el análisis de la calidad de software, define 11 factores y 23 criterios. McCall propone algunos de los factores de calidad que métricas comúnmente utilizadas para evaluar la calidad del software.

En la Tabla 1., se muestra el análisis que se hizo del modelo de calidad de McCall con relación a un objetivo de aprendizaje. Las métricas propuestas por McCall para indicar el grado que cada sistema posee una determinada característica que impacta la calidad con su respectiva puntuación.

Tabla 1. Métricas del Modelo McCall

Perspectivas	Factores	Criterios
Operatividad del producto: Factores de calidad que influyen en el grado en que el software cumple con su especificación.	**Usabilidad**: la factibilidad de uso del software	Operatividad Entretenimiento Comunicación
	Integridad: la producción del programa del acceso no autorizado	Control de acceso Auditoria de acceso
	Corrección: el grado en que una funcionalidad coincide con su especificación	Rastreabilidad Completitud Consistencia
	Fiabilidad-confiabilidad: la capacidad de los sistemas de no fallar / la medida en que falla del sistema.	Consistencia Exactitud Tolerancia a fallos
	Eficiencia: además clasificado en eficiencia de la ejecución y la eficiencia de almacenamiento por lo general significa que el uso de los recursos del sistema, como por ejemplo el tiempo de procesador o el tiempo en memoria.	Eficiencia en ejecución Eficiencia en almacenamiento.
Revisión del producto: factores de calidad que influyen en la capacidad de	**Mantenibilidad**: esfuerzo requerido para localizar y arreglar un fallo en el programa dentro de su entorno operativo	Simplicidad Concreción
	Facilidad de prueba: la facilidad del programa de realizar pruebas para asegurarse de que esté libre	Simplicidad Instrumentación Autodescripción Modularidad

cambiar el productor de software.	de errores y cumpla con su especificación	
	Flexibilidad: la facilidad de hacer los cambios necesarios según lo solicitado en el entorno operativo.	Autodescripción Capacidad de expansión Generalidad Modularidad
	Reusabilidad: la facilidad de reutilización de software en un contexto diferente	Autodescripción Generalidad Modularidad
	Interoperatividad: el esfuerzo requerido para acoplar el sistema a otro sistema	Modularidad Similitud de datos Independencia del sistema Independencia de la máquina.
	Portabilidad: el esfuerzo requerido para transferir un programa desde un programa desde un entorno a otro.	Autodescripción Independencia del sistema Independencia de la máquina.

El modelo de McCall es uno de los primeros desarrollados, la mayoría de los factores definidos conservan su vigencia en la actualidad, y muchos otros modelos de calidad desarrollados y adaptados posteriormente se basen en él, incluso la norma ISO 9126 es una estandarización de este modelo. Entre las ventajas y desventajas del Modelo McCall son las siguientes:

- Se enfoca en el modelo final identificando los atributos claves desde el punto de vista del usuario.

- Identifica una serie de criterios, tales como la simplicidad, capacidad de expansión.

- No siempre existe una relación perfectamente lineal entre los valores métricos y las características que se deben de estimar.

- La idea del modelo es la descomposición del concepto genérico de la calidad en tres capacidades importantes para un producto de software y a su vez cada capacidad se descompone en un conjunto de factores.

Modelo de Boehm

Es un modelo de calidad fijo el cual fue propuesto por Barry W. Boehm en el año de 1978 y define la calidad del software en términos de atributos cualitativos y los mide usando métricas. El modelo no es muy distinto al de McCall, porque muchos de sus factores de calidad son los mismos. Este modelo también presenta sus factores de calidad estructurados jerárquicamente de alto a bajo nivel, aunque este modelo se basa en que el software debe de hacer lo que el usuario quiere que haga, por lo tanto, en su estructura presenta características de alto nivel. La Figura 8 muestra con detalle las características del modelo Boehm, el cual se centra en:

- Sus características operativas.

- Su capacidad para soportar los cambios.

- Su adaptabilidad a nuevos entornos.

- La evaluación del desempeño del hardware.

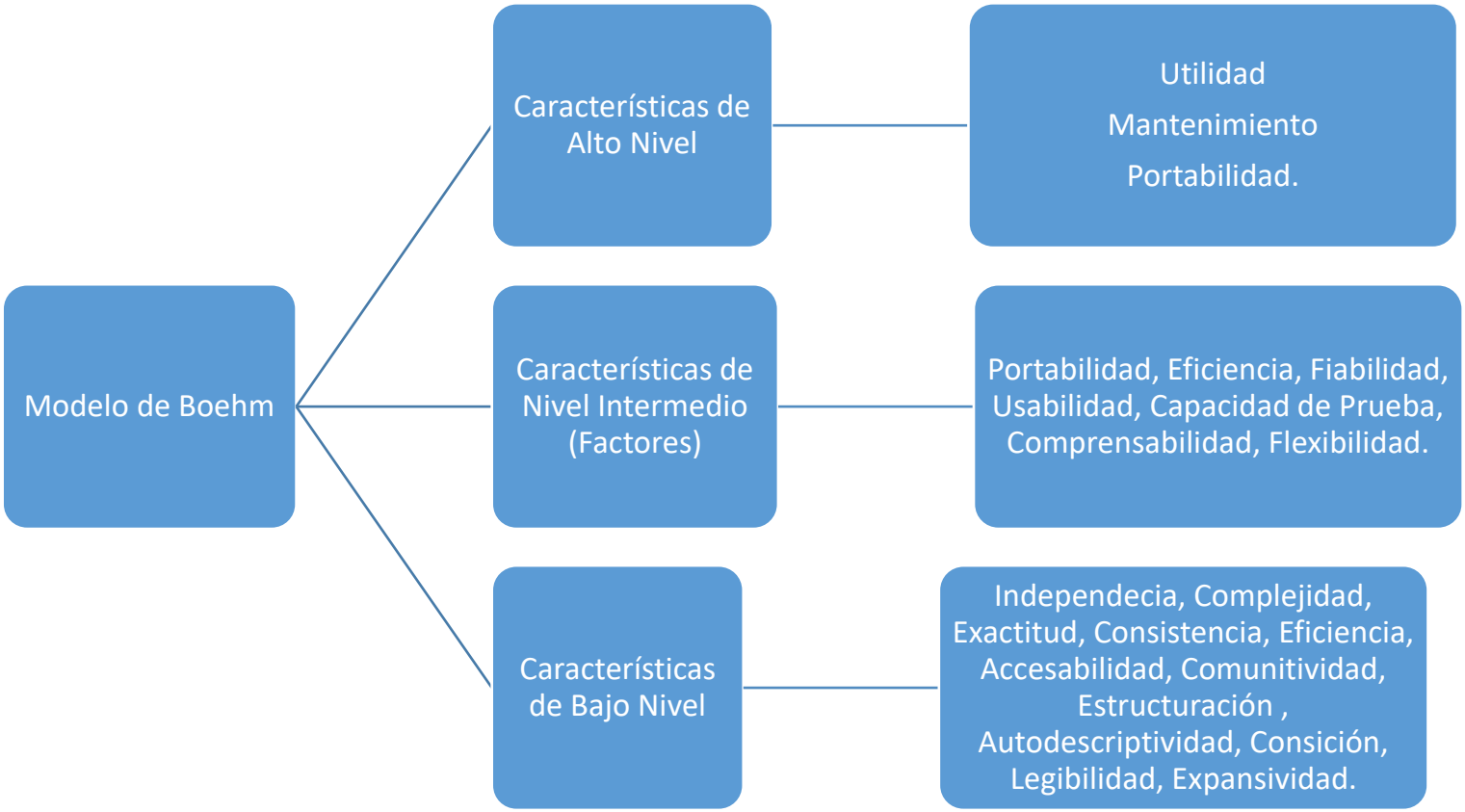

Figura 9. Modelo de Boehm (Elaboración Propia)

Características de alto nivel:

- Utilidad (usable, confiable, eficiente) es el producto en sí mismo.
- Mantenimiento es la manera de que tan fácil es modificarlo, entenderlo y reiniciarlo.
- Portabilidad si se puede seguir usándose si se cambia en el ambiente.

Características de nivel medio:

- Portabilidad (Utilidad general)
- Fiabilidad
- Eficiencia
- Usabilidad
- Capacidad de prueba
- Flexibilidad

Características de bajo nivel

- El nivel más bajo corresponde a las características asociadas a uno o dos criterios de calidad.

Aunque parezcan similares, la diferencia está en que McCall se enfoca en medidas precisas de alto nivel, mientras que Boehm presenta un rango más amplio de características primitivas. La mantenibilidad está más desarrollada en Boehm. La Tabla 2 muestra las diferencias.

Tabla 2. Diferencias entre los Modelos MacCall y Boehm

Criterio	**McCall**	**Boehm**	**Criterio**	**McCall**	**Boehm**
Correctitud	X	X	Confiabilidad	X	X
Integridad	X	X	Usabilidad	X	X
Eficiencia	X	X	Mantenibilidad	X	X
Testeabilidad	X		Interoperabilidad	X	
Flexibilidad	X	X	Reusabilidad	X	X
Portabilidad	X	X	Claridad		X
Modificabilidad		X	Documentación		X
Entendibilidad		X	Validez		X

Ventajas del modelo:

- Presenta un rango alto de características primitivas.
- Une los mejores elementos de otros modelos.
- Integra el desarrollo del software con el mantenimiento.
- Es el segundo modelo de calidad más conocido.

Desventajas del modelo:

- Genera mucho tiempo el análisis.

- Es un modelo costoso.
- Funciona mejor en grandes proyectos.
- Se trabaja siguiendo un protocolo y debe ser seguido estrictamente para un buen funcionamiento.

El modelo de Boehm tiene como principal objetivo la calidad del software, y que el software realice lo que desea el usuario, utilizando los recursos informáticos de manera correcta y eficiente, sea fácil de utilizar y aprender, y sea bien diseñado, codificado probado y mantenido. Este modelo es similar al de McCall ya que presenta una jerarquía de características como se muestra en el cuadro comparativo antes mencionado.

Modelo de Furps

Este modelo fue propuesto por Hewleet Packard y Robert Grady en el año de 1987. En el que se desarrollan un conjunto de factores de calidad de software, bajo el acrónimo de FUPRS funcionalidad, usabilidad, confiablidad, desempeño y capacidad de soporte. Define la calidad en términos de requerimientos funcionales y no funcionales, estos requerimientos propuestos por el modelo los cuales se dividen en caracteristicas y cada categoria tiene sus propias caracteristicas asociadas.

Los requerimientos funcionales (F) especifican funciones que el sistema debe de ser capaz de realizar, sin necesidad de tomar restricciones fisicas a consideracion y se definen a través de las entradas y salidas esperadas.

Los requerimientos no funcionales (URPS), usabilidad, confiabilidad, desempeño y capacidad de soporte, describen atributos del sistema o atributos del ambiente de sistema. En la tabla 3 se muestra la sigla del modelo de Furps y describe cada unos de los puntos y caracteristicas relacionados para obtener calidad en el producto.

Tabla 3. Atributos del Modelo FURPS

Sigla	Tipo de Requerimiento	Descripción
F	Funcionalidad	Características de Sistemas
		Capacidades
		Seguridad
U	Usabilidad	Factores Humanos
		Documentación
R	Fiabilidad (Realiability)	Recuperabilidad
		Precisión
		Predicción
P	Desempeño (Performance)	Velocidad
		Eficiencia
		Consumo
		Productividad
S	Soporte (Supportability)	Adaptabilidad
		Extensibilidad
		Mantenibilidad
		Compatibilidad
		Configurabilidad

Ventajas del Modelo de Furps

- Se pueden reducir los riesgos de no considerar alguna de las facetas del desarrollo de un sistema.
- Es posible de estandarizar algunos criterios para poder obtener los requerimientos.
- Los criterios son de fácil compresión lo que facilita su implementación.

Desventajas del modelo de Furps

- El modelo tiene bastante métricas lo que puede generar mayor demanda en tiempo y costos.

Modelo de Gilb

Este modelo presenta como objetivo principal la definición de los atributos de calidad que realmente le interesan al usuario y el nivel de calidad que debe tener cada uno de ellos para satisfacerlo ya que no tienen sentido exigir calidad en un producto, si no se cuenta con una base. Su pertenece a un tipo fijo el cual se compone de 4 dimensiones de calidad:

1. Calidad de trabajo que evalúa la capacidad natural del sistema para realizar su trabajo, con los siguientes atributos:
    a. Capacidad de proceso
    b. Capacidad de repuesta
    c. Capacidad de almacenamiento.
2. Disponibilidad que es la medida del sistema para realizar de forma útil el trabajo para el que fue diseñado, sus atributos son:
    a. Fiabilidad
    b. Mantenibilidad
    c. Integridad.
3. Adaptabilidad es la medida donde se mide la capacidad de un sistema para ser modificado de manera adecuada, sus atributos son:
    a. Improbabilidad

b. Extensibilidad

c. Transportabilidad.

4. Usabilidad/Utilidad es la medida de la facilidad con que la gente es capaz de utilizar el sistema, con los siguientes atributos:

    a. Requerimientos de entrada

    b. Requisitos de aprendizaje

    c. Habilidad de manejo.

Los elementos de calidad del modelo de Gilb se muestran en la Tabla 4.

Tabla 4. Característica y atributos del Modelo Gilb

Propiedades	Criterio de Calidad
Características	Corrección
	Facilidad de mantenimiento
	Integridad
	Facilidad de uso
Atributos	Capacidad de trabajo
	Disponibilidad
	Adaptabilidad
	Usabilidad

Como se puede observar, el modelo Gilb propone características como la corrección, la integridad la facilidad de mantenimiento y la facilidad de uso, como base para proporcionar indicadores útiles para los equipos de trabajo y obtener la calidad de software. En este punto es importante destacar que esta métrica

considera la interacción del usuario con el software, es decir, toma importancia a la interfaz del usuario.

Modelo de calidad de Dromey

Este modelo de calidad fue propuesto por Robert Geo Dromey en el año de 1995. Su principal objetivo es trabajar con una estructura que permite construir y utilizar un modelo de calidad el cual sea funcional para poder evaluar las etapas de requerimientos, diseño e implementación en la calidad del software. Este modelo plantea la calidad del producto por medio de las subcaracterísticas que pueden ser medidas y evaluadas como características.

El modelo de Dromey se conforma por un esquema de 6 relaciones binarias entre 3 entidades definidas (conjunto de componentes, propiedades que acarrean calidad de los componentes, los cuales se convierte en atributos de alto nivel).

Las características de calidad planteadas por Dromey son: eficiencia, confiabilidad, facilidad de mantenimiento, portabilidad, facilidad de uso y funcionalidad. Las características mencionadas son agrupadas de la siguiente manera en cuatro categorías las cuales implican las propiedades del producto correctitud, internas, contextuales, descriptivas como se muestra en la Tabla 5.

Tabla 5. Característica y atributos del Modelo Dromey

Propiedades de Producto	**Características de Calidad**
Correctitud	Funcionalidad
	Confiabilidad
Internas	Mantenibilidad
	Eficiencia
	Confiabilidad
Contextuales	Mantenibilidad
	Reusabilidad
	Portabilidad
	Confiabilidad
Descriptivas	Mantenibilidad
	Reusabilidad
	Portabilidad
	Usabilidad

Los pasos para la aplicación del modelo de Dromey:

1. Seleccionar el conjunto de atributos que se necesitan evaluar.
2. Realizar una lista de todos los componentes o módulos del sistema.
3. Identificar las propiedades de calidad de cada componente.
4. Determinar cómo afecta cada propiedad en los atributos de calidad.
5. Evaluar el modelo de calidad.

Las propiedades de este modelo se pueden utilizar en diferentes contextos, asociadas a la identificación de propiedades de calidad, la implementación del

modelo de Dromey se refleja como una definición de métricas de calidad y estadísticas asociadas al desarrollo del software.

2.6 Comparación de modelos de calidad

En la Tabla 6., se muestra una selección de los atributos con mayor frecuencia en los diferentes modelos antes mencionados, lo cual expresa la posibilidad de que exista consistencia en la forma de medir la calidad en el software además se puede hacer un comparativo entre los modelos los cuales tienen en común un nivel bajo y un nivel alto.

Un aspecto para considerar en los modelos es que no siempre un atributo posee el mismo nombre en cada uno de los modelos.

Tabla 6. Comparación de los modelos de calidad mencionados

Modelo/Atributos	**McCall**	**Boehm**	**Furps**	**Gilb**	**Dromey**
Reusabilidad	x	--	--	--	X
Exactitud	x	x	x	x	--
Confiabilidad	x	x	x	x	X
Eficiencia	x	x	x	--	X
Usabilidad	x	--	x	--	X
Comprensibilidad	--	x	--	--	--
Consistencia	x	x	x	--	X
Auto descripción	x	x	--	--	X

Funcionalidad	--	--	x	x	X

Analizando los modelos se permite identificar los factores de calidad, atributos y variables de medición, en los cuales algunos puntos son muy semejantes. Esto nos permite identificar los atributos de manera específica como lo es el de confiabilidad que es el atributo que en el que coinciden todos los modelos. De igual forma, puede observarse que el Modelo Gilb es el que menos atributos considera para medir la calidad, seguido del Modelo Furps y Boehm, mientras que los Modelos McCall y Dromey son los que más atributos miden para garantizar la calidad del software.

3.0 Conclusión

Hoy en día, el desarrollo de software general o específico ha tenido una gran demanda debido al incremento del uso en la tecnología como son los equipos de cómputo de escritorio o móviles (pc, celulares, tabletas, incluso aparatos domésticos), la automatización de procesos en las empresas o instituciones académicas, el uso de las redes sociales y las telecomunicaciones.

En la ingeniería de software se aplican métricas que permiten como su nombre lo indica “medir” el producto de diferentes formas, como se mencionó a lo largo de este documento se toman en cuenta métricas de longitud o tamaño, de función, complejidad, productividad, desempeño y calidad. Todas las métricas tienen una razón y se aplican según lo establecido en la planificación y desarrollo del proyecto, así como en la instalación del producto final, el mantenimiento y sobre todo en la satisfacción del cliente.

Por lo anterior, el punto central de esta investigación son las métricas de calidad. Como es de esperarse, exigir un producto final con calidad es una condición para la mayoría de los usuarios, por lo cual, se hace prioridad medir la calidad del producto para clasificarlo como “bueno” o “malo”. En la elaboración de este trabajo el principal objetivo es tener en cuenta los modelos más comunes y sus atributos, los cuales, permiten medir la calidad de software y la complejidad de los modelos establecidos.

Las métricas de calidad consideran estándares y modelos. Los primeros surgieron tomando en cuenta los atributos de los modelos de calidad, que como es bien sabido, el principal objetivo es incrementar la satisfacción del cliente mediante procesos de mejoras continuas. Una de las normas más conocidas en la parte de la calidad es ISO 9001. Con respecto a los segundos, se tomaron en cuenta cinco modelos de calidad, los cuales, son los más conocidos desde que se tiene conocimiento de la calidad de software. La meta fue identificar sus

características y sus atributos, así como, la semejanza que existe entre ellos. De tal forma que se enriquezca el conocimiento en el área de la ingeniería de software y de las métricas en él.

En la comparación de los modelos (Tabla 6), se puede apreciar que los modelos comparten la mayoría de los atributos, pero la forma de medirlos considera diferentes aspectos, por ejemplo, en el Modelo McCall fue el primer modelo de calidad el cual se enfoca en los atributos de desde la perspectiva del usuario una de las ventajas es que existe una relación directa entre los desarrolladores y el usuario, evalúa el producto a nivel bajo.

Mientras que el modelo Boehm pone mayor atención a que el software debe de hacer lo que el usuario quiere que haga, por lo tanto, se espera que se utilice los recursos del computador correcta y eficientemente, sea fácil de usar y de aprender para los usuarios este modelo es similar al de McCall.

Por otro lado, el modelo FURPS tiene como objetivo realizar la evaluación de la calidad de un producto, como primer punto se asigna prioridades y después se definen los atributos de calidad que pueden ser medidos.

En lo que respecta a Gilb se hace un estimado de cual es el nivel de calidad, y la capacidad del software para realizar el trabajo para el cual fue creado teniendo en cuenta el proceso, la respuesta y el almacenamiento.

Por último, con el modelo Dromey tiene el propósito de trabajar con una estructura que permite construir y utilizar un modelo de calidad enfocado en la facilidad evaluando etapas como son el diseño y la implementación.

De lo anterior se puede destacar que los modelos de calidad para el software en su mayoría se han inspirado en el modelo de Dromey lo cual hace que tengan una gran posibilidad de encontrar similitudes entre sus atributos de

calidad, además de proponer nuevos cambios que los modelos restantes no tomaron en cuenta.

Como sabemos hoy en día para las empresas desarrolladoras de software la satisfacción del cliente es lo más importante por lo cual al momento de hablar de calidad dentro de software implica calificarlo respecto al cumplimiento de las especificaciones iniciales y a la usabilidad, por lo cual, las métricas de calidad se convierten en una disciplina más dentro de la ingeniería de software y debe considerarse a lo largo de todo el ciclo de vida, el cual se ejecuta en paralelo desde la planificación del producto hasta la fase de producción.

Un ejemplo de ello se mostró en la sección 2.6 que lista varios proyectos que consideraron medir el software en diferentes etapas y que claramente ha beneficiado en gran porcentaje la reducción de errores y ha aumentado la satisfacción del cliente.

Referencias

(s.f.).

12207:2017, I. (2017). Software and systems engineering. Estados Unidos: Published.

Álvarez, N. (2019). Aprender Photoshop. Madrid: Prime.

Boehm, B. (1978). Characteristics of Software Quality. North-Holland Publishing, 130.

Boehm, B. (1981). Software engineering economics Prentice. Hall PTR., 60.

Braude, E. J. (2003). Ingenieria de Software. Mexíco: Alfaomega.

Ceballos Sierra, F. J. (1997). Cobol Estructurado. España: Macrobit.

Cuervo, M. C. (2017). Modelos de calidad del software, un estado del arte. colombia: Unilibre Cali.

DÉLÉCHAMP, F. (2018). Java y Eclipse. España: ENI.

Delgado, A. T. (2018). Metricas de Software. Attribute, Software Quality, Metrics, Software Metrics, 60.

Diaz, E. (2013, enero 10). Las Metricas de Halstead. Retrieved from http://www.programando.org/blog/2013/01/desafio-enero-las-metricas-de-halstead/

Domínguez, F. A. (2012). Herramienta de Evaluación de la Calidad de Objetos de Aprendizaje (herramienta COdA). Madrid: De Armas, I.

Dromey, G. (1994). A Model for Software Product Quality. Software Quality Model, 146-162.

Estada, A. F. (2006). Calidad de Software. España: Cienfuegos.

Fenton N. E., P. S. (1997). Software Metrics. A rigorous. and Practical Approach. 2nd Edition, 56.

Fillottrani, Pablo R. (2007, enero 17). Calidad en el Desarrollo de Sotfware. Retrieved from Metricas del Software: http://200.49.226.11/~prf/teaching/SQ07/clase3.pdf

Flores, C. S. (2010). Visual Studio 2010. peru: Macro.

Gedda, R. (2004). Linux breaks desktop barrier . Estados Unidos .

Gilb, T. (1987). Principles od Software. USA: Addison Wesley.

Glass, R. L. (2003). Facts and fallacies of software engineering. Addison Wesley.

Informe Final - SQO-OSS. (01 de octubre de 2010). Obtenido de observatorio de calidad de software para sotfware de codigo abierto: https://translate.google.com/translate?hl=es-419&sl=en&u=https://cordis.europa.eu/publication/rcn/10217_en.html&prev=search

ISO/IEC/IEEE. (2017). norma ISO 12207-1. Estados Unidos: https://www.iso.org/standard/63712.html.

Jimenez, D. S. (2005). Calidad de Software en el uso de Metodologias Agiles parae el Desarrollo de Software. Centro de Investigacion en Computacion , 90.

Lovelle, J. M. (1999). Calidad del Software. España: Universidad de Oviedo.

M, I., & Flynn, M. A. (2011). Sistemas operativos. España: Cengage Learning.

McCabe, J., Richards, P., & Walters, G. (1970). “Factors in Software Quality. Vols I,II, III.

McConnell, S. (1996). Rapid development Microsoft Press. Rapid development Microsoft Press, 40.

McCraken, D. (1997). Programming Language "Fortran" Extended. ANSI X3.

Moreno, J. L. (2007). Exploración de Modelos y Estándares de calidad para el producto software. Enlace Informatico, 99-111.

NORO, B. (2019). Paqueteria de Office. Madrid: Eni.

Pressman, R. (1993). Ingenieria de Software. Madrid: McGgraw Hill.

Pressman, R. S. (2005). Ingeniería de Software. Un enfoque práctico. Editorial MCGRAW-HILL, 2005.

Pressman, R. S. (2010). Ingeniería del Software. New York: McGrawHill.

Radatz Chairperson, J. (1993). Glossary of Software Engineering Terminology. IEEE Software Engineering Standard, 7.

Radatz Chairperson, J. (1993). Glossary of Software Engineering Terminology. IEEE Software Engineering Standard, 7.

Rey, A. (2015). Evaluación de la Calidad de la Tecnología Educativa. CVUDES.

Saiz, D. Z. (1998). Buscadores de Internet . Madrid: S.A. Ediciones.

Scalone, F. (2006). “Estudio comparativo de los modelos y Estándares de calidad del software. Buenos Aires.

Sommerville, I. (2005). Ingenieria del Software. Retrieved from Ingenieria del Sotfware Septima edicion: https://ulagos.files.wordpress.com/2010/07/ian-sommerville-ingenieria-de-software-7-ed.pdf

Sommerville, I. (2005). Ingenieria del Software. Madrid: Pearson Educacion.

SOTO PEÑA, J. R. (2015). Cuadro Comparativo de Modelos para evaluar la calidad del Software. madrid.

Soto, M. G. (2017). Análisis de Malware para Sistemas Windows. Madrid: RA-MA Editorial.

Tukey, J. W. (1957). En las ciencias de la computación y la ingeniería de software. mexico: maccgrill.