COMPARACION DE MODELOS DE CALIDAD, FACTORES Y METRICAS EN EL AMBITO DE LA INGENERIA DE SOFTWARE

Lic. Marcela Alejandra CONSTANZO
mconst1978@hotmail.com
GISP- Instituto de Tecnología Aplicada

Universidad Nacional de la Patagonia Austral - Unidad Académica Río Gallegos
Dpto. Ciencias Exactas y Naturales

Río Gallegos, Abril de 2.014

Resumen. Existen diferentes enfoques de desarrollo de software, en su mayoría priorizan la calidad en el proceso y el producto obtenido. Para poder lograr esto es importante el uso de modelos de calidad apropiados para cada metodología. Estos modelos de calidad presentan factores e indicadores que describen las características del software y sus relaciones y pueden ser adaptaciones de otros o creados tomando como base los estándares existentes.
Este trabajo describe y analiza algunos modelos de calidad desarrollados para las metodologías de desarrollo de software orientada a objetos, a componentes, a aspectos y los métodos ágiles, realiza una comparación según los criterios, factores y características que lo componen, niveles de abstracción y métricas para llevar a cabo la medición.

Palabras Claves*: Modelos de Calidad, Factores de Calidad, Enfoques de Desarrollo de Software, Métricas, Metodologías Agiles*

INTRODUCCIÓN

Es importante resaltar en la calidad en el Desarrollo de Software el uso de Modelos de Calidad y Metodología adecuada que permitan controlar todo el proceso.
La Ingeniería de Software provee diferentes metodologías para el desarrollo de software. La necesidad de desarrollo rápido de aplicaciones de alta calidad ha llevado a darle gran importancia al concepto de calidad en todas las etapas.
Un modelo proporciona un marco y un lenguaje para comunicarse, también proporciona un estándar y la experiencia necesaria en el tema a abordar. Un modelo de Calidad describe las características que componen la calidad del software y sus relaciones. Actualmente existen varios modelos que han ganado popularidad pero no tienen sustento científico, estos modelos presentan factores en común que permiten realizar las mediciones de acuerdo al interés que se presente. Han sido creados y adaptados para el Desarrollo Orientado a Objetos y a Componentes y en menor medida al Desarrollo de Software Orientado a Aspectos (DSOA) y las Metodologías Ágiles.
El presente trabajo describe y analiza los modelos de calidad y sus adaptaciones para los diferentes paradigmas de desarrollo existentes. Realiza una comparación según los criterios, factores y características que lo componen, niveles de abstracción y métricas para llevar a cabo la medición y un análisis de la información obtenidas y presenta un análisis sobre el pensamiento actual de las métricas para el desarrollo ágil.
En la Sección 1 se describe la revisión sistemática y metodología de búsqueda y recolección de la información, en la Sección 2 del documento se detallan los conceptos de calidad de software, los modelos de calidad y un breve resumen de que son las métricas y su clasificación. En la Sección 3 se describe una breve reseña sobre algunos enfoques de desarrollo y las metodologías ágiles, posteriormente en la Sección 4 se analizan los modelos de calidad para cada uno de los paradigmas de programación. En la Sección 5 se presenta una comparación de las métricas para los diferentes enfoques, en la Sección 6 se describen los lineamientos para medir la calidad en las metodologías ágiles y posteriormente un análisis gráfico de los resultados y la conclusión y recomendaciones basadas en la investigación.

1. REVISION SISTEMATICA DE LA INFORMACION

La necesidad de una revisión sistemática se deriva de la exigencia de los investigadores para resumir toda la información existente acerca de un fenómeno de manera exhaustiva e imparcial [38]

Las etapas contempladas para llevar a cabo el desarrollo de la revisión son las siguientes [39]:

1. Búsqueda de la información mediante los criterios de definidos para tal caso.

2. Selección de la documentación de valor para el desarrollo del trabajo teniendo en cuenta que respondan a las preguntas que definen el tema de investigación.

3. Análisis uno a uno la documentación seleccionada mediante los criterios de inclusión y exclusión definidos.

4. Extracción de los datos de importancia de cada documento para posteriormente generar las fichas correspondientes agrupando los datos según los siguientes temas:
    - Calidad de software
    - Modelos de calidad de software
    - Métricas
    - Enfoques de desarrollo de software (Objetos, Componentes y Aspectos)
    - Metodologías Agiles
    - Calidad aplicada a los diferentes enfoques

    La síntesis de los datos se representará en un cuadro con la siguiente estructura:
    - Enfoque
    - Factores de calidad
    - Atributos internos
    - Variables de medición
    - Métricas
    -

5. Análisis y presentación de los resultados obtenidos de la aplicación de la revisión sistemática de la información.

El objetivo de la revisión es organizar y documentar toda la información seleccionada sobre los modelos de calidad existentes para los diferentes enfoques de desarrollo y las metodologías ágiles para posteriormente confeccionar un cuadro comparativo de las métricas analizadas.

1.1 Búsqueda de la información

La búsqueda de la información está basada sobre el material aportado por investigadores reconocidos académicamente en el ámbito de la computación, tesis presentadas para títulos de grado y postgrado, como así también artículos expuestos en workshop, conferencias y encuentros internacionales, publicaciones en revistas, bibliografía disponible en la universidad, en bibliotecas virtuales y principalmente las que surjan de las búsquedas de palabras claves y combinación de ellas en los diferentes sitios web[1], tanto en español como en inglés.

[1] *Sitios Web utilizados para búsquedas primarias*

1.2 Selección de la documentación

Una vez realizadas las búsquedas primarias de la información se observa que los resultados arrojados para las palabras claves son cifras realmente considerables, y en las búsquedas que involucran combinación de palabras o frases los valores se redujeron radicalmente, es por eso que como primer paso, previo a la selección, se utilizaron frases con combinaciones de palabras como base de la búsqueda y se descartaron aquellos documentos que no cuentan con un título significativo para la investigación, al grupo de información resultante se le aplicó el proceso de selección según lo planificado.
Para la búsqueda secundaria se tuvo en cuenta el año de la publicación del documento, el reconocimiento académico del autor, evento en que fue presentado y el idioma de confección del mismo. Esta selección arrojó como resultado un total de *40* artículos que se *incluirán* para el análisis y extracción de los datos. Cabe mencionar que el rango en los años de publicación para los documentos que contienen información referente a la calidad de software se amplió, ya que no se pueden descartar aquellos que fueran anteriores al año 2005 por ser un tema que tiene mucho tiempo de investigación y desarrollo.

1.3 Análisis de la información seleccionada

El análisis de la información se realiza tratando de responder los interrogantes que se detallan a continuación, en base a la información que se obtiene de la lectura de la *introducción*, que es donde se describe el tema que se presentará en la publicación, y *conclusión*, donde se presentan las contribuciones del trabajo y posibles nuevas líneas de investigación.
Para poder determinar si el artículo seleccionado se ajusta al objetivo de la investigación debe dar respuesta a alguno de los cuestionamientos que se detallan a continuación:

- Presenta definiciones concretas sobre los temas abordados?
- Describe los enfoques de desarrollo?
- Presenta algún modelo de calidad para algún enfoque?
- Describe métricas para algún factor de calidad o variable de medición?
- Analiza la calidad en el desarrollo de software dentro de algún enfoque?
- Analiza la calidad para las metodologías ágiles?

Si se puede responder a estos interrogantes con la información obtenida, el artículo es considerado y se procede a agrupar los de acuerdo a la clasificación prevista según tipo de publicación. Para el fin de esta investigación, del total de información que resultó de la etapa de selección se procede a la lectura del material para extraer los datos que permitan la generación de las fichas que aportarán al objeto de la investigación.

1.4 Extracción de los datos de valor

Búsqueda Primaria	Fuente (Sitio Web de referencia)
1	http://scholar.google.es/
2	http://www.google.com/
3	http://www.scirus.com/
4	http://www.scielo.org/
5	http://site.ebrary.com/ - Sitio biblioteca virtual
6	http://168.226.35.7/OpenBiblioF/home/index.php - Sitio biblioteca UNPA

Para registrar los datos que surgieron del análisis de la información, que responde a las preguntas identificadas, y ajustándose al protocolo definido se cargaron los datos en fichas automatizadas y definidas.

1.5 Análisis y presentación de los resultados

La comparación entre las métricas analizadas para cada enfoque de desarrollo y las metodologías ágiles, y que fuera desarrollada aplicando el protocolo y revisión propuesta, se presentará en un cuadro organizado por enfoque y autor que posteriormente permitirá realizar análisis sobre el mismo.

2. CALIDAD DE SOFTWARE.

Existen varias definiciones asociadas al concepto de Calidad de Software, Pressman define la calidad de software como "la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente" [1]. Según ISO/IEC 8402 es un "Conjunto de características y relaciones entre ellas que proveen la base para la especificación de los requisitos de calidad y la evaluación de la calidad". Humphrey (1997) la define como "La ausencia de defectos, seguridad, confiabilidad y cumplimiento de las especificaciones", y sostiene que la calidad de software debe ser construida desde el comienzo, no puede ser añadida después [2].
La Ingeniería de Software según [3] se define como: "Establecimiento y uso de principios de ingeniería robustos, orientados a obtener software económico que sea fiable y funcione de manera eficiente sobre máquinas reales". La Calidad es una de las áreas de la ingeniería de software, actualmente existe una fuerte competencia por introducir productos en el mercado, pero no solamente se busca la mayor inserción, sino también la satisfacción del cliente, es por eso que se le está dando mayor importancia a la Calidad en todo el proceso de desarrollo de software.
Calidad en el desarrollo de software es asegurar el mínimo de sorpresas posibles durante todas las etapas del proceso, por eso es recomendable la utilización de Estándares o modelo de calidad. Un producto de alta calidad es uno que conlleva un conjunto de factores de calidad. Estos factores pueden ser descriptos en la especificación de requerimientos; pueden ser culturales, o sea que se espera que normalmente estén asociados con el producto mediante familiaridad de uso; o pueden ser factores de calidad que el desarrollador considere importante aunque no estén en los requerimientos del cliente o en las expectativas de usuarios.ISO 9001.
Es importante hacer mención a algunos términos utilizados en lo que a Calidad se refiere, en este sentido, Gestión de Calidad, es un Conjunto de Actividades para la implementación de políticas de calidad en todo el desarrollo de software. El fin de la gestión es interpretar al cliente y poner en práctica un plan para satisfacer sus expectativas.
Dentro las actividades que comprende la Gestión de Calidad se encuentra el Control de Calidad de Software que consiste en una serie de operaciones en las que se aplican técnicas para verificar la calidad del software y mantener controlado todo el proceso a lo largo del ciclo de vida. Otra actividad importante de la Gestión de Calidad es la Verificación y Validación del Software que consiste en comprobar si el producto obtenido cumple con los requisitos establecidos, es decir si funciona según lo solicitado por el usuario y cumple con sus expectativas.
El Control de Calidad tiene como objetivo la detección de errores en las fases tempranas del desarrollo, para evitar la propagación de los mismos y reducir costos en correcciones.

2.1 Modelos de Calidad del Software

Es importante incluir en la calidad de software la importancia de los requerimientos implícitos y explícitos del producto, que permiten medir la calidad del mismo, y los estándares de calidad y modelos de calidad existentes.
Cada uno de estos modelos de calidad consiste en un conjunto de características y/o factores. Estos factores pueden ser medidos directa o indirectamente, de medición directa como errores y unidades de tiempo e indirectamente como la facilidad de mantenimiento. Las medidas obtenidas deben ser comparadas para obtener una indicación de la realidad. Por ejemplo, mientras más alta es la complejidad, más difícil es conseguir el fácil mantenimiento del producto, es decir que dependiendo del tipo de software y del cliente, distintos factores serán necesarios para distintos atributos de calidad, esto indicará qué modelo de calidad o estándar se debe elegir para realizar el control de la misma.
Los modelos de calidad son aquellos documentos que integran la mayor parte de las mejores prácticas, proponen temas de administración en los que cada organización debe hacer énfasis, integran diferentes prácticas dirigidas a los procesos clave y permiten medir los avances en calidad [4].
Los estándares de calidad son aquellos que permiten definir un conjunto de criterios de desarrollo que guían la forma en que se aplica la ingeniería de software. Los estándares suministran los medios para que todos los procesos se realicen de la misma forma y son una guía para lograr la productividad y la calidad [4]. En el cuadro 1 se describe la estructura de un modelo de calidad.

Cuadro 1 – Estructura de un Modelo de Calidad

Los factores de calidad o atributos externos, son características que componen la calidad, representan la calidad desde el punto de vista del usuario. Los criterios de calidad o atributos internos, son aquellos en los que se descomponen los diferentes factores, representan la calidad desde el punto de vista del producto, son aspectos de calidad asociados a cada factor. Las métricas se definen para cada criterio de calidad, son medidas cuantitativas que indican el grado en el que está presente un atributo en el producto.
Calidad de software implica distinguir entre calidad del producto y calidad del proceso. Cuando se hace referencia a la calidad del producto, lo importante es obtener un software de alta calidad para enfrentar la fuerte competitividad existente actualmente, mientras que la calidad en el proceso de desarrollo permite garantizar productos con calidad aceptable.
Para evaluar la calidad de un producto de software, han surgido distintos modelos, formados por factores y criterios asociados. Al evaluar estos factores de calidad en las diferentes jerarquías, se puede determinar la calidad del producto de software. Entre los modelos más importantes que evalúan la calidad del producto de software se encuentran los siguientes:

2.1.1 Modelo Mc Call

Este modelo fue creado por Jim Mc Call en 1977. Establece 3 perspectivas para el análisis de la calidad de software, define 11 factores y 23 criterios relacionados a estos. Las métricas que propone son preguntas que ponderan numéricamente un determinado atributo del producto de software. Después de obtener los valores para todas las métricas de un criterio específico, el promedio de todas ellas es el valor para ese criterio [5].

En el cuadro 2, se presentan los criterios de calidad asociados a los factores de calidad en el modelo de Mc Call.

Perspectivas	Factores	Criterios
Operatividad del Producto: factores de calidad que influyen en el grado en que el software cumple con su especificación.	Usabilidad: La facilidad de uso del software.	Operatividad Entrenamiento Comunicación
	Integridad: La protección de programa del acceso no autorizado.	Control de Acceso Auditoría 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 el sistema.	Consistencia Exactitud Tolerancia a fallos
	Eficiencia: Además clasificado en la eficiencia de la ejecución y la eficiencia de almacenamiento y por lo general significa que el uso de los recursos del sistema, ejemplo: tiempo de procesador, memoria.	Eficiencia en Ejecución Eficiencia en Almacenamiento
Revisión del Producto: factores de calidad que influyen en la capacidad de cambiar el producto de software.	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 está libre de errores y cumple con su especificación.	Simplicidad Instrumentación Auto-descripción Modularidad
	Flexibilidad: La facilidad de hacer los cambios necesarios según lo solicitado en el entorno operativo	Auto-descripción Capacidad de expansión Generalidad Modularidad
Transición del Producto: factores de calidad que influyen en la capacidad de adaptar el software a los nuevos entornos.	Reusabilidad: La facilidad de reutilización de software en un contexto diferente.	Auto-descripción Generalidad Modularidad
	Interoperabilidad: El esfuerzo requerido para acoplar el sistema a otro sistema.	Modularidad Similitud de comunicación Similitud de datos Independencia del sistema Independencia de la máquina
	Portabilidad: El esfuerzo requerido para transferir un programa desde un entorno a otro.	Auto-descripción Independencia del sistema Independencia de la máquina

Cuadro 2 – Diagrama de Mc Call – Criterios asociados a factores de calidad

Si bien el modelo de Mc Call 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.

2.1.2 Modelo FURPS

Este modelo fue desarrollado por Hewlett-Packard en el año 1987. En el se desarrollan un conjunto de factores de calidad de software, bajo el acrónimo de FURPS: funcionalidad (Functionality), usabilidad (Usability), confiabilidad (Reliability), desempeño (Performance) y capacidad de soporte (Supportability)[6]. En el Cuadro 3 se muestra el diagrama de FURPS y los criterios de calidad y factores asociados.

Factores	Criterios
Funcionalidad	Características y capacidades del programa Generalidad de las funciones Seguridad del Sistema
Usabilidad	Factores Humanos Factores Estéticos Consistencia de la interfaz Documentación
Confiabilidad	Frecuencia y severidad de fallos Exactitud de las salidas Tiempo medio de fallos Capacidad de recuperación ante fallos Capacidad de predicción
Rendimiento	Velocidad de procesamiento Tiempo de respuesta Consumo de recursos Rendimiento efectivo total Eficacia
Capacidad de Soporte	Extensibilidad Adaptabilidad Capacidad de Prueba Capacidad de configuración Compatibilidad Requisitos de instalación

Cuadro 3 – Diagrama de FURPS – Criterios asociados a factores de calidad

2.1.3 Modelo BOHEM

Este modelo propone una jerarquía de niveles, en forma de un árbol con tres ramas principales, que permiten que el software sea de utilidad: Portabilidad, Facilidad de Uso y Facilidad de Mantenimiento. Se estructura en tres niveles: Aplicaciones primarias, Construcciones Intermedias (factores) y Construcciones Primitivas, y finalmente las Métricas que determinan los valores para los criterios (construcciones primitivas) [7]. El diagrama de Bohem se presenta en el Cuadro 4.

Factores	Criterios
Portabilidad	Independencia dispositivos Completitud
Fiabilidad	Completitud Exactitud Consistencia
Eficiencia	Eficiencia dispositivo Accesibilidad

Ingeniería humana	Accesibilidad Comunicatividad Estructuración Auto-descripción
Comprensibilidad	Consistencia Estructuración Auto-descripción Concisión Legibilidad Expansibilidad
Modificabilidad	Estructuración

Cuadro 4 – Diagrama de Bohem – Criterios asociados a factores de calidad

2.1.4 ISO/IEC 9126

El Estándar internacional (ISO), aplicable a todo tipo de software, está basado en un modelo jerárquico con tres niveles: Características, Subcaracterísticas y Métricas. En el primer nivel tiene seis características principales: Funcionalidad, Fiabilidad, Eficiencia, Facilidad de Mantenimiento, Portabilidad y Facilidad de Uso. Estas características (factores) están compuestas a su vez por 27 subcaracterísticas (subfactores) relacionadas con la calidad externa, y 21 subcaracterísticas relacionadas con la calidad interna [8], en el Cuadro 5 se presentan los factores y criterios asociados al modelo.

Factores	Criterios
Funcionalidad	Adaptabilidad Exactitud Interoperabilidad Seguridad
Usabilidad	Comprensibilidad Aprendizaje Operatividad Atractivo
Mantenibilidad	Análisis Cambio Estabilidad Prueba
Fiabilidad	Madurez Tolerancia a fallos Recuperabilidad
Eficiencia	Comportamiento del tiempo Uso de los recursos
Portabilidad	Adaptabilidad Instalación Coexistencia Reemplazo

Cuadro 5 – Diagrama de ISO/IEC 9126 – Criterios asociados a factores de calidad

2.2 Métricas

Para poder medir el grado en que un sistema o proceso posee un atributo dado se definen y utilizan métricas. Se define Métricas de Software como la forma eficaz de proporcionar evidencia empírica que puede mejorar la comprensión de las diferentes dimensiones de la complejidad del software [9].

Estas métricas son aplicadas constantemente durante todo el proceso de desarrollo del software y también en el producto obtenido. Estas mediciones brindan información importante que permite a los desarrolladores mejorar los procesos y productos.
Las Métricas son propias de cada modelo de calidad y se crean para medir los criterios definidos en él. El proceso de recopilación de las métricas de software se realiza mediante la obtención de datos de los procesos de Ingeniería de software y de los proyectos y productos de software. Estas medidas son utilizadas para poder llevar a cabo el cálculo de las métricas, de cuya aplicación y posterior evaluación se plantean los indicadores para las mismas.
Las Métricas pueden ser directas o indirectas, las primeras son aquellas que no necesitan ningún otro atributo o entidad, mientras que las segundas se forman por la combinación de una o más métricas directas [6]. Por ejemplo métricas directas serian LOC: líneas de código fuente escritas o HPD: horas-programador diarias; y métricas indirectas HPT: horas-programador totales (Σ HPD) o LCFH: líneas de código fuente por hora de programador (LOC/HPT).
A continuación se describen una serie de propiedades que se debe cumplir para la validez de las métricas [11].
Para métricas directas:
1. Para que un atributo pueda ser medido, debe permitir que diferentes entidades sean distinguibles una de la otra.
2. Una métrica debe cumplir la condición de representación.
3. Cada unidad que contribuye en una métrica válida debe ser equivalente.
4. Diferentes entidades pueden tener el mismo valor.
Para métricas indirectas:
1. La métrica debe estar basada en un modelo explícitamente definido de relaciones entre ciertos atributos.
2. El modelo debe ser dimensionalmente consistente.
3. La métrica no debe mostrar ninguna discontinuidad inesperada.
4. La métrica debe usar unidades y escalas correctamente.
Según el contexto de aplicación las métricas se clasifican en Métricas de proceso, de proyecto y de producto.
El conjunto de métricas a usar debe dejar claro qué aspectos de la calidad son los que propone medir y a quién van dirigidos. Programadores, gestores y usuarios tienen diferentes puntos de vista de lo que significa calidad por lo que el conjunto de métricas a utilizar debería estar basado en un modelo de calidad bien definido [34].

3. ENFOQUES DE DESARROLLO DE SOFTWARE

3.1 Desarrollo de Software Orientado a Objetos

Las metodologías tradicionales estructuradas se basan en la descomposición de un sistema en módulos, atendiendo a consideraciones procedimentales y/o de datos. En el Desarrollo de Software Orientado a Objetos (DSOO) los sistemas se estructuran alrededor de los objetos que existen en el modelo del sistema [20].
Un objeto es una unidad que encapsula estructura y comportamiento. Si se logran identificar todos los objetos que intervienen en el sistema y sus relaciones, se obtendrá una aproximación mucho más exacta de la realidad.
La orientación a objetos se basa en tres principios básicos: todo son objetos, encapsulamiento / ocultación y herencia / polimorfismo. El primer principio indica la unidad básica de trabajo. El segundo permite englobar en un mismo concepto a los datos y a las operaciones. El tercero permite agrupar y tratar de igual forma a objetos similares [21].

3.2 Desarrollo de Software Basado en Componentes

Un componente es una unidad de composición de aplicaciones software que posee un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes, de forma independiente en tiempo y espacio [23].
El Desarrollo de Software Basado en Componentes (DSBC) tiene dos características importantes, la arquitectura del componente y el proceso de desarrollo del mismo. La arquitectura debe ser estandarizada para los componentes de software, ya que si no nó podrá llevarse a cabo la reutilización. En el proceso de desarrollo se considera al componente como un aspecto central para el desarrollo de software [24].
El ciclo de vida del desarrollo de sistemas basados en componentes [25] incluye, entre otras, las siguientes etapas:
• Análisis: Exploración y evaluación de componentes disponibles en el mercado; estudio y especificación de la organización y de sus requisitos; subsiguiente especificación de los modelos de proceso de negocio adecuados.
• Selección: Elección de una arquitectura de componentes que satisfaga los requisitos, de acuerdo con métodos y criterios adecuados y según la disponibilidad de componentes en el mercado y servicios de los proveedores.
• Contratación: Redacción del contrato formal según los requisitos establecidos sobre los componentes seleccionados, los resultados del análisis, su evaluación, y las condiciones de implantación.
• Implantación de los componentes: Ajuste e integración con otros componentes implantados en el sistema software bajo desarrollo.
Existen varios tipos de componentes, los componentes comerciales Commercial Off-The-Shelf (COTS), los de código abierto Free and Open Source Software (FOSS), componentes desarrollados a medida, los servicios web, etc.

3.3 Desarrollo de Software Orientado a Aspectos

El Desarrollo de Software Orientado a Aspectos (DSOA) [17] es un paradigma que provee una mejor separación de las capas y que conduce a la producción de software fácil de mantener y reutilizar. DSOA se basa en los aspectos como una abstracción destinada a modularizar los requerimientos transversales y mejorar la capacidad de mantenimiento del sistema y la reutilización.
Los aspectos son propiedades de un software que tienden a atravesar sus funcionalidades principales. Consideramos un aspecto a una unidad modular que se disemina por la estructura de otras unidades funcionales. Los aspectos existen tanto en la etapa de diseño como en la de implementación.
Un concerns es una parte del problema que se quiere tratar como una sola unidad conceptual [14]. Las abstracciones básicas del DSOO no alcanzan para separar los concerns especiales que se encuentran en sistemas con mayor complejidad. Estos requerimientos son denominados crosscutting concerns, estos requerimientos tienden a dispersarse y enredarse con el resto de las funcionalidades del sistema volviendo al código inteligible, difícil de mantener y reutilizar [13]. El DSOA proporciona un mecanismo para la separación de estos concerns mediante la utilización de aspectos como unidad de abstracción.

3.4 Metodologías Ágiles

El esquema tradicional para el desarrollo de software ha demostrado ser efectivo y necesario en grandes proyectos, sin embargo, este enfoque no resulta ser el más adecuado para los proyectos con requisitos muy cambiante, y que exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad [26].
Las restricciones de tiempo y flexibilidad hacen que las metodologías tradicionales presenten dificultades al ser utilizadas, es por eso que muchos equipos de desarrollo se resignan a prescindir de las buenas prácticas de la Ingeniería del Software. En este contexto, las Metodologías Ágiles (MA) [26] surgen como una posible solución a estos problemas. Por estar especialmente orientadas para proyectos pequeños, las MA constituyen una solución a medida para ese entorno, aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto.
Si bien el concepto de MA se remonta a los años 90, en el año 2001, durante una reunión de expertos en desarrollo de software surgió el término "Ágil" aplicado a la ingeniería de software, el objetivo primordial de los desarrolladores fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto [27].
Como resultado de esta reunión se creó The Agile Alliance, una organización dedicada a promover los conceptos relacionados con el desarrollo ágil de software.
Como características principales comunes a casi todos las MA se destacan:

- Los ciclos cortos de trabajo, de longitud fija determinada con anterioridad, con funcionalidades precisas. Se estima de acuerdo a las funcionalidades que se pueden realizar en un tiempo determinado.
- Una vez iniciado el ciclo no se permite añadir nuevas funcionalidades solo pequeños cambios sumamente necesarios en las especificaciones.
- Al final de cada ciclo se debe contar con código terminado y operativo, que implemente las funcionalidades planificadas para cada ciclo. La idea es comenzar a ver resultados desde un principio.
- Al finalizar cada ciclo se dedica un tiempo para reflexionar sobre el camino recorrido y lo que resta por recorrer, para contar con una visión general del proyecto.
- Participación activa del cliente, formando parte del equipo y estando físicamente cerca de los desarrolladores.
- Constante comunicación y realimentación entre todos los miembros del equipo.
- Reuniones de trabajo frecuentes entre todos los miembros del equipo. Cortas, y en la que cada uno aporta tres aspectos concretos de su trabajo, para que todos sepan lo que se está haciendo.

Los 12 principios del manifiesto ágil [28] destacan las siguientes premisas y marcan la diferencia con los procesos tradicionales:

- Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas.
- Desarrollar software que funciona más que conseguir una buena documentación.
- La colaboración con el cliente más que la negociación de un contrato.
- Responder a los cambios más que seguir estrictamente un plan.

<u>Los principios son:</u>
I. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporten un valor.
II. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.

III. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
IV. El grupo de negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
V. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.
VI. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
VII. El software que funciona es la medida principal de progreso.
VIII. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
IX. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
X. La simplicidad es esencial.
XI. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.
XII. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.
Las principales metodologías que comparten las bases de desarrollo son entre otras, XP, SCRUM, CRYSTAL, LEAN.

4. CALIDAD APLICADA A LOS ENFOQUES DE DESARROLLO

4.1 Calidad aplicada al DSOO

Las métricas tradicionales como "puntos de función" y "complejidad dicromática" [17], se han usado eficientemente en el paradigma procedural. Sin embargo, estas no aplican a los aspectos del paradigma orientado a objetos: clases, objetos, acoplamiento, etc.,
Las métricas orientadas a objetos se centran en las mediciones que se pueden aplicar a las características de encapsulamiento, ocultamiento de información, herencia y técnicas de abstracción de objetos que hacen única a una clase
Los objetivos principales de las métricas orientadas a objetos son los mismos que los existentes para las métricas surgidas para el software estructurado [18]:
• Evaluar mejor la calidad del producto.
• Estimar la efectividad del proceso.
• Mejorar la calidad del trabajo realizado en el nivel del proyecto.
Las métricas para sistemas orientados a objetos deben ser concordantes con las características que distinguen el software orientado a objetos del software convencional.
Se definen seis características que dan lugar a unas métricas especializadas [19]:
Localización
La localización es una característica de software que, indica la forma en que se concentra la información dentro de un programa. En el contexto orientado a objetos, la información se concentra mediante el encapsulamiento tanto de datos como de procesos dentro de los límites de una clase u objeto [18].
Encapsulamiento
Se define el encapsulamiento como “el empaquetamiento” (o enlazado) de una colección de elementos [19].
Ocultamiento de información
El ocultamiento de información suprime los detalles operativos de un componente de un programa. Tan solo se proporciona la información necesaria para acceder a ese componente o a aquellos otros componentes que deseen acceder a él [18].

Herencia

Le herencia es un mecanismo que hace posible que los compromisos de un objeto se difundan a otros objetos. Le herencia se produce a lo largo de todos los niveles de la jerarquía de clases [18].

Polimorfismo

El polimorfismo representa un concepto de teoría de tipos, en el que un solo nombre (tal como una declaración de una variable), puede denotar instancias de muchas clases diferentes, en tanto en cuanto estén relacionadas por alguna superclase común. Cualquier objeto denotado por este nombre es, por tanto, capaz de responder a algún conjunto común de operaciones, de diversas formas [18].

Técnicas de abstracción de objetos

La abstracción es un mecanismo que permite al diseñador centrarse en los detalles esenciales de algún componente de un programa sin preocuparse por los detalles de nivel inferior [18].

Las métricas orientadas a objetos hacen hincapié en conceptos tales como el encapsulamiento, herencia, complejidad de clases y polimorfismo. Por lo tanto las métricas orientadas a objetos se centran en las mediciones que se pueden aplicar a las características de encapsulamiento, ocultamiento de información, herencia y técnicas de abstracción de objetos que hacen única a una clase [17].

Métricas OO:

Las métricas que se detallan a continuación son las llamadas métricas tradicionales y adaptadas al DSOO:

WMC (Weighted Methods per Class)*:* Cuenta los métodos de una clase.

DIT (Depth in Inheritance Tree)*:* Esta métrica se define como la longitud del camino mas largo desde la clase raíz en la jerarquía de herencia.

NOC (Number Of Children): cuenta el número de clases que heredan de una clase dada, es decir, el número de clases en el árbol de herencia hijos de una clase determinada.

CBO (Coupling Between Object Classes): acoplamiento entre objetos, es el número de clases a las cuales una clase dada esta acoplada. Se da dependencia entre dos clases cuando una clase usa métodos o variables de la otra clase.

RFC (Response for a Class): es el número de métodos en el conjunto de respuesta de la clase

En [40] se describen y analizan un conjunto de métricas para el DSOO:

1) Chen

- *CCM* (Class Coupling Metric),
- *OXM* (Operating Complexity Metric),
- *OACM* (Operating Argument Complexity Metric),
- *ACM* (Attribute Complexity Metric),
- *OCM* (Operating Coupling Metric),
- *CM* (Cohesion Metric),
- *CHM* (Class Hierarchy of Method)
- *RM* (Reuse Metric).

2) Morris

Conjunto de métricas orientadas a objetos que mide la complejidad y la cohesión. Considera al sistema en forma de estructura de árbol, analizando la profundidad del mismo y la cantidad de subnodos.

3) Lorenz and Kidd

Propone una serie de métricas agrupadas en cuatro categorías, tamaño, herencia, internas y externas. Algunas métricas utilizadas son:

CS (Class size)
NOO (Number of operations overridden by a subclass)
NOA (Number of operations added by a subclass)
SI (Specialization index)
OS (Operation size)
OC (Operation complexity)
NP (Number of parameters per operation)

4) MOOSE

El conjunto de métricas CK se compone de seis indicadores que evalúan diferentes características del diseño orientado a objetos:

WMC (Weighted Methods per Class)
DIT (Depth of Inheritance Tree)
NOC (Number of children)
CBO (Coupling Between Objects)
RFC (Response for class)
LCOM Methods (Lack of Cohesion in)

5) EMOOSE

MPC (Message Pass Coupling)
DAC (Data Abstraction Coupling)
NOM (Number of Methods)
Size1: Se utiliza para calcular el número de línea de código.
Size2: Se utiliza para contar el número de atributos locales y el número de operación definida en la clase.

6) <u>Conjunto de métricas MOOD (Metrics for Object oriented Desing</u>) [34]

<u>Métricas a Nivel Sistemas:</u>

Encapsulación

MHF (Method Hiding Factor): Proporción de la suma de las invisibilidades de los métodos en todas las clases entre el número total de métodos definidos en el sistema.

AHF (Attribute Hiding Factor): Proporción entre los atributos definidos como protegidos o privados y el número total de atributos.

Herencia

MIF (Method Inheritance Factor): Proporción de la suma de todos los métodos en todas las clases.

AIF (Attribute Inheritance Factor): Proporción del número de atributos heredados entre el número total de atributos.

Polimorfismo

PF (Polymorphism Factor): Proporción entre el número real de posibles diferentes situaciones polimórficas para una clase, entre el máximo número posible de situaciones polimórficas en la clase.

Paso de Mensajes

CF (Coupling Factor): Proporción entre el máximo número posible de acoplamientos en el sistema y el número real de acoplamientos no imputables a herencia. Comunicación entre clases.

Métricas a Nivel de Acoplamiento:

CBO (Coupling Between Objects): Número de clases a las cuales una clase esta legiada.

Métricas a Nivel de Herencia:

DIT (Depth on Inheritance Tree): Mide el máximo nivel en la jerarquía de herencia.

NOC (Numbers of children): Número de subclases que pertenecen a una clase.

SIX (Specialisation Index per Class): Indice de especialización por clase que muestra en que medida las subclases redefinen el comportamiento de sus superclases.

Métricas a Nivel de Clases:

LCOM (Lack of Cohesion in Methods): Establece en qué medida los métodos hacen referencia a atributos.

RFC (Response For a Class): Cuenta las ocurrencias de de llamadas a otras clases desde una clase en particular.

WMC (Weighted Methods per Class): Número de métodos definidos en una clase.

Métricas a Nivel de Métodos:

LOC (Lines of code per Methods): Es el número de líneas de código activas en un método.

NOM (Number of Messager Send): Mide el número de mensajes enviados en un método, segregados por el tipo de mensaje (unarios, binarios o clave)

7) Goal Question

Este enfoque se definió originalmente para la evaluación de defectos para un conjunto de proyectos en el entorno de la NASA. Proporciona un marco que incluye tres pasos:

1. Enumere los principales objetivos del proyecto de desarrollo o mantenimiento.
2. Derivar de cada objetivo las preguntas que deben ser contestadas para determinar si se están cumpliendo los objetivos.
3. Decidir lo que se debe medir con el fin de ser capaz de responder adecuadamente a las preguntas.

Métricas (nivel cuantitativo): Un conjunto de datos se asocia a todas las preguntas con el fin de responder de una manera cuantitativa. Estos datos pueden ser objetivos y subjetivos.

8) LI

Propone seis métricas:
NAC (Number of Ancestor Classes)
NLM (Number of Local Methods)
CMC (Class Method Complexity)
NDC (Number of Descendent Classes)
CTA (Coupling Through Abstract data type)
CTM (Coupling through Message Passing)

4.2 Calidad aplicada al DSBC

El DSBC proporciona un método para construir sistemas haciendo uso de componentes reutilizables. Aumenta la eficiencia de los costos durante el desarrollo, proporciona una mayor fiabilidad en cuanto a su funcionamiento y disminuye la presión a la hora del mantenimiento. [23]

El estudio de la calidad de los componentes software juega un papel muy importante en el DSBC. Por ejemplo, en los procesos de selección de tales componentes es necesario conocer con detalle el comportamiento relativo a aquellos criterios que se corresponden con los requisitos del sistema en desarrollo, tanto funcionales como no-funcionales (p.e., respecto

eficiencia, usabilidad, etc.). Necesidades similares aparecen en otras actividades como en el momento de integrar los componentes, de mantener el sistema, etc.[25]
Los factores que generan estas diferencias son:
• Atomicidad: los componentes son unidades indivisibles desde el punto de vista de su gestión. Consecuentemente, podemos estudiar su calidad individualmente.
• Reusabilidad: los componentes son módulos que se reusan e integran, en una o más aplicaciones. Ello exige un alto grado de precisión en la descripción de la calidad, especialmente en el caso de componentes COTS y servicios web.
• Evolución: los componentes que integran la aplicación transitan por sucesivas versiones que no se corresponden necesariamente con las versiones de los sistemas en los que se integran, especialmente en el caso de los mencionados COTS. La descripción de la calidad de los componentes debe facilitar el estudio del impacto de tales evoluciones. [25]
A continuación se presenta un modelo de calidad adaptado a componentes el cual se basa en el Modelo genérico ISO/IEC 9126-1 desarrollado por [22]. Para adaptar este modelo se determinaron las sub-características que son relevantes para componentes, y cual se debían particularizar, y posteriormente se definieron métricas e indicadores específicos para las características de calidad involucradas.
En este modelo desaparecen la característica Portabilidad y las sub-características Tolerancia a Fallos, Estabilidad y Analizabilidad. Aparecen como nuevas las sub-características Compatibilidad y Complejidad. Algunas de ellas cambian su sentido:
Funcionalidad: esta característica mantiene el mismo sentido para los componentes que para un producto software. Es la capacidad del componente para proporcionar las funciones que satisfagan las necesidades establecidas o implícitas cuando se usa bajo las condiciones especificadas.
Compatibilidad: indica con que versiones predecesoras es compatible un componente, es decir, si se mantiene la funcionalidad del mismo al integrarse en el contexto donde estaba la versión anterior que se desea sustituir.
Fiabilidad: es aplicable directamente a los componentes, y fundamental para su reutilización.
La sub-característica Madurez se mide en función de los cambios que sufren las versiones comerciales y la velocidad a la que aparecen.
Recuperabilidad: en función de una serie de atributos que pueden estar presentes o no en su diseño, indicando los métodos que se utilizan para implementarlos.
Facilidad de Uso y todas sus sub-características cambian de sentido, dado que un componente no será utilizado por un usuario final directamente sino por los diseñadores y desarrolladores de aplicaciones software. La facilidad de uso real de un componente debe interpretarse como la capacidad del componente para ser utilizado en la construcción de un producto o sistema software.
Complejidad: una nueva sub-característica cuyo sentido es dar una medida de la complejidad del uso e integración del componente en un producto o sistema software.
Eficiencia se respeta la definición y clasificación que hace la ISO de esta característica, aunque otros autores prefieren hablar de Rendimiento (Performance) y usan otra sub-clasificación.
Mantenibilidad: mide la capacidad de un producto de software de ser modificado, entendiendo por modificación cualquier corrección, mejora o adaptación del software. Por ello, las sub-características Cambiabilidad y Facilidad de Prueba son las que deben ser medidas para los componentes.
Portabilidad: es la capacidad de poder ser reutilizado en distintos entornos, esa es la esencia misma de los componentes, que son diseñados y desarrollados específicamente para ser reutilizados.

Para este Modelo de Calidad existen atributos medibles durante el Tiempo de Ejecución (TE) de los componentes y otros que se miden durante todo el Ciclo de Vida (CV). Estos atributos se encuentran asociados a las sub-características que componen el modelo, y se agrupan de acuerdo al momento en el que son observados.

Métricas::

Presencial: Esta métrica indica si un atributo está presente en el componente o no. Para ello se utiliza una variable booleana y una variable de tipo string que especifique como o con que método, formato etc, se implementa el atributo en cuestión en caso de estar presente.

Tiempo: Esta métrica se utiliza para medir intervalos de tiempo, utiliza una variable de tipo entero para indicar el valor absoluto y una variable de tipo string para indicar las unidades, por ejemplo, 10 segundos o 4 meses.

Nivel: Se utiliza para indicar un grado de esfuerzo, habilidad, etc., cuando se da una medida subjetiva dentro de una escala de valores. Es una variable entera que puede tomar el siguiente rango de valores: 0 (Muy Bajo), 1 (Bajo), 2 (Medio), 3 (Alto), 4 (Muy Alto).

Ratio: Se utiliza para dar un porcentaje (entre 0 y 100).

Entero: variable de tipo entero.

Además de estas métricas básicas, para medir algunos atributos se utiliza también *índices*, que se obtienen a partir de dos métricas básicas, generando un "*indicador*". En general, es conveniente distinguir entre métricas básicas e indicadores, pues estos últimos son valores derivados de los primeros.

Indicadores para medir la usabilidad del software:

Atributos:

- *Contenido de los manuales*

  Indicadores:

  1. Cobertura del manual

  Métricas:

  - Porción de elementos funcionales descriptos en el manual

  2. Consistencia del manual

  Métricas:

  - Proporción de elementos funcionales incorrectamente descrito en el manual
  - Integridad de los manuales
  - Diferencia entre la versión del componente y la versión manual

  3. Legibilidad del manual

  Métricas:

  - Relación de figuras por páginas del manual
  - Relación de tablas por páginas del manual
  - Relación de diagramas UML por páginas del manual

- *Tamaño de los manuales*

  1. Idoneidad del Manual

  Métricas:

  - Promedio de páginas por elementos funcionales

- *Eficiencia de los manuales*

  1. Relación de Eficacia

  Métricas:

  - Proporción de elementos funcionales que se utilizan correctamente después de leer el Manual

  2. Relación de Comprensibilidad

  Métricas:

  - Proporción de elementos funcionales correctamente interpretados después de leer el Manual

Conjunto de Métricas y factores de calidad propuesto en [12]
CPD (Component Packing Density)
CID (Component Interaction Density)
CM (Criticality Metrics)
NC (Number of Cycle Metrics)
AC (Active Component Metrics)
Este conjunto de métricas fue evaluado utilizando las nueve propiedades Weyuker, las originales se encuentran disponibles en [44] y fueron modificadas en [12].
Las métricas presentadas pueden ser utilizadas en diferentes etapas del ciclo de desarrollo del software, proporcionando mejoras en el diseño, en la integración de tareas complejas o críticas y como extender componentes ya desarrollados.

En [44] se describen las métricas para caracterizar el efecto de dependencia en la estructura de diseño en los Sistemas de Software Basados en Componentes:
Métricas para acoplamiento de componentes: se entiende por Acoplamiento el impacto de la dependencia de X2 a X1, esto significa que cuando se modifica X1, habrá impacto en X2.
IC (Interface Coupling)
CC (Component Coupling)
CBCC coupling (Component-based software System Coupling)
Métricas de flujo de información de componentes: el objetivo de esta medida es predecir los componentes críticos. Un componente crítico es aquel que es más propenso a contener errores durante las pruebas.
IIF (Interface Information Flow)
CIF (Component Information Flow)
CBSIF (CBSS Information Flow)

La reutilización del software en general se considera como una de las maneras más eficaces de aumentar la productividad y mejorar la calidad de software. Para hacer que la reutilización del software ocurra, debe haber un cambio en la forma de desarrollo [45].
Métricas importantes y relevantes aplicables para el análisis de calidad de los componentes:
CICM (Component Interface Complexity Metric)
CSM (Component Size Metric)
CPM (Component Portability Metric)
CRM (Component Reliability Metric)
CFM (Component Functionality Metric)
CCSM (Component Customer Satisfaction Metric)
CCM (Component Cost Metrics)
Métricas para el ensamblaje de componentes:
CIM (Component Interaction Metric)
AIM (Actual Interactions Metric)
TIPM (Total Interactions Performed Metric)
CpIM (Complete Interactions Metric)
DCCM (Direct Class Coupling Metrics)
IDCCM (Indirect Class Coupling Metrics)
DCMCM (Direct Component Coupling Metrics)
IDCMCM (Indirect Component Coupling Metrics)

4.3 Calidad aplicada al DSOA

Se propone un modelo de calidad y un grupo de métricas [13] para el DSOA que permiten reunir información sobre el código y el diseño de los atributos fundamentales para aspectos, la separación de concern, el acoplamiento, la cohesión y el tamaño. Se adaptan métricas clásicas y las propuestas para el desarrollo orientado a objetos para poder medir estos atributos.
El Modelo presentado define terminología y relaciones entre la reutilización, mantenibilidad y un conjunto de métricas. Se basa en una extensa revisión de modelos existentes, definiciones clásicas de atributos de calidad y teorías tradicionales de diseño.
Los atributos planteados para el DSOA están construidos en forma de árbol, ya que la calidad se conforma de muchas cualidades, estas cualidades internas son denominadas factores. Este modelo también conecta los atributos internos con el conjunto de métricas
El conjunto de métricas generado para el DSOA y presentado en [13] se compone de cinco métricas de diseño y cinco métricas para código, y son agrupadas de acuerdo al atributo que miden. Este conjunto puede ser adaptado y utilizado para la medición de otros atributos tales como fiabilidad y capacidad de prueba.
Este grupo se compone de cinco métricas de diseño y cinco métricas de código, agrupadas de acuerdo a los atributos que miden: separación de concerns (SoC), acoplamiento, cohesión y tamaño.

Separación de concerns
Capacidad de identificar, encapsular y manipular las partes de software que son relevantes para un requerimiento especial [14].

Métricas:
CDC (Concern Diffusion over Components): es una métrica de diseño que cuenta el número de los componentes principales que contribuyen a la aplicación de un requerimiento.
CDO (Concern Diffusion over Operations): cuenta el número de operaciones primarias cuyo objetivo principal es contribuir a la aplicación de un requerimiento.
CDLOC (Concern Diffusion over LOC): cuenta el número de puntos de transición de cada problema, a través de las líneas de código. La idea de esto es identificar los requerimientos cambiantes. Indica cuan entremezclado se encuentra el código.

Cohesión
Medida de la proximidad de la relación entre sus componentes internos [15].

Métricas:
CBC (Coupling between Components): se define para un componente, clase o aspectos. Cuenta el número de componentes a los que se encuentra acoplado.
DIT (Depth of Inheritance Tree): se define como la longitud máxima de un nodo a la raíz del árbol. Cuenta los niveles de jerarquía en la declaración de herencia de una clase o aspecto.

Acoplamiento
Indicación de la fuerza de las interconexiones entre los componentes en un sistema [15].

Métricas:
LCOO (Lack of Cohesion in Operations): Este indicador mide la falta de cohesión de uncomponente.

Tamaño del Software
Mide físicamente la longitud de diseño y el código del software [16].

Métricas:
VS (Vocabulary Size): cuenta el número de componentes del sistema, es decir, el número de clases y aspectos en el sistema. Este indicador mide el tamaño del

vocabulario del sistema. Cada nombre de componente se cuenta como parte del vocabulario del sistema, las instancias no se cuentan.
LOC (Lines of Code): cuenta el número de líneas de código. Esta es la medida tradicional de tamaño. La documentación y aplicación de comentarios, así como líneas en blanco, no se interpretan como código.
NOA (Number of Attributes): cuenta el número de atributos de cada clase o aspecto. Los atributos heredados no se cuenta.
WOC (Weighted Operations per Component): Esta métrica mide la complejidad de un componente en términos de sus operaciones.

4.3.1 Métricas Bad-Smell

Este conjunto de modelo propone un conjunto de métricas que contiene indicadores para identificar el código bad-smell oculto en el software. Bad-smell se utiliza metafóricamente para describir patrones de software asociados generalmente a un mal diseño y mala programación en la programación orientada a objetos [42]. Propone 15 métricas, seis a nivel Pointcut, 8 a nivel Aspecto y 1 a nivel clase. Estas métricas fueron validadas sobre aplicaciones simples, y se detallan a continuación:

Nivel Pointcut
NAdP (Number of Advices refer to a Pointcut)
NAdAsP (Number of Advices in Aspect refer to a Pointcut)
NSAdP (Number of Subaspect Advices refer to an aspect Pointcut)
NNSAdP (Number of Nom-Subaspect Advices refer to an aspect Pointcut)
SJP (Set of the corresponding Joinpoints of a Pointcut)
NOAsP (Number of Other Aspects refer to a Pointcut)

Nivel Aspecto
NPAs (Number of Pointcuts defined in a Aspect)
NNPAs (Number of Named Pointcuts defined in a Aspect)
NUPAs (Number of Unnamed Pointcuts defined in a Aspect)
SCT (Set of the inherited Classes of a given Type)
NAMA (Number of introduced Abstract Methods in an Aspect)
NAdAs (Number of Adviced in Aspect)
NIAs (Number of Introductions in Aspect)
SNOAsP (Sum of NOAsP)

Nivel Clase
NPC (Number of Pointcuts definded in a Class)

5. COMPARACIONES

En el Cuadro 7 se presentan los diferentes enfoques de desarrollo analizados anteriormente describiendo las métricas utilizadas para medir los atributos internos asociados a cada uno de ellos.

ENFOQUE	FACTORES DE CALIDAD	ATRIBUTOS INTERNOS	VARIABLES DE MEDICION	METRICAS	REFERENCIAS
DSOO	• **Adaptabilidad**	**Localización**	1. Tamaño de una clase	1. RFC 2. WMC	[18]
	• **Reusabilidad**	**Encapsulamiento**	2. Profundidad y ancho de una Jerarquía de herencia de clases	1. DIT 2. NOC	[18]
	• **Reusabilidad**	**Ocultamiento de información**	3. Grado de Acoplamiento entre clases	1. CBO	[18]
	• **Mantenimiento** • **Fiabilidad** • **Comprensibilidad**	**Herencia**			[18]
		Polimorfismo			[18]
		Técnicas de abstracción de objetos.			[18]
		Nivel Sistemas *1. Encapsulación*		1. MHF 2.AHF	[35]
		Nivel Sistemas *2. Herencia*		1. MIF 2.AIF	[35]
		Nivel Sistemas *3.Polimorfismo*		1.PF	[35]
		Nivel Sistemas *4.Paso de Mensajes*		1.CF	[35]
		Nivel Acoplamiento		1. CBO	[36]

		Nivel Herencia		1.DIT 2.NOC 3:SIX	[36] y [37]
				1.LCOM 2.RFC 3.WMC	[36]
				1.LOC 2.NOM	[37]
				1. CCM 2. OXM 3. OACM 4. ACM 5. OCM 6. CM 7. CHM 8. RM	Conjunto de Métricas Chen [40]
			1. Tamaño 2. Herencia 3. Internas 4. Externas	1. CS 2. NOO 3. NOA 4. SI 5. OS 6. OC 7. NP	Conjunto de Métricas Lorenz and Kidd [40]
		Nivel de Clases		1 .WMC 2. RFC 3. LCOM	Conjunto de Métricas CK [40][41]
		Nivel Herencia		1. DIT 2. NOC	Conjunto de Métricas CK [40][41]
		Nivel Acoplamiento		1. CBO	Conjunto de Métricas CK [40][41]
				1. MPC 2. DAC 3. NOM 4. Size1 5. Size2	Conjunto de EMOOSE [40]

				1. NAC 2. NLM 3. CMC 4. NDC 5. CTA 6. CTM	Métricas Li [40]
DSBC	• **Funcionabilidad**	**Corrección (TE)**	1. Presión 2. Exactitud Computacional	Ratio Ratio	[22]
		Seguridad (TE)	1. Cifrado de datos 2. Capacidad de Control 3. Capacidad para Auditar	Presencial Presencial Presencial	[22]
		Idoneidad(CV)	1. Cobertura 2. Exceso 3. Cobertura de Implementación	Ratio Ratio Ratio	[22]
		Interoperatibilidad/Conformidad (CV)	1. Compatibilidad de datos 2. Conformidad con estándares 3. Certificaciones	Presencial Presencial Presencial	[22]
		Compatibilidad (CV)	1. Compatibilidad hacia atrás	Presencial	[22]
	• **Fiabilidad**	**Recuperabilidad (TE)**	1. Secuenciabilidad 2. Persistencia 3. Transaccional 4. Tratamiento de Errores	Presencial Presencial Presencial Presencial	[22]
		Madurez (CV)	1. Volatilidad 2. Evolucionabilidad Fallos Eliminados	Tiempo Entero Entero	[22]

	• **Usabilidad**	**Facilidad de Aprendizaje (CV)**	1. Periodo para usar correctamente 2. Periodo para configurar correctamente 3. Periodo para administrar correctamente 4. Periodo para dominar	Tiempo Tiempo Tiempo Tiempo	[22]
		Facilidad de Comprensión (CV)	1. Documentación de usuario 2. Sistema de Ayuda 3. Documentación computacional 4. Formación 5. Cobertura de la demostración	Nivel Nivel Presencial Presencial Ratio	[22]
		Operatividad (CV)	1. Esfuerzo para operar 2. Esfuerzo para configurar 3. Esfuerzo para administrar	Nivel Nivel Nivel	[22]
		Complejidad (CV)	1. Interfaces ofrecidas 2. Interfaces externas utilizadas 3. Indice de complejidad	Entero Entero Indice	[22]
	• **Eficiencia**	**Comportamiento temporal (TE)**	1. Tiempo de Respuesta 2. Capacidad de Emisión 3. Capacidad de Recepción	Tiempo Entero Entero	[22]
		Utilización de Recursos (TE)	1. Requisitos de Memoria 2. Utilización de Disco	Entero Entero	[22]

	• **Mantenibilidad**	**Cambiabilidad (CV)**	1. Modificabilidad 2. Indice de modificabilidad 3. Capacidad de control del cambio	Entero Indice Nivel	[22]
		Facilidad de Prueba (CV)	1. Auto-test de arranque 2. Batería de pruebas	Presencial Presencial	[22]
	• **Usabilidad**	**Contenido de los Manuales**	1. Cobertura del Manual	Porción de elementos funcionales descriptos en el manual	[22]
			2. Consistencia del Manual	Proporción de elementos funcionales incorrectamente descrito en el manual Integridad de los manuales Diferencia entre la versión del componente y la versión manual	[22]
			3. Legibilidad del Manual	Relación de figuras por páginas del manual Relación de tablas por páginas del manual Relación de diagramas UML por páginas del manual	[22]
		Tamaño de los Manuales	1. Idoneidad del Manual	Promedio de páginas por elementos funcionales	[22]

		Eficiencia de los Manuales	1. Relación de Eficacia	Proporción de elementos funcionales que se utilizan correctamente después de leer el Manual	[22]
			2. Relación de Comprensibilidad	Proporción de elementos funcionales correctamente interpretados después de leer el Manual	[22]
	• **Usabilidad** • **Eficiencia** • **Mantenibilidad** • **Flexibilidad** • **Reusabilidad** • **Interoperabilidad** • **Portabilidad** • **Comprobabilidad** • **Confiabilidad**		1. Volumen de E/S 2. Velocidad E/S 3.Eficiencia en mantenimiento 4.Eficiencia en la ejecución 5.Tolerancia a error 6.Simplicidad 7.Instrumentación General 8.Modularidad 9.Capacidad de expansión Generalidad	1. CPD 2. CID 3. CM 4. NC 5. AC	[12]
	• **Reusabilidad** • **Mantenibilidad**	**Acoplamiento de componentes**		1. IC 2. CC 3. CBSS Coupling	[44]
	• **Reusabilidad** • **Mantenibilidad**	**Flujo de Información de componentes**		1. IIF 2. CIF 3. CBSIF	[44]

	• **Reusabilidad**		Métricas para el análisis de calidad de componente	1. CICM 2. CSM 3. CPM 4. CRM 5. CFM 6. CCSM 7. CCM	[45]
		Complejidad de Integración Rendimiento	Métricas para ensamblaje de componentes	1. CIM 2. AIM 3. TIPM 4. CpIM 5. DCCM 6. IDCCM 7. DCMCM 8. IDCMCM	[45]
DSOA	• **Reusabilidad** • **Mantenibilidad** • **Comprensibilidad** • **Flexibilidad**	**Separación de concerns**		9. CDC 10. CDO 11. CDLOC	[13]
	• **Reusabilidad** • **Mantenibilidad** • **Comprensibilidad** • **Flexibilidad**	**Cohesión**		1. LCOO	[13]
	• **Reusabilidad** • **Mantenibilidad** • **Comprensibilidad** • **Flexibilidad**	**Acoplamiento**		1. CBC 2. DIT	[13]
	• **Reusabilidad** • **Mantenibilidad** • **Comprensibilidad**	**Tamaño del Software**		1. VS 2. LOC 3. NOA 4. WOC	[13]
		Nivel Pointcut		1. NAdP 2. NAdAsP 3. NSAdP 4. NNSAdP 5. SJP 6. NOAsP	[42]

		Nivel Aspecto		1. NPAs 2. NNPAs 3. NUPAs 4. SCT 5. NAMA 6. NAdAs 7. NIAs 8. SNOAsP	[42]
		Nivel Clase		1. NPC	[42]

Cuadro 7 – Métricas por enfoque

6. CALIDAD APLICADA A LAS METODOLOGIAS AGILES

En esta sección se presentan ciertas consideraciones respecto del concepto de calidad aplicado a las MA, las mismas fueron tratadas separadamente de los paradigmas de desarrollo analizados por tratarse de una metodología relativamente nueva, y que si bien actualmente, se encuentra en pleno auge, las modelos de calidad existentes resultan obsoletos para ser aplicados por no coincidir, en su mayoría, con las variables de medición y atributos internos.
Es importante mencionar que en el proceso de desarrollo de las metodologías tradicionales el estricto cumplimiento, según lo planificado, de las variables tiempo y presupuesto conlleva a obtener un producto de calidad, mientras que en las MA la modificación de alguna de las variables tiempo, costo y calidad no genera necesariamente un problema siempre que se cuente con la colaboración del cliente [30].
Las iteraciones reducen el tiempo de desarrollo, entregando en la finalización de cada una, un subproducto usable y de calidad.
Las MA brindan un medio de control de la imprevisibilidad que es la adaptabilidad. También rechazan las métricas como herramienta de control del rendimiento de las personas, puesto que no se han encontrado métricas fiables en este sentido. Por ello favorecen la gestión delegatoria, donde los que finalmente hacen el trabajo deciden sobre cómo hacerlo.
Las MA están orientadas a la productividad, es por eso que cuando se analizan la calidad de los procesos y productos no siempre se obtienen resultados óptimos. La forma de alcanzar una calidad aceptable es mediante la interacción constante con el cliente que permite mejorar con cada iteración el producto.
Existe una aparente falta de elementos y/o artefactos que apoyen el mantenimiento de las aplicaciones.
Algunos factores importantes que influyen en el proceso de mantenimiento son [29]:

- La complejidad del software.
- El tamaño del software.
- El conocimiento que del software tengan los desarrolladores.

Estos factores son impactados por prácticas de las MA de desarrollo de software, permitiendo reducir el efecto negativo de los mismos en el mantenimiento y conservando la agilidad.
Algunas de estas prácticas que afectan [29]:
1. **TDD** (**Desarrollo dirigido por pruebas):** es una técnica de desarrollo en la que no se implementa una funcionalidad hasta que no se haya escrito la prueba para la misma. Aporta mejorando el mantenimiento preventivo y reduciendo el mantenimiento correctivo del software.
2. **Refactorización**: consiste en reorganizar el código para que sea más limpio y por ende más fácil de mantener. Con esta técnica no se pretende cambiar los resultados del código sino la forma en que se obtienen. Esta técnica también es tomada de la metodología XP y aporta a la categoría de mantenimiento perfectivo del software.
3. **Priorización**: esta práctica, proveniente de XP y de SCRUM, tiene como objetivo que los stakeholders del negocio decidan cuáles son los requisitos que en realidad se deben desarrollar como prioridad. Con esta práctica se pretende que el software sea tan grande como realmente se requiera, aportando al mantenimiento preventivo del mismo.
4. **Diseño simple**: en XP se promueve el diseño simple, en el cual no se tienen en cuenta funcionalidades que podrían requerirse en el futuro, sino únicamente las funcionalidades requeridas en la actualidad. Esta práctica previene el incremento innecesario de la complejidad del software, uno más de los factores que afectan negativamente el mantenimiento del mismo.
5. **Conocimiento del software**: las metodologías ágiles se basan en la interacción cara a cara entre el equipo de desarrollo y los stakeholders del negocio, por lo cual, el conocimiento del

proyecto y del software se da de una manera tácita en los miembros del equipo. El conocimiento del software es uno de los más importantes factores que afectan positivamente el mantenimiento.

Métricas:

Existen métricas que no aportan ningún valor dentro del desarrollo ágil, como la cantidad de líneas de código (KLOC), tareas completas o tiempo dedicado a cada tarea. Para las MA se presentan otro tipo de métricas tales como [33]:

Reafirmar y reforzar los principios Agiles: si el software funciona es la principal medida de progreso.

Medir los resultados, no solo las salidas.

Seguir las tendencias no los números: evitar estar continuamente mirando los indicadores del proyecto, y centrarse en producir software, fijar el rumbo y dirigirse a la meta.

Centrarse en un pequeño número de indicadores: centrarse en la producción de artefactos de software que realmente importan (el código), y conseguirlo en el menor tiempo posible.

Indicadores y métricas fáciles de recoger: centrarse en la producción del software, no en informar sobre su producción.

Un enfoque más congruente y a medida es el que se presenta en [43] para las MA, en él se proponen una serie de herramientas sencillas para aplicar en un determinado proyecto ágil.
A continuación se describe una serie de pruebas que pueden ser aplicadas para medir procesos ágiles:

1. Afirma y refuerza, Lean y principios ágiles
2. Medidas de resultado, no de salida
3. Sigue las tendencias, no números
4. Pertenece a un pequeño conjunto de métricas
5. Fácil de recolectar
6. Releve, más que oculte, su contexto y variables significativas
7. Provea el combustible para una conversación significativa
8. Provea retroalimentación regular y frecuentemente
9. Puede medir proceso o producto
10. Aliente a una buena calidad

Este enfoque propone la creación de una métrica principal, lo ideal es que esta métrica se establezca por gestión y decisión ejecutiva y debe permitir controlar la rendición de cuentas. Los métodos ágiles alientan a las empresas a ser responsables por el valor producido por el esfuerzo del desarrollo de software. Una vez que está en marcha la métrica clave se propone diseñar métricas de apoyo denominadas “diagnostico” ya que se utilizan para diagnosticar y mejorar los procesos que producen el valor del negocio. Un ejemplo de un diagnóstico contextual: un equipo decide tomar medidas para mejorar el "flujo" de su trabajar por la reducción de interrupciones.

Las métricas tales como el EVA (análisis de valor del trabajo) y el costo para realizarlo son medidas muy importantes para el desarrollo ágil [10]. El valor a largo plazo no se refiere al valor del negocio, el término se refiere al valor según lo expresado por el presupuesto del proyecto o programa.

Métricas

PV (Valor Planificado): El valor de la obra planeada para ser realizada con base en el presupuesto (en dólares o en horas)

EV (Valor Ganado): El valor integrado de trabajo efectivamente realizado en función del

presupuesto (en dólares o en horas).
AC (Costo Actual): El coste incurrido para ese incremento de trabajo.
BAC (Presupuesto completo): El presupuesto asignado para completar el trabajo.
ETC (Estimación hasta la conclusión): El importe previsto para completar el trabajo restante (en dólares u horas), con base en los resultados anteriores.
EAC (Estimación Completa): El importe total previsto para todo el trabajo en el plan del proyecto, con base en los resultados anteriores.

- Valor Planificado = BAC * Porcentaje completo planificado
- Valor Ganado = BAC * Porcentaje completo actual
- Indice de rendimiento de Costo (CPI) = EV/AC
- Indice de rendimiento del cronograma(SPI) = PV/AC
- ETC = (BAC-EV)/CPI
- EAC = BAC/CPI o AC+ETC

ANALISIS DE RESULTADOS

En el Cuadro 7 se comparan los modelos creados o adaptados para los enfoques de desarrollo presentados en la Sección 3, analizando los atributos y variables internas medidos para cada factor considerado de importancia para el paradigma. Las métricas son propias de cada modelo de calidad y se crean para medir los criterios definidos en él, es por ello que se enumeran las utilizadas por cada autor para medir el atributo analizado.
Cada enfoque cuenta con una serie de atributos internos propios y particulares, medibles durante el tiempo de ejecución o durante todo el ciclo de vida, cada modelo analizado presenta una serie de métricas utilizadas para medir las variables asociadas a los diferentes atributos.
Algunos de los documentos analizados describen los modelos claramente, permitiendo identificar los factores de calidad, atributos y variables internas de medición junto con las métricas aplicadas para ello. En otros casos hacen incapié en las métricas sin detallar explícitamente los factores que componen el modelo.
Del análisis anterior se desprende la cantidad de métricas incluidas en los modelos analizados para cada uno de los diferentes enfoques. El Gráfico 1 representa este número de métricas y permite observar que el mayor número es para DSOO y DSBC, esto se debe a que son los enfoques más conocidos y utilizados actualmente. Cabe mencionar, que si bien las MA no fueron incluidas dentro de la comparación, por no seguir las mismas líneas a la hora de medir la calidad en el proceso de desarrollo y en el producto obtenido, también se representó gráficamente la cantidad de métricas analizadas.

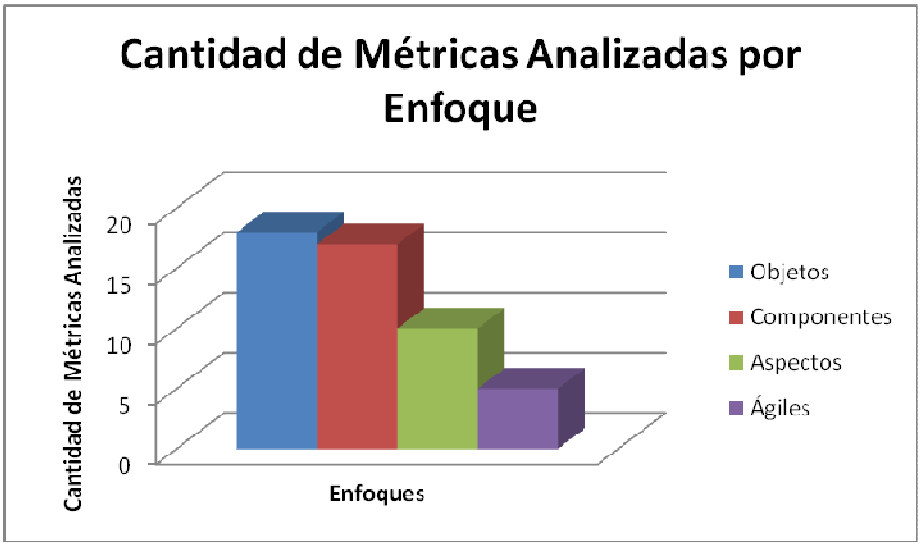

Gráfico 1 – Métricas por enfoque

Del análisis de los modelos de calidad presentados en este trabajo para cada uno de las metodologías de desarrollo de software, se confeccionó el Cuadro 8 que indica los factores de calidad y atributos internos que se priorizan y se utilizan en los diferente enfoques.

Factores de Calidad	**Enfoques**			
	DSOO	**DSBC**	**DSOA**	**MA**
Funcionalidad		X		X
Usabilidad		X		
Mantenibilidad	X	X	X	X
Eficiencia		X		
Fiabilidad	X	X		
Adaptabilidad	X			
Reusabilidad	X		X	
Comprensibilidad	X		X	
Flexibilidad			X	

Cuadro 8 – Factores de Calidad por enfoque

En este cuadro se observa que el factor Mantenibilidad es considerado de importancia en los paradigmas analizados y en las MA de desarrollo.

En relación a lo presentado en el Cuadro 8 se puede concluir que los MC analizados para cada enfoque, cubren diferentes factores de calidad, siempre tomando como referencia a los modelos de calidad estándares, pero en su mayoría el factor Mantenibilidad se encuentra presente en la mayoría de ellos y también en las MA.
Se utilizan diferentes métricas aplicadas para medir este factor, dependiendo del enfoque para el cual se desarrolló o adaptó el modelo. Estas métricas utilizadas, en general fueron creados para medir los atributos en consideración con las variables internas de cada enfoque, pero algunas de ellas son utilizados por varios modelos desarrollados para un mismo paradigma o adaptadas para enfoques diferentes.

CONCLUSIONES

Los estándares de calidad de software surgen con la necesidad de obtener productos libres de fallas, que satisfagan las necesidades del cliente y así lograr su satisfacción. Si bien los factores de calidad dependen del modelo, existes algunos estándares, los cuales sirven de base a la hora de adaptar o crear nuevos modelos de calidad.

Es importante contar con un modelo de calidad propio y adecuado al enfoque utilizado para el desarrollo, que se centre especialmente en los detalles de calidad particulares para la metodología. Es necesario identificar el conjunto de atributos de calidad y métricas para realizar su evaluación y que se ajusten exclusivamente a la metodología a utilizar, ya sea, por una particularización de un modelo general existente como las ISO o la creación de uno nuevo basándose en los ya existentes.

El análisis de los modelos ya existentes para los diferentes enfoques, permitió realizar una comparación entre ellos identificando factores, indicadores internos, variables de medición y métricas aplicadas para cuantificar esas variables.

Se analizaron más de 50 métricas en torno a los 9 factores de calidad identificados para los 3 enfoques de desarrollo y las metodologías ágiles. Como resultado de este análisis y las comparaciones obtenidas se puede concluir que del conjunto de factores de calidad identificados en cada modelo existen algunos como la Mantenibilidad que se incluye en la mayoría de los modelos, independientemente del enfoque de desarrollo, esto indica que es un punto importante a tener en cuenta a la hora de medir la calidad.

También se puede observar que el mayor número de métricas encontradas es sobre la base de los DSOO y DSBC, esto se debe a que son metodologías afianzadas y muy utilizadas actualmente para desarrollar software. Sin embargo, para el DSOA y las MA, la presencia de MC aplicados es notablemente menor, esto indica, que si bien su aplicación busca obtener procesos y producto de calidad aceptable, todavía se encuentran en etapa de maduración. Cabe mencionar que algunos modelos analizados para aspectos intentan adaptar las métricas utilizadas para el DSOO.

En relación a las MA no se encontró un MC desarrollado, sin embargo se describen criterios a tener en cuenta para lograr productos de calidad aceptable utilizando esta metodología. Medidas de calidad heredadas de otras metodologías pueden obstaculizar el alto rendimiento de las MA. La idea es promover la creación de una medida principal de progreso, en [43] se recomienda la elección de una métrica clave que está estrechamente ligada a la economía de la inversión. Todas las métricas subordinadas deben considerarse como meros instrumentos para lograr el éxito.

Como trabajo futuro, se pretende profundizar en el factor de calidad “usabilidad” y las métricas asociadas para su aplicación a un software de desarrollo.

REFERENCIAS.

[1] Pressman, R. “Ingeniería del Software. Un enfoque práctico”, 5ta edición, McGraw–Hill Interamericana, España, 2002.

[2] Humphrey, W., Introduction to the Personal Software Process, Addison Wesley Longman, Inc., Massachusetts, 1997.

[3] Pressman, R. “Ingeniería de Software. Un enfoque práctico”,1995

[4] Dinora S., Jiménez O., “Calidad de Software en el uso de Metodologías Ágiles para el Desarrollo de Software”, Centro de Investigación en Computación, Instituto Politécnico

Nacional, Av. Juan de Dios Bátiz s/n esquina Miguel Othón de Mendizábal, Col. Nueva Industrial Vallejo, México.
[5] McCall, J.A., Cavano, J.P., "A Framework for the Measurement of Software Quality", ACM Software Quality Assurance Workshop, 1978.
[6]Olsina, Luis. "Ingeniería Web; Marco de medición y evaluación de calidad". Departamento de informática.Universidad Nacional de San Luis - La Rioja – Catamarca, 2007
[7] Bohem, B.W. , "Software Engineering Economics", Prentice Hall, 1981.
[8] ISO/IEC 9126: "Software Engineering - Product quality", International Organization for Standardization, 2000.
[9] Briand, L., El Emam, K., Morasca, S. "Theoretical and Empirical Validation of SoftwareProduct Measures". Technical Report ISERN-95-03, Fraunhofer Institute for Experimental Software Engineering, Germany, 1995.
[10] Sitio web de la Alianza de Scrum de Metodologías de Desarrollo Agiles http://www.scrumalliance.org/community/articles/2013/july/agile-project-reporting-and-metrics, Diciembre 2013.
[11] Kitchenham B. A., Pleeger S. L., y Fenton N. "Towards a Framework for Software Measurement Validation". IEEE Transactions on Software Engineering, Vol. 21, No. 12, pp 929-944, 1995.
[12] Narasimhan L., Bayu H., Theoretical Considerations for Software Componet Metrics, ISSN 1307-6884, 2005.
[13] Sant'Anna C., Garcia A., Chavez C., Lucena C., Staa A. "On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework", 2003
[14] Tarr, P. et al. "N Degrees of Separation: Multi -Dimensional Separation of Concerns". Proceedings of the 21st International Conference on Software Engineering, 1999.
[15] Sommerville, I. "Software Engineering", 6.ed. Harlow, England, Addison -Wesley, 2001.
[16] Fenton, N., Pfleeger, S. "Software Metrics: A Rigorous and Practical Approach". 2.ed. London: PWS, 1997.
[17] Quintero A.; Visión General de la Programación Orientada a Aspectos. Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla.
[18] Negro P.; Umbrales para Métricas Orientadas a Objetos; Universidad Abierta Interamericama; 2008.
[19] Berard E., "Metrics for Object-Oriented Software Engineering" comp-software-eng; 1995.
[20] Meyer B. "Object Oriented Software Construction"; 1988.
[21] Castro P.; "Ingeniería del Software Orientada a Objetos"; Revista del Instituto Tecnológico de Informática; http://www.iti.es/media/about/docs/tic/05/2004-10-ISOO.pdf
[22] Bertoa M., Vallecito A.; "Atributos de Calidad para Componentes de Software"; Departamento de Lenguajes y Ciencias de la Computación; Universidad de Málaga; 2002.
[23] Szyperski, C.; "Component Software: Beyond Object-Oriented Programming"; Addison-Wesley, New York, NY; 1997.
[24] Ning J.Q.; "Component-Based Software Development Model"; In Proceedings of the Annual International Computer Software and Applications Conference (COMPSAC'96), pp.389–394, IEEE; 1996.
[25] Carvalo J.P., Franch X., Quer C.; "Calidad de Componentes de Software" Capítulo 10 - Versión Preliminar; 2011.
[26] Calderón, A., Dámaris, S., Valverde Rebaza, J.C.: Metodologías Ágiles., 1—37; 2007
[27] Canós, J., Letelier, P.., Penadés, M.C..: Métodologías Ágiles en el DS, pp. 1—8, Valencia
[28] Sitio web de la Alianza de Metodologías Agiles http://agilealliance.org/; Mayo 2013

[29] http://www.intergrupo.com/blog/mobile/desarrollo-agil-software-mantenimiento.aspx; Junio 2013
[30] Proyecto previo a la obtención del título de Ingeniero en Sistemas Informáticos y de Computación, Lilian Elizabeth Arroba Medina, 2011
[32] http://www.slideshare.net/suncustomeruniversity/la-practica-del-aseguramiento-de-calidad-en-mtodos-giles-presentation#btnLast; Mayo 2013
[33] http://www.gestiondeproyectosit.es/blogit/2012/12/metricas-agiles-i/; Junio 2013
[34] Rodríguez D., Harrison R., "Medición en la orientación a objetos", School of Computer Science, Cybernetics & Electronic Engineering University of Reading, UK, 2007
[35] Brito e Abreu F., Melo W. "Evaluating the impact of Object- Oriented Design on Software Quality". Proceedings of 3rd International Software Metrics Symp., Berlin, 1996
[36] Chidamber S. R., Kemerer C. F. "A metric suite forsz object oriented design", IEEE Transactions on Software Engineering, pp. 467–493, 1994
[37] Lorenz M., Kidd J. Object Oriented Metrics. Englewood, NJ: Prentice Hall, 1994
[38] Kitchenham B., Guidelines for performing Systematic Literature Reviews in Software Engineering, p. 1-57 , 2007
[39] Mendes E., A Systematic Review of Web Engineering Research, IEEE 0-7803-9508 5/05, p.499-505, 2005
[40] Dubey S., Sharma A., Rana A. Comparison Study and Review on Object-Oriented Metrics, Global Journal of Computer Science and Technology, V. 12, Issue 7, 2012
[41] Meenakshi, Nasib G., Sunil S., Survey of Object-Oriented Metrics: Focusing on Validation and Formal Specification, V.37, Numbers 6, 2012
[42] Srivisut K., Muenchaisri P., Bad-Smell Metrics for Aspect-Oriented Software, 6th IEEE/ACIS, 2007
[43] Hartmann D., Dymond R., Appropriate Agile Measurement: Using Metrics and Diagnostics Business Value, IEEE Computer Society Press, 2006
E. Weyuker "Evaluating Software Complexity Measures", IEEE Transactions on Software Engineering, vol. 14, no. 9, 1988
[44] Abdellatief M., Md Sultan A., Component-based Software System Dependency Metrics based on Component Information Flow Measurements, ICSEA 2011: The Sixth International Conference on Software Engineering Advances, 2011
[45] Chen J., YEAP W., Bruda S., A Review of Component Cuopling Metrics for Component-Based Development, World Congress on Software Engineering, IEEE, 2009.