UNIVERSIDAD DE PAMPLONA
FACULTAD DE INGENIERÍAS Y ARQUITECTURA
PROGRAMA DE INGENIERIA DE SISTEMAS

TRABAJO DE GRADO PRESENTADO PARA OPTAR AL TÍTULO DE INGENIERO DE SISTEMAS

TEMA:

ESTUDIO FUNCIONAL DE LA PLATAFORMA “SONARQUBE” PARA LA EVALUACIÓN DE CÓDIGO FUENTE CON RESPECTO A LA CALIDAD DEL PRODUCTO SOFTWARE.

AUTOR:

HAROL ANDREY VERA CELIS

PAMPLONA, NORTE DE SANTANDER
JULIO 2019

UNIVERSIDAD DE PAMPLONA
FACULTAD DE INGENIERÍAS Y ARQUITECTURA
PROGRAMA DE INGENIERIA DE SISTEMAS

TRABAJO DE GRADO PRESENTADO PARA OPTAR AL TÍTULO DE INGENIERO DE SISTEMAS

TEMA:

ESTUDIO FUNCIONAL DE LA PLATAFORMA “SONARQUBE” PARA LA EVALUACIÓN DE CÓDIGO FUENTE CON RESPECTO A LA CALIDAD DEL PRODUCTO SOFTWARE.

AUTOR:

HAROL ANDREY VERA CELIS

DIRECTOR:

EDGAR ALEXIS ALBORNOZ ESPINEL
Magister en ciencias computacionales

PAMPLONA, NORTE DE SANTANDER
JULIO 2019

CITAS

“La calidad nunca es un accidente, siempre es resultado de un esfuerzo de la inteligencia”

John Ruskiin

DEDICATORIA

Dedico este trabajo en primera instancia a DIOS por iluminarme y guiarme para poder cumplir mi meta.

A mis padres, por ser mi ejemplo a seguir, ayudándome en todo momento y ser mi pilar de apoyo durante este recorrido académico.

A mis amigos, compañeros y colegas con los cuales compartí gratos momentos ayudando a mi formación personal, académico y profesional. I.O.

Por ultimo pero no menos importante agradecer de todo corazón a todos aquellos docentes que con su experiencia y sabiduría me formaron como un excelente profesional.

RESUMEN

La siguiente investigación tiene como fin el estudio funcional de una plataforma de análisis estadístico para la gestión de la calidad del código fuente del software, llamada “SONARQUBE” la cual es de licencia libre y gratuito, este estudio se realizó para determinar la relación que tiene la plataforma con los modelos de calidad de producto software en la actualidad, los cuales son: ISO 9126, 14598 y 25000, de esta manera saber que tan acertadas están sus estadísticas a las ISOS mencionadas anteriormente, por otro lado se hiso uso de esta plataforma examinando un proyecto de software libre, para estudiar el funcionamiento y manejo de SONARQUBE, analizando sus resultados y concluyendo que tan aceptable es la eficacia y veracidad de la información generada y sus estadísticas.

Teniendo en cuenta la historia y diversas versiones con las que cuenta la plataforma, se da a conocer las características principales para la selección de la versión 6.4 la cual fue un punto primordial para desarrollar el contenido de este trabajo, así mismo, se muestra diferentes formas de aplicar e implementar el scanner de SONARQUBE a un proyecto de lenguaje java, partiendo de uno básico, seguido de uno en el IDE NetBeans vinculado a partir de un plugins y finalizando con el software de Jenkins incluyendo la cobertura de pruebas unitarias en dicho escaneo.

Al concluir los escaneos se realiza una interpretación de los resultados analizando las estadísticas obtenidas haciendo énfasis en corrección de errores, cobertura y porcentajes de aceptación de calidad, ultimando con un cuadro comparativo entre los objetivos de SONARQUBE y las normativas de calidad de producto software.

ABSTRACT

The following research has as purpose the functional study of a platform of statistical analysis for the management of the quality of the source code of the software, called "SONARQUBE" which has free license and it's free, this study was carried out to determine the relation that the platform has with software product quality models presented nowadays, which are: ISO 9126, 14598 and 25000, in this way to know how accurate their statistics are to the ISOS mentioned above, On the other hand, a use of this platform was made examining a free software project, to study the operation and management of SONARQUBE, analyzing its results and concluding how acceptable the effectiveness and veracity of the information generated and its statistics are.

Taking into account the history and various versions of the platform, the main features for the selection of version 6.4, which was a key point to develop the content of this work are shown, as well as different forms to apply and implement the SONARQUBE scanner to a java language project, starting from a basic one, followed by one in the linked NetBeans IDE from a plugin and ending with the Jenkins software including the coverage of unit tests in that scan.

At the end of the scans, an interpretation based on the statistics obtained is shown, with an emphasis on error correction, coverage and percentages of quality acceptance, with a final comparison between the objectives of SONARQUBE and the software product quality regulations.

TABLA DE CONTENIDO

INDICE DE IMÁGENES

INDICE DE TABLAS

1. INTRODUCCIÓN

En el entorno de ingeniería del software ha existido durante décadas cierta inseguridad a la hora de entregar proyectos software, está radica en la aceptación del proyecto en términos de calidad, por parte de los roles directamente implicados como desarrolladores, compradores y usuarios finales, lo cual determina el éxito del mismo, y la reputación de las partes involucradas.

Surge el interrogante, ¿a qué se debe esta preocupación?, La principal causa es la falta de conocimiento acerca de las normativas, plataformas digitales y métodos de preparación, ejecución y análisis de resultados de pruebas orientadas a la calidad del producto software, (universidad de los andes, 2015).

Dando una posible solución a esta problemática, se desarrolló esta investigación basada principalmente en el estudio, interpretación y documentación de normativas internacionales establecidas para la calidad del producto software, entre ellas están: ISO 9126,14598, 25000, junto a exámenes de aplicativos en la plataforma de análisis estadístico de código fuente llamado SonarQube, esto con el fin de realizar un estudio funcional de dicha plataforma con relación a la calidad del producto software.

La plataforma mencionada con anterioridad, sirvió para establecer criterios de calidad de producto software a través de sus reglas de aplicación, fundamentada principalmente en la característica de mantenibilidad de la ISO 25000

En el desarrollo de este trabajo se muestra como da a conocer SonarQube su visión de calidad, procedimientos de evaluación utilizados y la forma adecuada de vincularse con proyectos de software de lenguaje java mediante diferentes herramientas digitales, mostrando de una manera simple pero detallada el uso e implementación de pruebas unitarias, scanner de código fuente, y análisis de métricas y solución de errores de programación.

Concluyendo con el análisis de resultados de los aplicativos examinados por SonarQube y un comparativo entre la plataforma y los objetivos de las normativas estudiadas, determinando de esta manera que tan acertada son las métricas de la plataforma en cuanto a la ISO de calidad de producto software.

1.1 Planteamiento del problema

Las empresas de desarrollo software han presentado un bajo rendimiento relacionado con la calidad del producto software, tal y como lo da a conocer The CHAOS Manifesto, The Standish Group (2013, p.5), donde muestra que solo un 39% de la resolución de proyectos tipo software terminan con total éxito, un 43% tiene algún tipo de dificultad entregando tarde, por encima de presupuesto y/o con menos de las características y funciones solicitadas, y en los peores casos un 18% es cancelado antes de ser terminado o entregado y nunca usado.

En este orden la problemática principal se basa en la poca información que cuentan las empresas acerca de modelos de calidad, implementación y análisis de dichos modelos, y/o software relacionados que realicen estos procesos de acuerdo a las normativas que implica la calidad del producto software, sin necesidad de incluir incrementos monetarios en los proyectos, procesos que entorpezcan las entregas del producto y a su vez sean confiables.

1.2 Justificación

Este proyecto se desarrolló con el fin de determinar la estrecha relación que tiene la plataforma SonarQube en su análisis y procesos de evaluación de calidad, con las normativas acordes a la calidad de producto software, y como, a partir de éste análisis se puede cambiar la concepción de “calidad” de las empresas de productos software, fomentando su posible implementación.

Mediante la información obtenida se analiza la plataforma SonarQube y la relación con las normativas observadas, para dar una perspectiva acerca del porque el déficit en la calidad de proyectos informáticos en cuanto a calidad de producto software, lo anterior se encuentra plasmado en un paralelo dando a conocer las características de las ISO implicadas en la plataforma.

La implementación de la plataforma SonarQube se realiza para poder mostrar una visión diferente de la calidad basado en herramientas tecnológicas, mostrando la facilidad del proceso de evaluación, estudio de métricas y corrección de errores.

1.3 Delimitación

Objetivo General

Realizar un estudio funcional de la plataforma “SonarQube” para la evaluación de código fuente con respecto a la calidad del producto software.

Objetivos Específicos

- Efectuar un estudio de los modelos de calidad de producto software.
- Analizar la plataforma “SonarQube” con enfoque en las normas y modelos de calidad de producto software.
- Aplicar a un proyecto de software libre el análisis de código fuente con la plataforma “SonarQube”
- Realizar un paralelo entre los objetivos de los modelos de calidad de producto software y los resultados obtenidos por la plataforma “SonarQube”.

Acotaciones

- Este estudio se enfoca netamente a la parte funcional el software “SonarQube”.
- Las normativas a utilizar son las ISO 9126, ISO 25000.

2. CONCEPTOS PRELIMINARES

La Calidad es un área de estudio muy grande que abarca diferentes conceptos y focalizando la calidad del software se especializa aún más, se abordaran los conceptos esenciales, como lo son definiciones, normativas y herramientas enfocadas a calidad de producto software.

2.1. Calidad

Enfocando a una opinión referente a software y tomando la ISO/IEC/IEEE 24765 segunda edición (2017, p.364) define calidad como el “grado en que el sistema satisface las necesidades establecidas e implícitas de sus diversas partes interesadas, y por lo tanto proporciona valor” Por otro lado la Organización Internacional de Normalización ISO/ IEC 25010(2011a) plasma la calidad de software como la “capacidad de un producto, servicio, sistema, componente, o proceso para satisfacer las necesidades, expectativas o requisitos del cliente o del usuario”.

Tomando las anteriores definiciones de calidad se puede intuir que se trata de un conjunto de cualidades que otorgan un valor a un producto o servicio denotando superioridad respecto a los de su misma categoría, dicha categoría es desarrollo de producto software.

2.1.1. Calidad de producto software.

Las entidades encargadas del desarrollo de productos software en los últimos años han implementado diversos métodos y técnicas para mejorar sus productos, contemplando el crecimiento de la competencia, y analizando la importancia de ofertar software con estándares y modelos de calidad.

Se necesita definir un conjunto de actividades cuidadosamente planificadas que les permita a las empresas vigilar los productos a lo largo de todas las etapas del ciclo de vida de desarrollo del software, para asegurar la calidad del producto final. (Orantes Jiménez, 2007, p.8)

Para realizar estas evaluaciones del producto software existen diferentes modelos los cuales determinan la calidad usando atributos puntuales en diferentes etapas de éste, entre las más importantes se encuentran:

2.1.1.1 El modelo de McCall.

Este modelo fue presentado en el 1977 por Richards y Walters, contando con el patrocinio de Air Forcé y Dod, se basa en el punto de vista del cliente y el producto final analizando atributos fijados por el cliente, dichos atributos se consideran factores de calidad ya sean internos o externos.

Debido a la inmensidad de estos factores de calidad, se tiende a dificultar su medición, por eso, se toma en consideración atributos de un nivel más bajo llamados criterios de calidad.

Este modelo distribuye sus factores en tres grandes ejes visibles para los clientes, tomados de once factores organizados alrededor de los ejes, que a su vez despliegan los criterios de calidad.

- **Ejes**
  - ✓ Operación del producto.
  - ✓ Transición del producto.
  - ✓ Revisión del producto.

- **Factores**
  - ✓ Facilidad de uso.
  - ✓ Integridad.
  - ✓ Corrección.
  - ✓ Fiabilidad.
  - ✓ Eficiencia.
  - ✓ Facilidad de mantenimiento.
  - ✓ Facilidad de prueba.
  - ✓ Flexibilidad.
  - ✓ Interoperabilidad.
  - ✓ Portabilidad.

Cada uno de los anteriores factores cuenta con diferentes criterios basados en la diversidad del proyecto.

- **Operación del producto.**
  - Corrección: ¿El software hace lo que necesito?
  - Fiabilidad: ¿Lo Hace de forma exacta siempre?
  - Eficiencia: ¿Aprovecha el hardware lo mejor posible?
  - Integridad: ¿Es seguro?
  - Facilidad de Uso: ¿Lo puedo usar con facilidad?

- **Revisión del producto**
  - Facilidad de prueba: ¿Se puede probar?
  - Flexibilidad: ¿Se puede modificar?
  - Facilidad de mantenimiento: ¿Que tan difícil es arreglarlo?
- **Transición del producto**
  - Portabilidad: ¿Se puede usar en otra máquina?
  - Reusabilidad: ¿Se puede reutilizar parte del Software?
  - Interoperabilidad: ¿Se puede comunicar con otros sistemas? (Pilalonga, 2017).

2.1.1.2 El modelo de Bohem

Propuesto por Barry Bohem en 1978, para él el software debe seguir estrictamente lo deseado por el cliente, hace énfasis en tres características que se espera de un software de calidad.

- Usar los recursos de una manera adecuada y eficiente.
- Fácil de usar y aprender para los usuario y clientes.
- Encontrarse bien diseñado, codificado, ser probado y fácil mantenibilidad.

Su estructura está basada en 3 niveles generados por sus características, de alto nivel, nivel intermedio y primitivo.

a. Características de alto nivel

Estas características representan requerimientos generales de uso:

- Utilidad, cuan (usable, confiable, eficiente) es el producto en sí mismo.
- Mantenimiento, cuan fácil es modificarlo, entenderlo y retestearlo.
- Utilidad general, si puede seguir usándose si se cambia el ambiente.

b. Características de nivel intermedio

Estas características representan los factores de calidad de Boehm:

- Portabilidad
- Fiabilidad
- Eficiencia
- Usabilidad
- Capacidad de prueba
- Flexibilidad

c. Características Primitivas

Este es el nivel más bajo y corresponde a características directamente asociadas a una o dos métricas de calidad:

- Portabilidad
- Eficiencia
- Usabilidad
- Testeabilidad
- Entendibilidad
- Modificabilidad (Pilalonga, 2017).

2.1.1.3 Modelo FURPS+

Este modelo fue desarrollado por Hewlett-Packard en el año 1987 dentro de su estructura establece cinco características como factores de calidad que son los que le dan nombre:

- **F**unctionality (Funcionalidad).
- **U**sability (Usabilidad).
- **R**eliability (Confiabilidad).
- **P**erfomance (Prestación) y
- **S**upportability (Soporte).

El modelo FURPS incluye, además de los factores de calidad y los atributos, restricciones de diseño y requerimientos de implementación, físicos y de interfaz. Una limitación de este modelo de calidad es que no tiene en cuenta la portabilidad de los

productos software que se estén atendiendo, factor digno de consideración en función de las exigencias actuales que recaen sobre el proceso de desarrollo del software. (Ingeniería del software FURPS, 2008).

Los modelos definidos anteriormente sirvieron como preámbulo para el desarrollo de la normativa ISO 9126, de aquí su importancia al ser conceptualizadas.

2.2 Norma de evaluación ISO/IEC 9126 capítulos 1- 4.

La ISO, bajo la norma ISO-9126, ha establecido un estándar internacional para la evaluación de la calidad de productos de software el cual fue publicado en 1992 con el nombre de “Information technology –Software product evaluation: Quality characteristics and guidelines for their use”, en el cual se establecen las características de calidad para productos de software. (Cataldi, 2000, P.1).

Esta normativa plantea diversos términos, los cuales considera apropiados y de importancia para el estudio del contenido de la ISO, los conceptos más relevantes son:

- **Medida**: Proporciona una indicación cuantitativa de la cantidad, dimensiones o tamaño de algunos atributos de un producto.
- **Medición**: Acto de determinar una medida.
- **Métrica**: Es una medida del grado en que un sistema, componente proceso posee un atributo dado.(slideshare,2014)

Dicho estándar se divide en cuatro partes, 9126-1 modelos de calidad, 9126-2 métricas externas, 9126-3 métricas internas, 9126-4 métricas en uso, las divisiones nombradas y sus características son determinantes para medir el software dependiendo de la etapa en la que se encuentre.

2.2.1 ISO / IEC 9126-1 Modelo de calidad.

La norma en su primera parte, realiza una clasificación en un conjunto de características y subcaracterísticas, consideradas de vital importancia para el buen uso del estándar, planteada como la muestra la imagen 1.

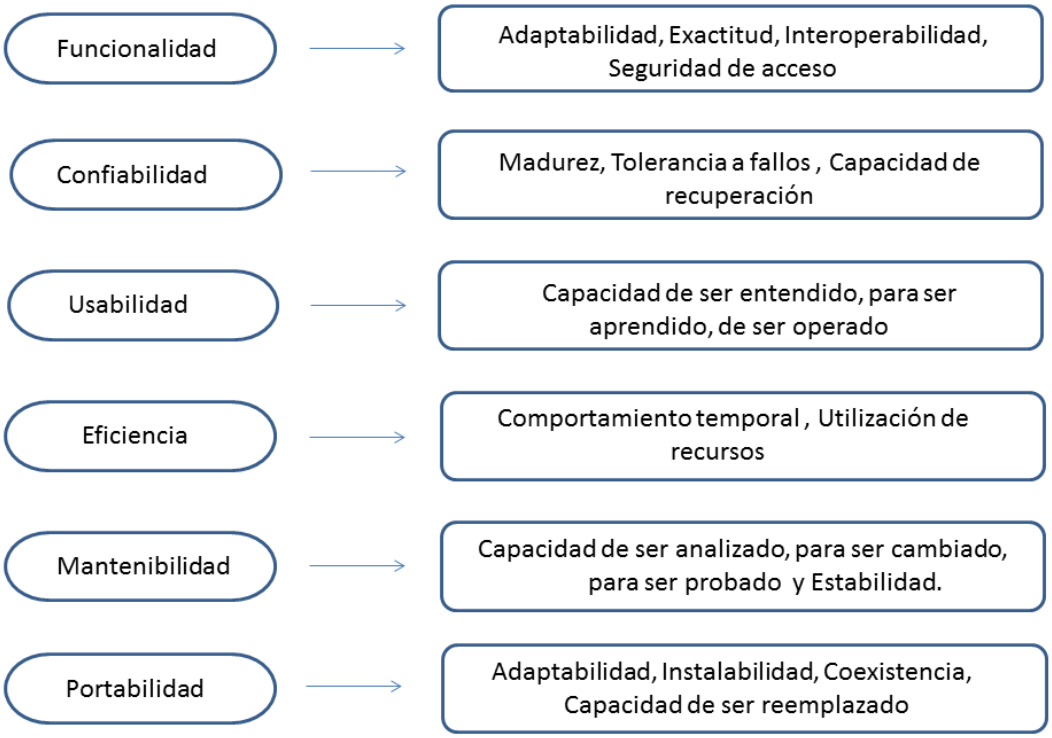


Imagen 1 división modelo de calidad ISO 9126

Teniendo clara la anterior clasificación, la ISO define cada una de sus características y subcaracterísticas, sabiendo que la ISO 9126 fue la antecesora de la ISO 25000 se conceptualiza de sus características, dejando la contextualización de sus subcaracterísticas a la normativa más reciente la ISO 25000.

- **Funcionalidad**

Se basa en un sistema de cualidades que representan la existencia de un conjunto de funciones y sus respectivas características especificadas, donde dichas funciones compensan las necesidades indicadas o involucradas.

- **Fiabilidad**

Propone cualidades para referirse a la capacidad del software para conservar sus niveles de funcionamiento derivadas de condiciones de especificas por un periodo determinado.

- **Usabilidad**

Consiste en un conjunto de atributos o cualidades fundamentales para evaluar el esfuerzo del usuario a la hora de entender y utilizar el sistema o software

- **Eficiencia**

  Esta característica plasma una evaluación de relación entre en nivel de funcionamiento del software y la cantidad de recursos de computo consumidos siguiendo ciertas condiciones indicadas

- **Mantenibilidad**

  Se refiere a las propiedades del sistema que permite calcular el esfuerzo necesario para realizar modificaciones específicas, como correcciones de errores o bugs, por el incremento o cambios de funcionalidades.

- **Portabilidad**

Trata de la habilidad del software para posibles traspasos de un entorno a otro, sin necesidad de modificaciones. (Borbón Ardila, 2013).

2.2.2 ISO / IEC 9126-2 Métrica Externa

Esta sección de la normativa se basa en la medición del comportamiento de los sistemas basados en las computadoras que incluyen al software, se relaciona dicho comportamiento cuando se encuentra en ejecución.
Sus principales características radican en proporcionar métricas externas para medir cada una de las características plasmadas en la ISO 9126-1, también proporciona una guía para usuarios basados en las métricas para planificarlas, seleccionarlas, aplicarlas e interpretar los resultados de éstas, y por último, no asigna rango de valores para sus métricas en niveles o en grados de conformidad, ya que estos valores se definen para cada producto o en partes especificas del mismo. (Camargo, 2014, p.3)

2.2.3 ISO / IEC 9126-3 Métrica Interna

Su principal objetivo es tratar de asegurar que se logren los requisitos de la calidad externa y calidad en uso.
Mide al software y sus características por sí mismo es decir que es medible a partir de las características intrínsecas, como el código fuente o sus funciones. Este proporciona las métricas internas para medir los atributos de las seis características (Funcionalidad, Confiabilidad, Usabilidad, Eficiencia, Capacidad de mantenimiento, Portabilidad),
Los desarrolladores, evaluadores, gestores de calidad, personal de mantenimiento, proveedores, usuarios y compradores pueden seleccionar parámetros de la norma para los requisitos de la definición.
Estas métricas se aplican a productos en su fase no ejecutable y durante las etapas de desarrollo además permite medir la calidad de entregables requeridos y así predecir la calidad del producto final.

Por otro lado permite al usuario iniciar acciones correctivas temprano en el ciclo de desarrollo evitando en la mayoría de los casos sobrecostos y tiempo extra de trabajo. (Camargo, 2014, p.3)

2.2.4 ISO / IEC 9126-4 Métrica calidad en uso

Miden los efectos de utilizar el software en un contexto de uso específico, proporciona las métricas de calidad en uso para medir los atributos (Eficacia, Productividad, Satisfacción, Seguridad).
Se basa en la perspectiva de calidad del usuario y la utilización efectiva del software por parte de él, donde la capacidad del software le permita a éste lograr las metas propuestas en contextos específicos. (Camargo, 2014, p.4)

Atributos calidad en uso.

- **Eficacia**
  Permite al usuario alcanzar los objetivos especificados del software con exactitud y completitud, en un contexto de uso especificado.

- **Productividad**
  Permite al usuario emplear cantidades apropiadas de recursos con relación a la efectividad alcanzada.

- **Satisfacción**
  Es la capacidad del producto de alcanzar niveles aceptables del riesgo de hacer daño a personas, al negocio, al software, a las propiedades o al medio.

  En otras palabras denota todo aquello que puede ser perjudicial para el desarrollo del proyecto y sus participantes.

- **Seguridad**

Satisfacer a los usuarios en un contexto de uso especificado tomando como referencia la importancia de custodiar la información y el exceso a está, es la respuesta del usuario a la interacción con el producto. (Largo y Marín, SF, p.25)

La anterior normativa proporciona diferentes planteamientos del porque la importancia para medir un software.

- Indicar la calidad del producto.
- Evaluar la productividad del agente que desarrolla el producto.
- Evaluar los beneficios en términos de productividad y calidad mediante el uso de nuevos métodos y herramientas de ingeniería de software.
- Establecer una línea de base para la estimación.
- Ayudar a justificar el uso de nuevas herramientas o de formación adicional. (Vegas,2014).

2.3 Evaluación de productos de software ISO / IEC 14598

Es necesario considerar mediciones en el proceso empleado para diseñar, desarrollar, probar y controlar el producto. En esto juega un papel relevante dicha normativa, la cual ofrece una visión general, explicando la relación entre su serie (división) y el modelo de calidad de ISO 9126.

Además define los términos técnicos utilizados, contiene requisitos generales para la especificación y evaluación de la calidad del software, y clarifica los conceptos generales.

También provee un marco de trabajo para evaluar la calidad de todos los tipos de productos de software y establece requisitos para métodos de medición y evaluación de los productos de software.

El proceso de evaluación se especifica en tres situaciones diferentes requisitos para desarrolladores, requisitos para compradores y requisitos para evaluadores.

La ISO 14598 está prevista para que se use conjuntamente con la ISO 9126-1. El propósito de la evaluación de la calidad del software es hacer que tanto el desarrollo y la adquisición del software cumplan las expectativas y necesidades del usuario. Está norma define el proceso de evaluación y provee los requerimientos y las guías que conducen a evaluaciones de calidad (slideshare, 2014).

En sus diferentes etapas, establece un tablero de trabajo para evaluar la calidad de los productos de software proporcionando, además, métricas y requisitos para los procesos de evaluación de los mismos.

En particular, es utilizada para aplicar los conceptos descritos en la norma ISO / IEC 9126.

Se definen y describen las actividades necesarias para analizar los requisitos de evaluación, para especificar, diseñar y realizar acciones de evaluación y para concluir la evaluación de cualquier tipo de producto de software.

2.3.1 Características

La norma define las principales características del proceso de evaluación

- **Repetitividad.** La repetitividad se refiere a la posibilidad de obtener resultados consistentes al replicar un estudio con un conjunto distinto de datos, pero obtenidos siguiendo el mismo diseño experimental.
- **Reproducibilidad** se refiere a la capacidad que tenga una prueba o experimento de ser reproducido o replicado por otros, en particular, por la comunidad científica. (Wikipedia, 2018)
- **Imparcialidad.** Pretende garantizar que la prueba estandarizada sea justa, sin prejuicios ni sesgos.
- **Objetividad.** La objetividad es un atributo necesario que debe detallarse claramente para satisfacer los propósitos científicos de todo proyecto de evaluación. (López y Pedraza, 2017, p.22)

Para estas características se describen las medidas concretas que participan:

Análisis de los requisitos de evaluación.
Evaluación de las especificaciones.
Evaluación del diseño y definición del plan de evaluación.
Ejecución del plan de evaluación.
Evaluación de la conclusión. (Padra, 2013)

La Norma ISO/IEC 14598 define el proceso para evaluar un producto de software, el mismo consta de seis partes:

- **ISO/IEC 14598-1 Visión General**: provee una visión general de las otras cinco partes y explica la relación entre la evaluación del producto software y el modelo de calidad definido en la ISO/IEC 9126, está plantea requisitos de evaluación, además el cómo especificar, diseñar y ejecutar la evaluación como lo muestra la imagen 2.

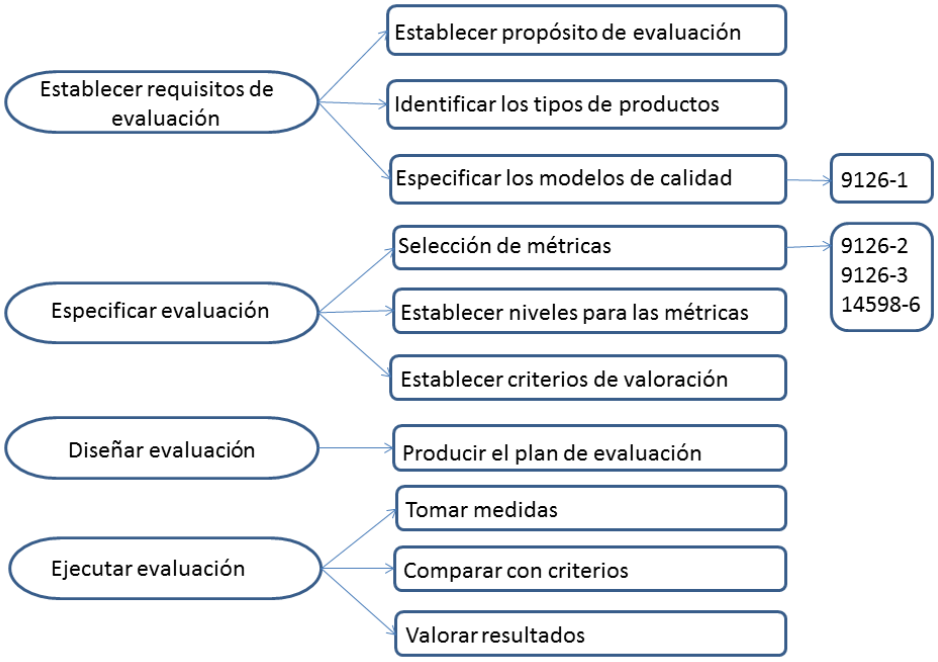


Imagen 2 Visión general ISO 14598-1.

El proceso de especificación del modelo de calidad va conjunto a la ISO 9126-1 Características de calidad.
El proceso de selección de métricas va conjunto a la ISO 9126-2 Métricas externas, ISO 9126-3 Métricas internas, ISO 14598-6 Módulos de evaluación.
En la imagen 2, se muestra los ejes principales de características y subcaracterísticas enfocados a la correlación con la ISO 9126 y en qué parte se usan exactamente.

Los resultados de aplicar está normativa los pueden utilizar 3 sectores importantes, el primer sector es el de **gerentes y desarrolladores**, de esta manera poder medir el cumplimiento de los requisitos y posibilidad de realizar mejoras en caso de ser necesario.
Por otro lado está el sector de **analistas** quien establece la relación entre métricas de la ISO 9126 (métrica externa e interna) y el último de estos es el sector del **personal de mejoras** quienes son los encargados de dar a conocer posibles mejoras a través de la información suministrada por el estudio de calidad del producto software.

- **ISO/IEC 14598-2 Planeamiento y Gestión**: contiene requisitos y guías para las funciones de soporte tales como la planificación y gestión de la evaluación del producto del software.

  Lo fundamental de este componente es Planificar mediciones y actividades esto incluyen la preparación de las políticas, definición de objetivos organizaciones, identificación de la tecnología, identificación y aplicación de técnicas de evaluación para software desarrollado y adquirido, asignación de responsabilidades entre otras.

- **ISO/IEC 14598-3 Proceso para desarrolladores:** provee los requisitos y guías para la evaluación del producto software cuando la evaluación es llevada a cabo en paralelo con el desarrollo por parte del desarrollador.
  Se enfoca en el uso de indicadores los cuales predicen la calidad final del producto midiendo los productos intermedio que se desarrollan en el ciclo de vida.
- **ISO/IEC 14598-4 Proceso para adquirientes**: provee los requisitos y guías para que la evaluación del producto software sea llevada a cabo en función a los compradores que planean adquirir o reutilizar un producto de software existente o predesarrollado.
- **ISO/IEC 14598-5 Proceso para evaluadores**: provee los requisitos y guías para la evaluación del producto software cuando la evaluación es llevada a cabo por evaluadores independientes.
  Su rol es importante generar los requerimientos, especificaciones diseño, actividades y reporte de la evaluación del producto.
- **ISO/IEC 14598-6 Documentación de Módulos**: provee las guías para la documentación del módulo de evaluación, donde se especifica el modelo de calidad y métricas para la evaluación con sus respectivos métodos.(EcuRED,2012)

2.4 ISO 25000 SQUARE (Requisitos y Evaluación de Calidad de Productos de Software)

Esta normativa surgió como mejora a su antecesor la norma ISO 9126, con refuerzo de términos considerados como confusos a la hora de su interpretación, e inclusión de algunos otros conceptos, donde se reasignaron algunas subcaracterísticas y se redefinieron otras.

SQuaRE establece criterios para especificar requisitos de calidad de productos software, junto con sus métricas y su evaluación, aplica unos modelos de calidad para unificar las definiciones de calidad de los clientes y los procesos de desarrollo.

Al igual que la ISO 9126 presenta algunos términos previos para aumentar la ilustración de la norma.

- **Atributo**: Propiedad inherente de una entidad que puede distinguirse cuantitativa o cualitativamente ya sea manual o automáticamente.
- **Calidad interna**: Capacidad de un conjunto estático de atributos para satisfacer las necesidades declaradas e implícitas de un producto software bajo ciertas condiciones especificadas.
- **Calidad externa**: Capacidad de un producto software para desarrollar el comportamiento de un sistema de forma que satisfaga las necesidades declaradas e implícitas de un sistema utilizado bajo ciertas condiciones especificadas.
- **Calidad en uso**: Grado en que un producto satisface objetivos con efectividad, seguridad, satisfacción y productividad.
- **(Medida) primitiva**: Medida, tanto base como derivada, utilizada para medir la calidad del software.
- **Medida base**: Conjunto formado por la medida definida en términos de un atributo más el método para su cuantificación.
- **Medida derivada**: Medida obtenida a partir dos o más medidas base.
- **Módulo de evaluación**: Módulo tecnológico para la medida de características, subcaracterísticas y atributos de evaluación, incluyendo métodos y técnicas de evaluación, entradas a procesar, datos a recoger y medir, y herramientas y procedimientos de apoyo.
- **Validación**: Confirmación por medio de pruebas objetivas de que se satisfacen los requisitos para un uso específico o para una aplicación.
- **Verificación:** Confirmación por medio de pruebas objetivas de que se satisfacen los requisitos especificados.(Marulanda, 2014)

Una de las características preservadas de la ISO 9126 es sus divisiones de aplicación según la etapa del ciclo de vida en la que se encuentre, está familia de normas posee la siguiente división:

ISO/IEC 2500n: División de la gestión de calidad

Los estándares que forman esta división definen todos los modelos comunes, términos, y referencias a los que se alude en las demás divisiones de SQuaRE, está se encentra conformada por:

- ISO/IEC 25000-Guia para SQuaRE: Contiene el modelo de la arquitectura, la terminología, un resumen de las partes, los usuarios previstos y las partes asociadas, así como los modelos de referencia.
- ISO/IEC 25001- Planificación y Gestión: establece los requisitos y orientaciones para gestionar la evaluación y especificación de los requisitos del producto software.

- **ISO/IEC 2501n: División del modelo de calidad**

  Presenta un modelo de calidad detallado, el cual incluye características para la calidad interna, externa y en uso del producto, formada por:

  - ISO/IEC 25010 - Modelos de calidad de sistema y software: describe el modelo de calidad para el producto software y para la calidad en uso. Esta Norma presenta las características y subcaracterísticas de calidad frente a las cuales evaluar el producto software.
  - ISO/IEC 25012 - Modelo de calidad de datos: define un modelo general para la calidad de los datos, aplicable a aquellos datos que se encuentran almacenados de manera estructurada y forman parte de un Sistema de Información.

- **ISO/IEC 2502n: División de las mediciones de calidad**

  Incluye un modelo de referencia de calidad del producto, definiciones, métricas y una guía práctica para su aplicación, muestra aplicaciones de métricas para la calidad del software interna, externa y en uso, posee la siguiente subdivisión:

  - ISO/IEC 25020 - Modelo de referencia de medición y guía: presenta una explicación introductoria y un modelo de referencia común a los elementos de medición de la calidad. También proporciona una guía para que los usuarios seleccionen o desarrollen y apliquen medidas propuestas por normas ISO.
  - ISO/IEC 25021 - Elementos de medida de calidad: define y especifica un conjunto recomendado de métricas base y derivadas que puedan ser usadas a lo largo de todo el ciclo de vida del desarrollo software.
  - ISO/IEC 25022 - Medición de la calidad en uso.: define específicamente las métricas para realizar la medición de la calidad en uso del producto.
  - ISO/IEC 25023 - Medición de la calidad del producto del sistema y software: define específicamente las métricas para realizar la medición de la calidad de productos y sistemas software.
  - ISO/IEC 25024 - Medición de la calidad de los datos: define específicamente las métricas para realizar la medición de la calidad de datos.

- **ISO/IEC 2503n: División de requisitos de calidad.**

  Ayuda a especificar requisitos que pueden ser usados en el proceso de especificación para la calidad de un producto software previo a ser desarrollado o como entrada para un proceso de evaluación, dentro de su división se encuentra:

  - ISO/IEC 25030 - Requerimientos de calidad: provee de un conjunto de recomendaciones para realizar la especificación de los requisitos de calidad del producto software.

- **ISO/IEC 2504n: División de evaluación de la calidad.**

  Muestra recomendaciones y guías para la evaluación de un producto software tanto si la llevan a cabo evaluadores, clientes o desarrolladores, se encuentra formada por:

  - ISO/IEC 25040 - Evaluación modelo de referencia y guía: propone un modelo de referencia general para la evaluación, que considera las entradas al proceso de evaluación, las restricciones y los recursos necesarios para obtener las correspondientes salidas.
  - ISO/IEC 25041 - Guía de evaluación para desarrolladores, adquirentes y evaluadores independientes: describe los requisitos y recomendaciones para la implementación práctica de la evaluación del producto software desde el punto de vista de los desarrolladores, de los adquirentes y de los evaluadores independientes.
  - ISO/IEC 25042 - Módulos de evaluación: define lo que la Norma considera un módulo de evaluación y la documentación, estructura y contenido que se debe utilizar a la hora de definir uno de estos módulos.
  - ISO/IEC 25045 - Módulo de evaluación de recuperabilidad.: define un módulo para la evaluación de la subcaracterística Recuperabilidad(INTERNATIONAL STANDARD,2011, p.1-2)

  Como énfasis del trabajo se tomó solo la sección 2501n “División de modelo de calidad”, contando con que es el único estándar de esta serie el “ISO/IEC 25010– “Modelos de calidad de sistema y software”, que detalla las características para la calidad interna, externa y en uso. Éste se hace descomponiendo cada tipo de calidad en características, aunque el nivel de descomposición es mayor para la calidad en uso que para la interna o externa.

2.4.1 Características y subcaracterísticas de modelo de calidad según ISO 2501n

- **Funcionalidad**

Capacidad del producto de software para proveer las funciones que satisfacen las necesidades explicitas e implícitas cuando el software se utiliza en condiciones específicas, sus subcaracterísticas y definiciones se encuentra en la imagen 3.

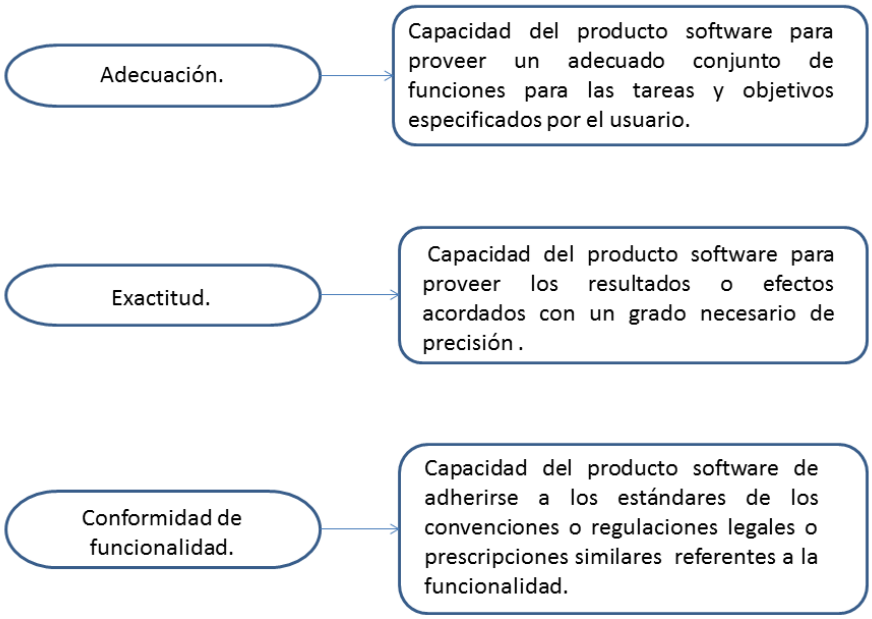


Imagen 3 Subcaracterísticas de funcionalidad

- **Seguridad**

  Capacidad del producto software para proteger la información y los datos de modo que las personas o los sistemas no autorizados no pueda leerlos o modificarlos y a las personas o sistemas autorizados no se les niegue el acceso a estos, sus subcaracterísticas y definiciones, se encuentra en la imagen 4.

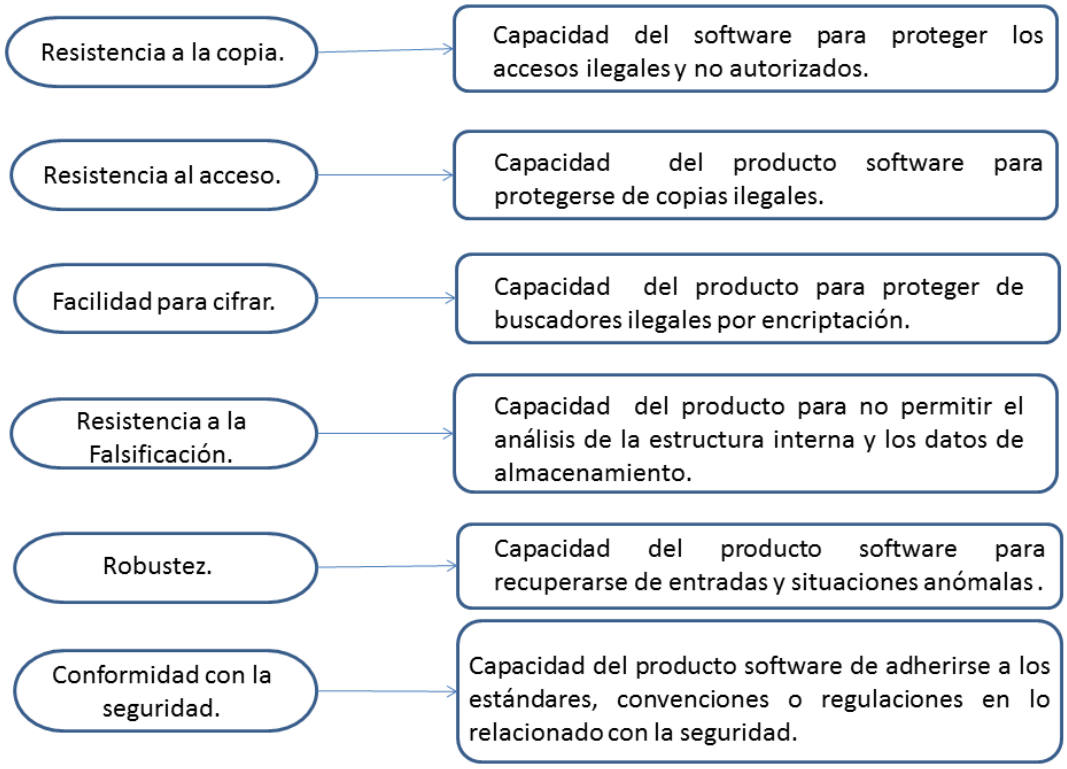


Imagen 4 Subcaracterísticas de seguridad

- **Interoperabilidad**

  Capacidad del producto software de interactuar con uno o más sistemas, se utiliza en lugar de compatibilidad para evitar una posible ambigüedad con la reemplazabilidad, sus subcaracterísticas y definiciones se encuentran en la imagen 5.

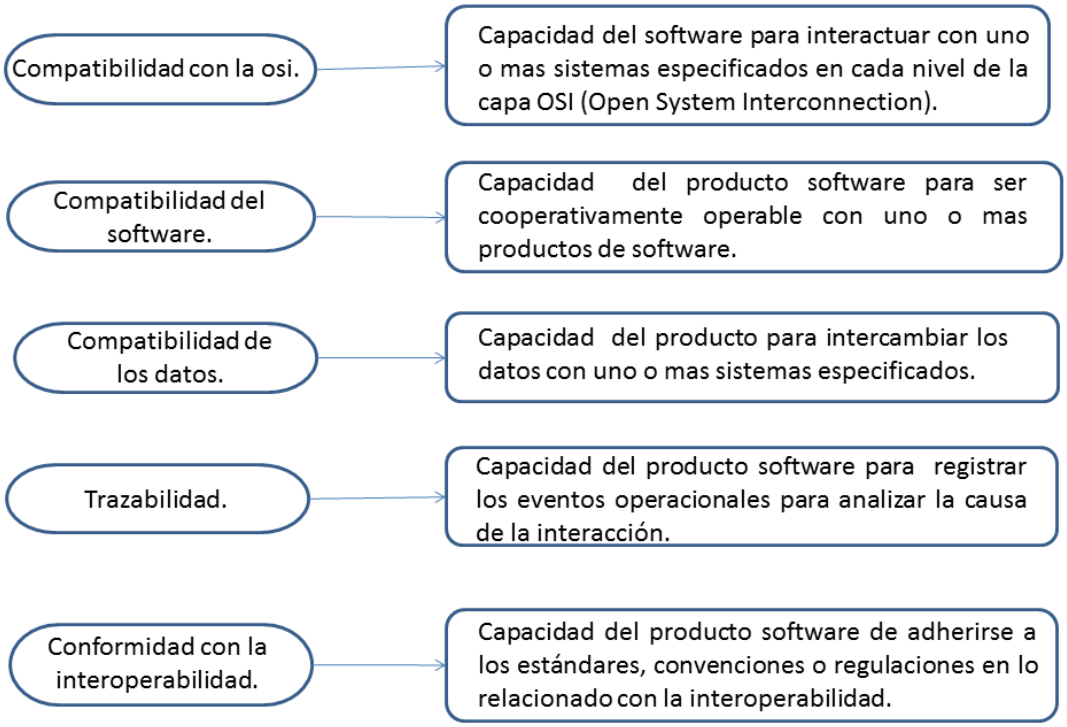


Imagen 5 Subcaracterísticas de interoperabilidad

- **Fiabilidad**

  Capacidad del producto software para mantener un nivel específico de funcionamiento cuando se está utilizando bajo condiciones específicas, sus subcaracterísticas y definiciones se encuentran en la imagen 6.

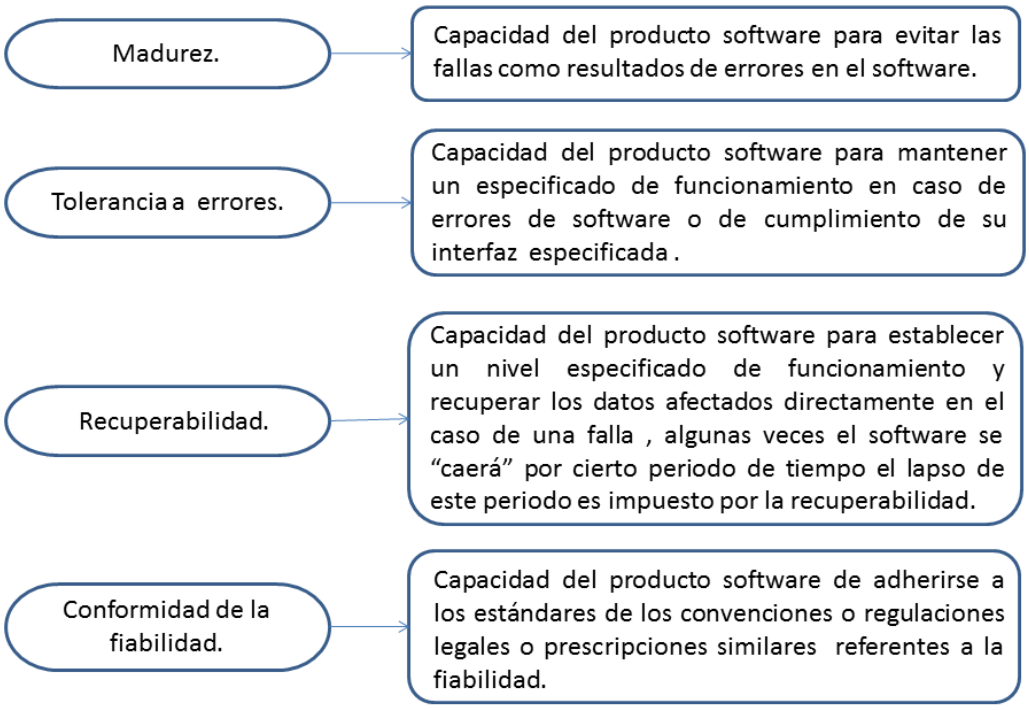


Imagen 6 Subcaracterísticas de fiabilidad

- **Usabilidad**

  Capacidad del producto software de ser aprendido, entendido, usado y atractivo al usuario cuando se está utilizando bajo condiciones específicas, sus subcaracterísticas y definiciones se encuentra en la imagen 7.

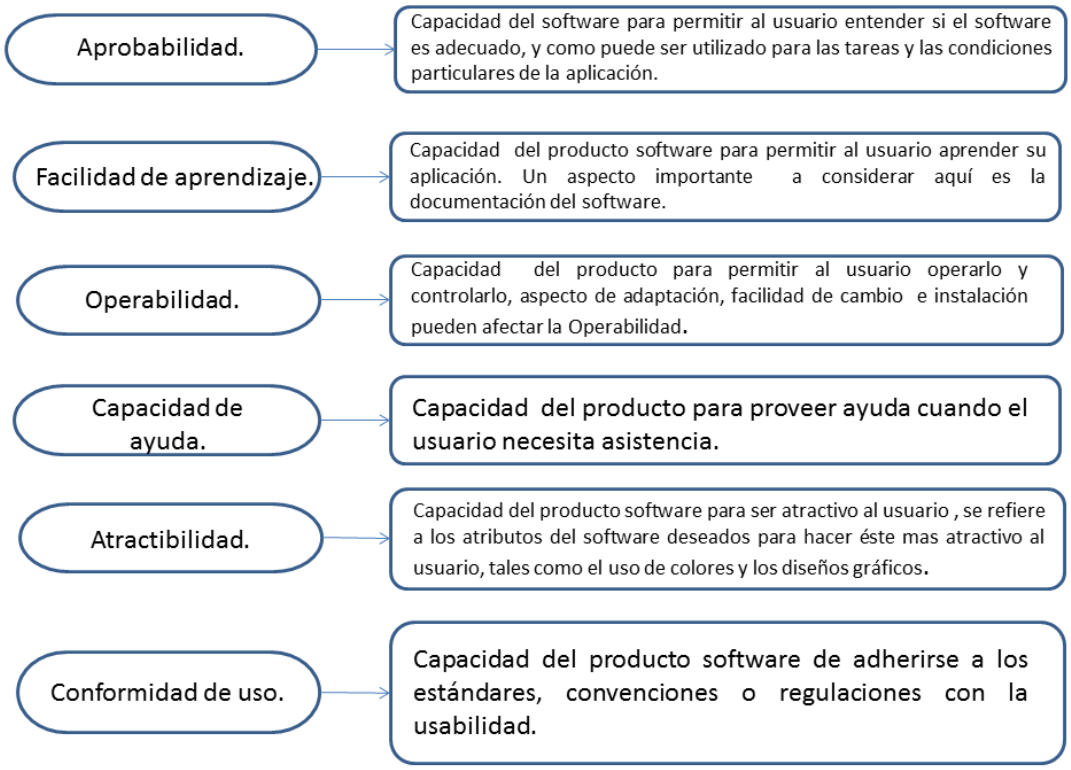


Imagen 7 Subcaracterísticas de fiabilidad

- **Eficiencia**

  Capacidad del producto software para proveer un desempeño adecuado con la cantidad de recursos utilizados bajo condiciones planeadas, sus subcaracterísticas y definiciones se encuentran en la imagen 8.

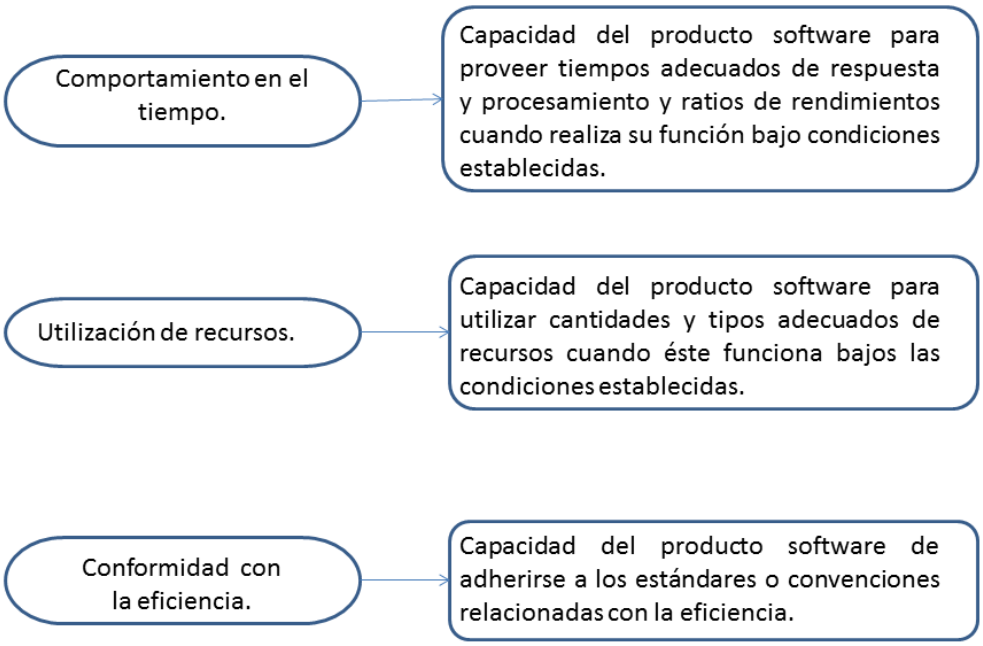


Imagen 8 Subcaracterísticas de eficiencia

- **Mantenibilidad**

  Capacidad del producto software para ser modificado, estas pueden incluir correcciones, mejoras o adaptación del software a cambios en el entorno, según especificaciones de requerimientos funcionales, sus subcaracterísticas y definiciones se encuentran en la imagen 9.

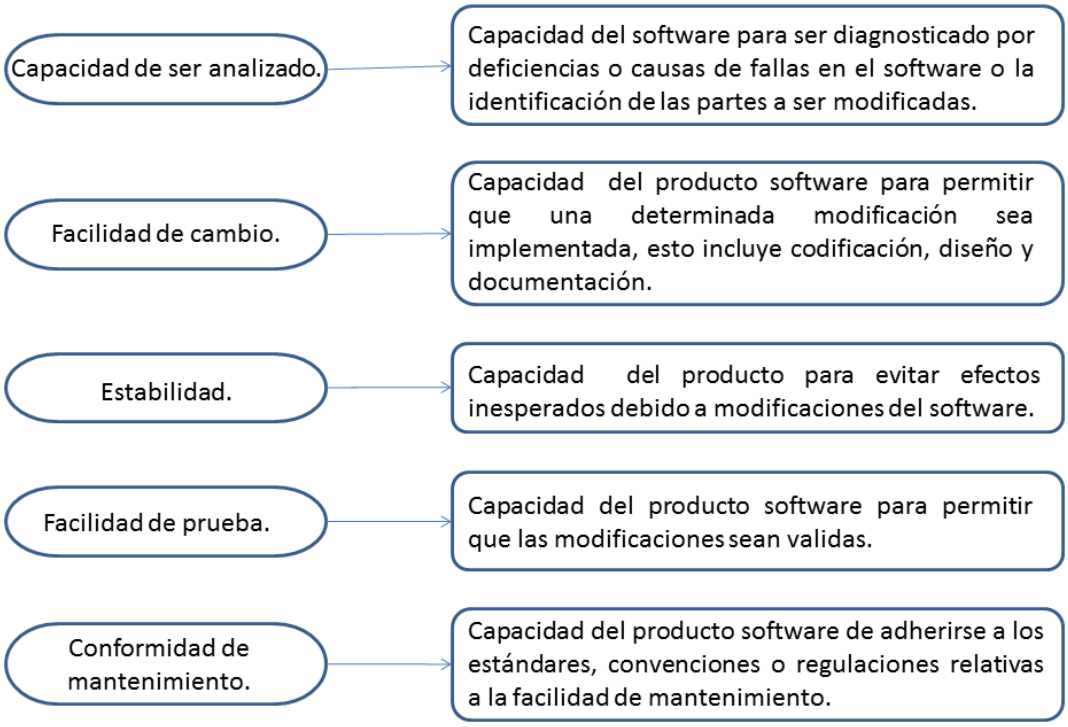


Imagen 9 Subcaracterísticas de mantenibilidad

- **Portabilidad**

  Capacidad de un software para ser trasladado de un entorno a otro, puede incluir entornos organizacionales, de hardware o de software, su subcaracterísticas y definiciones se encuentran en la imagen 10.

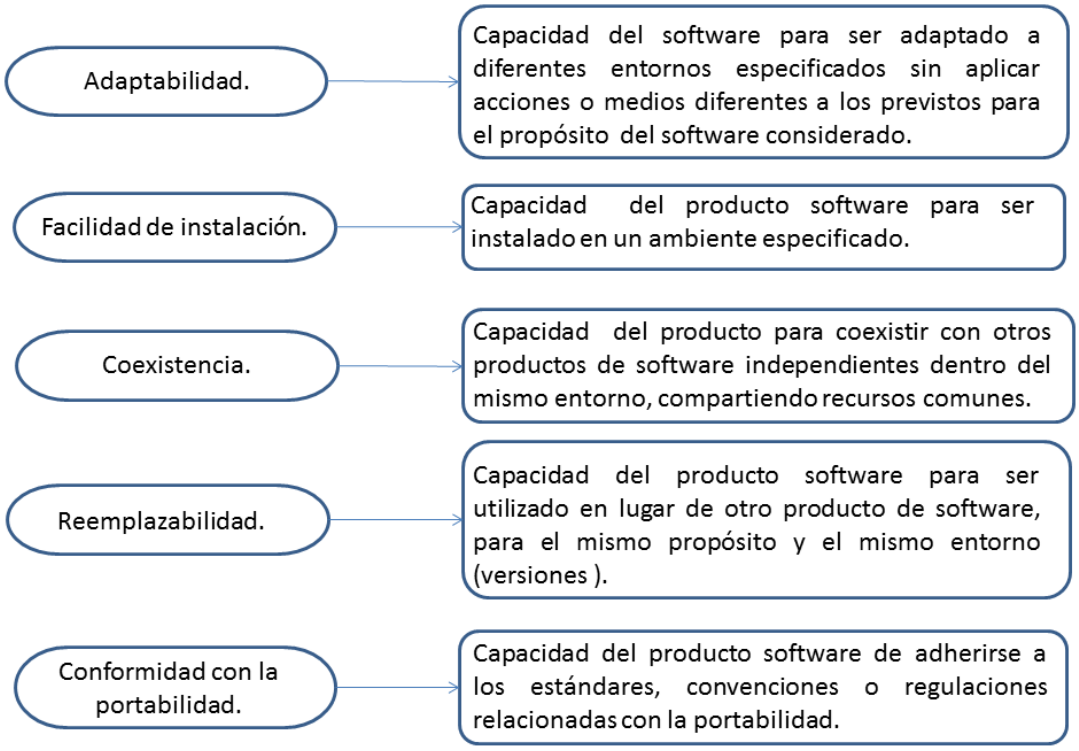


Imagen 10 Subcaracterísticas de portabilidad

2.4.2 Especificación de evaluación de calidad según ISO 2504n

En esta serie de estándares se presentan distintos aspectos y tareas relativas a la evaluación del producto software, se trabaja conjunto a la sección ISO 25010, sugiere la siguiente documentación:

- **ISO 25040 módulo de referencia para la evaluación.**

  Describe los requisitos generales a cumplir en la especificación y evaluación de la calidad del software, conjunto a esto proporciona una base para realizar dicha evaluación, además especifica los requisitos que deben cumplir los métodos para la evaluación y medida del software

- **ISO 25041 módulo de evaluación.**

  En esta parte se describen las estructuras y contenido de la documentación que se deben describir en los módulos de evaluación.

- **ISO 25042 proceso de evaluación para desarrolladores**

  Contempla requisitos y recomendación de carácter práctico para la implementación de la evaluación, cuando éste se da en paralelo con el desarrollo.

- **ISO 25043 proceso de evaluación para compradores**

  Describe requisitos y recomendaciones para la medida y evaluación sistemática de productos de software comerciales, productos desarrollados a medida o productos a modificar

- **ISO 25044 proceso de evaluación para evaluadores**

  Detalla los requisitos y recomendación para la evaluación de software de forma que dicha evaluación sea confiable y precisa. (Betancur, 2014).

2.4.3 Mantenibilidad.

2.4.3.1 Dimensión Mantenibilidad

Como se ha visto anteriormente, la mantenibilidad se define como la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas, perfectivas o adaptativas.

- **Perfectivas y Evolutivas**: se trata de la incorporación de nuevas prestaciones y funcionalidades al software, así como en mejorar el rendimiento de las existentes.
  Igualmente, de readaptar nuevos procedimientos de trabajo que consigan hacer más operativo el sistema. Esas nuevas prestaciones, en ocasiones, corresponden a nuevos desarrollos muy importantes y cuyo coste es, por tanto, muy elevado.
- **Correctivas**: trata de corregir los fallos y defectos de los programas, así como de los efectos derivados de éstos. En ocasiones el defecto en sí es muy leve, y por tanto su corrección fácil y rápida, pero las consecuencias derivadas de éste requiere de la reconstrucción de la integridad de los datos, lo cual se magnifica si afecta a varios o todos los clientes.
- **Adaptativas**: corresponde a los cambios que hay que realizar derivados de los cambios en los sistemas operativos, en el hardware, en la arquitectura del

sistema informático, etc. Incluso por los cambios relacionados con las propias tendencias del mercado, véase por ejemplo el cambio de un entorno cliente-servidor a un entorno web del software. Aunque suele corresponder a un porcentaje bajo en comparación con el resto de mantenimientos, ante ciertos cambios puede ocasionar una reescritura del código por completo o procesos de migración extremadamente manuales que ocasionan costes a veces inasumibles por el fabricante del software.

2.4.3.2 Sub-Dimensiones de Mantenibilidad

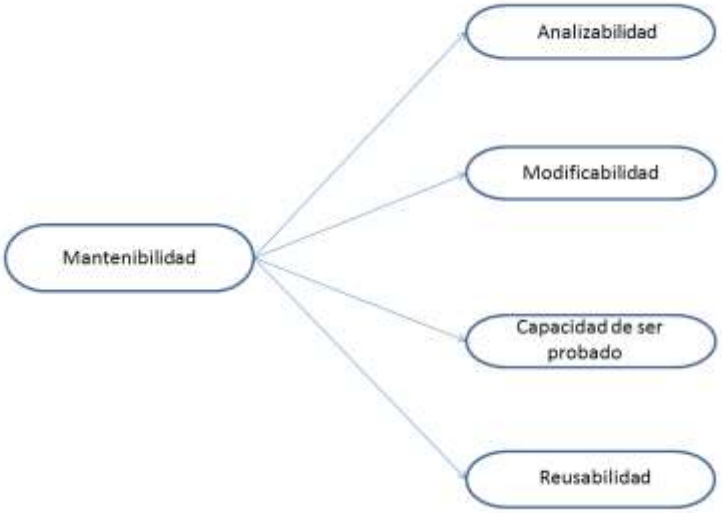


Imagen 11 Sub-Dimensiones Mantenibilidad

- **Analizabilidad**
  Se puede definir la analizabilidad como la facilidad para evaluar el impacto de un cambio sobre el resto del software, diagnosticar las deficiencias o causas de fallos en el software, o identificar las partes a modificar.
  Las métricas de analizabilidad deberían ser capaces de medir atributos tales como los recursos o esfuerzo del personal de mantenimiento o del usuario cuando intentan diagnosticar las deficiencias o causas del fallo del software, o identificar las partes a ser modificadas.

- **Modificabilidad**
  La modificabilidad es la capacidad del producto software que permite que sea modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño.

Un sistema cumple con responsabilidades. Una responsabilidad es una acción, un conocimiento que debe ser mantenido, o una decisión a ser tomada por un sistema.

La relación entre responsabilidades se define como acoplamiento entre la dos. El grado de acoplamiento entre dos responsabilidades se mide como la probabilidad en la que el cambio de una responsabilidad implica un cambio en la otra.

Hay varios parámetros a tener en cuenta a la hora de diseñar un software modificable

- ✓ **Costo promedio de modificar una responsabilidad:**
  Es basa en reducir el costo de modificar una responsabilidad y que esto no implique cambios en otras responsabilidades, reduce el costo de cualquier modificación que implique cambios en otra.
- ✓ **Grado de acoplamiento:**
  Reducir el acoplamiento entre dos responsabilidades, reduce el costo de una modificación en una de las responsabilidades responsabilidad.
- ✓ **Cohesión:**
  Las responsabilidades son asignadas a módulos del sistema. Si una responsabilidad tiene un alto grado de acoplamiento con la responsabilidad del mismo tipo, el costo de cambiar la primera responsabilidad es menor si las dos responsabilidades son asignadas al mismo módulo o conjunto.
  Los cambios tempranos en el sistema son menos costosos que los cambios hechos más adelante.

- **Capacidad de ser probado**

La capacidad de ser probado consiste en la facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios.

Las pruebas son básicamente un conjunto de actividades dentro del desarrollo de software. Dependiendo del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento de dicho proceso de desarrollo.

Existen distintos modelos de desarrollo de software, así como modelos de pruebas. A cada uno corresponde un nivel distinto de involucramiento en las actividades de desarrollo.

Hay varios tipos de pruebas:

- ✓ **Pruebas estáticas**
  Son el tipo de pruebas que se realizan sin ejecutar el código de la aplicación.
  Puede referirse a la revisión de documentos, ya que no se hace una ejecución de código. Esto se debe a que se pueden realizar “pruebas de escritorio “con el objetivo de seguir los flujos de la aplicación.

- ✓ **Pruebas dinámicas**
  Todas aquellas pruebas que para su ejecución requieren la ejecución de la aplicación.
  Las pruebas dinámicas permiten el uso de técnicas de caja negra y caja blanca con mayor amplitud. Debido a la naturaleza dinámica de la ejecución de pruebas es posible medir con mayor precisión el comportamiento de la aplicación desarrollada.

- **Reusabilidad**
  Es la capacidad de un activo que permite que sea utilizado en más de un sistema software o en la construcción de otros activos.
  La reutilización de software aparece como una alternativa para desarrolla aplicaciones de una manera más eficiente, productiva y rápida. La idea es reutilizar elementos y componentes software en lugar de tener que desarrollarlos desde un principio. (Valenciano, 2005, p.25)

2.5 Paralelo diferencias ISO 9126 E ISO 25000 cambios importantes

Tomando como punto de referencia que la ISO 9126 es la antecesora de la ISO 25000 se debe tener en cuenta los cambios generales contemplados en la tabla 1 y los cambios internos plasmados en la tabla 2.

Tabla 1 Paralelo ISO 9126 ISO 25000 cambios generales

ISO 9126	ISO 25000-SQUARE
Está dividida en 4 partes: - ISO 9126-1características de la norma - ISO 9126-2 métrica externa - ISO 9126-3 métrica interna - ISO 9126-4 métrica para la calidad en uso.	SQUARE se subdivide en: - ISO/IEC 2500n: División de dirección de calidad. - ISO/IEC 2501n: División del modelo de calidad. - ISO/IEC 2502n: División de medida de calidad. - ISO/IEC 2503n: División de requisitos de calidad. - ISO/IEC 2504n: División de evaluación de calidad
Pertenece a la primera generación de estándares de calidad de un producto de software.(permaneció desde 1977 hasta 2014)	Pertenece a la segunda generación de estándares de calidad de un producto de software.(rige desde 2005 hasta hoy día)

ISO 9126-1 se basa en dos partes bases conceptuales: - Calidad interna y externa - Calidad en uso	ISO 25010 al igual que si antecesora se basa en dos partes: - Calidad interna y externa - Calidad en uso
La Calidad interna y externa propone 6 características	La Calidad interna y externa propone 8 características
La Calidad en uso propone 4 características	La Calidad en uso propone 5 características
No tiene	Introducción a un nuevo modelo de referencia general
No tiene	Introducción de guías dedicadas y detalladas para cada revisión
No tiene	Introducción de elementos de medida de calidad dentro de la división de medida de calidad

Tabla 2 Paralelo ISO 9126 VS IS0 25000 Cambios internos

ISO/IEC 9126-1	SQUARE	CARACTERÍSTICAS
Funcionalidad-	Cambiado a Adecuación funcional	Este nombre es más preciso, y no provoca confusiones con otros significados e interpretaciones de funcionalidad
Interoperabilidad	Subcaracterística de Compatibilidad	Movido a Compatibilidad
Seguridad		Subcaracterística propia de ISO 9126- 1
Madurez	Disponibilidad	Es considerado que disponibilidad es mucho más importante que madurez
No implementada.	Robustez	Subcaracterística de SQUARE, implementada por su importancia para determinar la calidad.
Eficiencia-	Eficiencia de rendimiento	Renombrado por considerarla más puntual y especifica acotándola solo a rendimiento.
Usabilidad	Operabilidad	Renombrado para no provocar conflictos con otras definiciones

Comprensibilidad	Reconocimiento de adecuación	El nuevo nombre de SQUARE es mucha más preciso y entendible en aplicación.
Operabilidad	Facilidad de uso	Renombrado para mayor entendimiento de la subcaracterística.
No implementada	Útil	Nueva subcaracterística de SQUARE
No implementada	Accesibilidad técnica	Nueva subcaracterística de SQUARE
Seguridad	Seguridad	En SQUARE es una característica, en la ISO 9126 es una subcaracterística, se añade más importancia a esta característica, fundamentada en los avances de los ataques cibernéticos.
	Compatibilidad	No estaba suficientemente declarado en las subcaracterísticas de portabilidad en la ISO 9126-1,se considera de vital importancia incluirla como una característica propia en la ISO25000
	Interoperabilidad	En la ISO 9126-1 es una subcaracterística de Funcionalidad, es movida a una subcaracterística de compatibilidad.

Se observa que los cambios importantes son planteados en la tabla 2, SQuaRE hace modificaciones basadas en la forma equivoca en que es entendida la normativa por los grupos de trabajo y de desarrollo, junto con algunas adiciones de subcaracterísticas y características para la ISO 25000, considera el cambio más importante el paso de la subcaracterística de la 9126 “seguridad” a una característica de la 25000, además de estos existieron modificaciones de nombres y definiciones para de esta manera evitar los mal entendidos de interpretación y aplicación de la norma.

2.6 Pruebas de software

Son un conjunto de actividades que se realizan con el fin de identificar posibles fallos de funcionalidad, mantenibilidad, usabilidad y configuración de un programa o aplicativo.

A medida que crecen los proyectos informáticos, incrementa su nivel de complejidad e interoperabilidad, con lo cual trae la apreciación defectos (bugs), en la mayoría de los casos sin importancia, pero que pueden conducir a un deterioro de calidad del producto.

En la actualidad existen diversos tipo de pruebas aplicables a software como lo son pruebas funcionales, pruebas de sistemas, pruebas no funcionales, pruebas de caja negra, pruebas de caja blanca entro otros. (PMO informática, sf)

Se considera de vital importancia la conceptualización de las pruebas de caja blanca, principalmente en las pruebas unitarias debido a que estas son las directamente implicadas con la calidad del producto software.

2.6.1 Pruebas unitarias.

Estas pruebas se puede realizar a lo largo de los diferentes procesos dentro de las metodologías agiles, básicamente consisten en segmentos de código planteados para examinar si el código principal funciona acorde a lo que se espera que haga, tratando de suplir los requisitos funcionales del código y analizando el resultado de estas.

El desarrollo de esta tipo de pruebas, contempla 3 partes, el **Arrange**, donde se definen los requisitos a cumplir del código principal, el **Act**, esta parte contempla la creación de los fragmentos de código, los cuales serán usados para ejecutar las pruebas y donde se acumularan los resultados a medida que se vallan realizando estás, y por último el **Assert,** en esta sección se comprueban si los datos arrojados son correctos o no relacionados con los requisitos, a partir de estos resultados se determina si se continua, se realizan reparación o se detiene a replantear requerimientos, hasta que el defecto desaparezca o se mitigue.(APIUMHUB,2017)

2.6.1.1 Características

- **Automatizable:** Los resultados de cada test deben ser específicos, dando la oportunidad de automatizar las pruebas de forma grupal o individual.
- **Completas:** Al realizar las pruebas solo se tiene en cuenta segmentos del código principal, pero al final del proyecto debe tratarse en su totalidad.
- **Repetibles:** Independientemente de cuantas veces se realice un test el resultado siempre será el mismo, importando el orden en cada una de ellas, además pueden usarse en diversas ocasiones.

- **Independientes:** Los segmentos de código de cada test no interfieren en el libre desarrollo del proyecto, tampoco obstruye con el trabajo de otros desarrolladores al encontrase aislado,
- **Rápida creación:** Al tratarse de un trabajo extra para desarrolladores, estos test son de una fácil creación, aproximadamente 5 min en desarrollarse, están diseñados estrictamente para facilitar y acelerar el trabajo.

Entre las principales ventajas de las pruebas unitarias se encuentra ayudar a generar un código limpio y de calidad, detectar errores rápidamente facilitando los cambios y modificaciones necesarias y reducir el coste de corrección de errores en futuras entregas, además proporciona un entorno de trabajo ágil, debido a que genera búsqueda de errores en los etapas básicas del proyecto, sin necesidad de regresar al inicio del proyecto hallando defectos. (APIUMHUB, 2017)

2.7 Jenkins

De nada sirve tener las pruebas unitarias, si no se cuenta con alguna herramienta que analice y compruebe la cobertura según dichas pruebas, para esto se puede usar la Jenkins servidor de integración continua, gratuito, open-source y actualmente uno de los más empleados para esta función.

Jenkins puede indicar que se lancen métricas de calidad y visualizar los resultados dentro de la misma herramienta. También se podrá ver el resultado de los tests, generar y visualizar la documentación del proyecto. (García Oterino, 2015)

2.7.1 Características

- **Integración continúa y entrega continua**. se puede utilizar como un servidor de CI simple o se puede convertir en el centro de entrega continua para cualquier proyecto.
- **Fácil instalación.** Jenkins es un programa autónomo basado en Java, listo para ejecutarse de manera inmediata, con paquetes para Windows, Mac OS X y otros sistemas operativos similares a Unix.
- **Configuración fácil.** Jenkins se puede configurar y configurar fácilmente a través de su interfaz web, que incluye comprobaciones de errores sobre la marcha y ayuda integrada.
- **Multi-lenguaje, multiplataforma.** Jenkins posee diversidad de lenguajes y plataformas compatibles con su sistema entre ellos Android, C, docker, java, php, entre otros.

2.8 SonarQube

Como objetivo principal de este trabajo, se analiza y estudia el funcionamiento de la herramienta SonarQube, la cual se define dentro de su página oficial como “una plataforma de código abierto para realizar revisiones automáticas con análisis estático de código para detectar errores, olores de código (code smells) y vulnerabilidades de seguridad en más de 20 lenguajes de programación, incluidos Java, C #, JavaScript, TypeScript, C / C ++, COBOL y más. SonarQube admite un enfoque de fuga como práctica para codificar la calidad.”

En sus inicios a mediados del 2007, cuando se crearon las primeras líneas de código, los fundadores de SonarQube (originalmente llamado Sonar) tuvieron el sueño de que algún día todos los desarrolladores tuvieran la capacidad de medir la calidad del código de sus proyectos. Su lema: "**La inspección continua debe convertirse en la corriente principal como integración continua**"

Para hacer realidad este sueño, invirtieron todo su tiempo y energía en desarrollar SonarQube como un producto de código abierto, trabajando estrechamente con la comunidad. Hoy en día, SonarQube es utilizado por más de 85,000 organizaciones que, a cambio, proporcionan comentarios y contribuciones regulares. (SonarQube, sf)

Para suplir las necesidades de sus usuarios SonarQube cubre 7 ejes principales de la calidad del software y una vez analizado un proyecto muestra información detallada sobre la arquitectura y el diseño, comentarios del software, código duplicado, reglas de programación acordes con el lenguaje utilizado, bugs potenciales y su posible solución, datos referentes a la complejidad del proyecto e incluso datos sobre pruebas unitarias (en caso de tenerlas), como número de pruebas unitarias pasadas correctamente y porcentaje de cobertura de las mismas.

Los 7 ejes de la calidad del código fuente, mostrados en la imagen 12:

- El Código duplicado.
- La adherencia a Estándares de codificación.
- Las Pruebas Unitarias.
- La Complejidad del código.
- Las Evidencias de Fallos Potenciales.
- El nivel de Comentarios.
- La valoración sobre Diseño y Arquitectura.

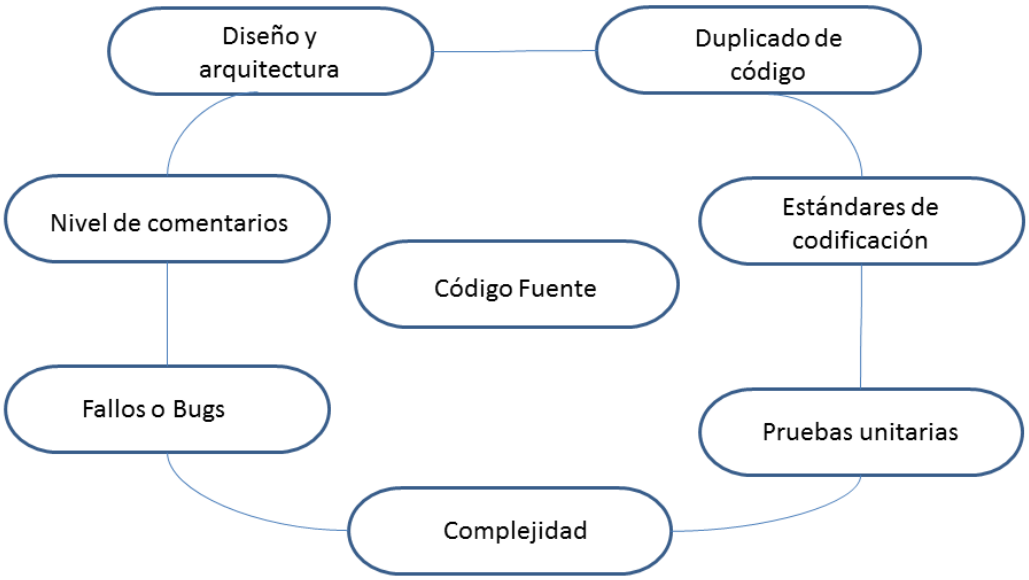


Imagen 12 Ejes de la calidad del código fuente

Ejemplos de errores cotidianos a la hora de programar en los cuales se hace más énfasis.

- ✓ Tener líneas de código duplicado (“Código duplicado”).
- ✓ No respetar los estándares de codificación y las mejores prácticas establecidas (“Estándares de codificación”).
- ✓ Tener una cobertura baja (o nula) de pruebas unitarias, especialmente en partes complejas del programa (“Pruebas unitarias”).
- ✓ Tener componentes complejos y/o una mala distribución de la complejidad entre los componentes (“Complejidad”).
- ✓ Dejar fallos potenciales sin analizar (“Fallos o Bugs”).
- ✓ Falta de comentarios en el código fuente, especialmente en las APIs públicas (“Nivel de comentarios”).
- ✓ Tener el temido diseño spaghetti, con multitud de dependencias cíclicas (“Diseño y arquitectura”).(Panel Testing, 2015)

2.8.1 Características.

- **Inspección continua**

SonarQube proporciona la capacidad no solo de mostrar la robustez de una aplicación, sino también de resaltar los problemas recientemente presentados. Con un Quality Gate en su lugar, puede arreglar la fuga y, por lo tanto, mejorar la calidad del código de manera sistemática.

- **Detectar problemas difíciles**

Los analizadores de código están equipados con potentes motores de flujo de datos sensibles a la ruta para detectar problemas delicados, como las referencias a puntos nulos, los errores lógicos, las fugas de recursos.

Se basa en: detectar errores, código de olores, vulnerabilidad de seguridad, activar reglas necesarias, explora todos los caminos de ejecución.

- **Centralizar la calidad**

Un lugar para proporcionar una visión compartida de la calidad del código para los desarrolladores, líderes tecnológicos, gerentes y ejecutivos a cargo de unos pocos miles de proyectos y también para actuar como una puerta de peaje para la promoción o el lanzamiento de aplicaciones.

Se basa en: todos los proyectos en un solo lugar, conjuntos de reglas compartidas, puerta de calidad unificada, servicios de proyectos cruzados, vistas basadas en el riesgo, Procesamiento paralelo de informes.

- **Integración de DevOps**

Según el portal paradigma digital define los DevOps como “un acrónimo inglés de development (desarrollo) y operations (operaciones), que se refiere a una metodología de desarrollo de software que se centra en la comunicación, colaboración e integración entre desarrolladores de software y los profesionales de sistemas en las tecnologías de la información (IT)”.DevOps es una respuesta a la interdependencia del desarrollo de software y las operaciones IT. Su objetivo es ayudar a una organización a producir productos y servicios software más rápidamente, de mejor calidad y a un coste menor”. (Ruiz, 2015)

Por su parte, SonarQube se integra con toda la cadena de herramientas DevOps, Es por eso que se proporcionan integraciones para MSBuild, Maven, Gradle, Ant y Makefiles.

- **Multi lenguaje**

Con SonarQube viene un analizador de código para cada lenguaje de programación principal. Cada analizador proporciona numerosas reglas para detectar problemas de calidad generales y específicos del idioma.

Entre los lenguajes más usados por SonarQube en sus scanner están Java, C/C++, HTML, phyton, Rubi, PHP entre otros. (SonarQube, sf)

2.8.2 Perfil de calidad (puertas de calidad).

- Es el conjunto de requerimientos de calidad que el código debe satisfacer
- Cada regla se define en términos de una regla de calidad.

- Cuando un fragmento del código no satisface una regla se reporta una no-satisfacción (Issue) (evidencia)
- Cada perfil de calidad es dependiente del lenguaje de programación.

Algunos ejemplos de reglas internas de la plataforma son:

- Métodos no deben tener una complejidad mayor que 10
- No debe haber **imports** no utilizados
- El nombre de las clases debe empezar por una letra en mayúscula
- Una variable no puede ser asignada a si misma
- No debe haber direcciones IP cableadas en el código

2.8.3 INFORMACION SOBRE REGLAS

a. Severidad: Refleja que tan crítico es el problema:
   - **Blocker:** Este código no puede entrar en producción.
   - **Critical:** si se trata de una emergencia, esto puede pasar a la producción, pero debe solucionarse inmediatamente después.
   - **Major**: debe solucionarse rápidamente, pero puede esperar hasta el siguiente intervalo programado.
   - **Minor:** de poca importancia pero corregible.
   - **Info**: información.

b. Tags(etiquetas): Agrupa reglas del mismo tipo por ejemplo:
   - Naming(nombrar)
   - Bugs
   - Security
   - Unused(no usado)

c. Tags más comunes para el perfil de Java
   - **Bug**- Algo está mal y probablemente afectará en producción.
   - **Performance** - Afecta el desempeño de la aplicación
   - **Security** - Está relacionado con la seguridad de una aplicación **OWASP-10** - Relacionado con una regla de los estándares de seguridad OWASP

d. Unused - código inútil, Ej. Una variable privada que nunca se utiliza.

e. Convention - Convención de formateo de código.

f. Pitfall - Nada está mal todavía, pero algo podría ir mal en el futuro.

g. Clumsy - Se realizan pasos adicionales para cumplir con algo que debería estar hecho de manera más clara y concisa.

h. User- **experience** - No hay nada mal en el código técnicamente hablando, pero puede que haga que la usabilidad no sea la mejor.(Universidad de los Andes,2015).

2.8.4 Deuda Técnica y SQALE Rating

Si se observa, los valores de la métrica de “Deuda Técnica” están expresados en unidades de tiempo. Así que la interpretación de lo que SonarQube considera como “Deuda Técnica” es el tiempo que se tendría que invertir para corregir esa carencia, ya sea porque es un mal código o porque es una deuda técnica asumida previamente.

A efectos prácticos, ayuda a entender que cuanta menor “Deuda Técnica” tenga un proyecto, se puede considerar que está en un mejor estado de “salud”. Y al asignar un peso(valor) temporal a las actividades a corregir dependiendo de la gravedad o urgencia de ésta, puede servir para hacerse una idea del por qué se tarda tanto en incluir ciertos cambios y si es necesario asignar recursos para amortizar la deuda, refactorizar, o incluso si podemos considerar la “banca rota” del proyecto, no seguir invirtiendo en nuevas funcionalidades y como responsables pedir un presupuesto para que éstas sirvan como embrión de otro proyecto en el que la inversión de su desarrollo no implique seguir acumulando deuda.

El detalle de cómo se calcula, y cómo SonarQube lo muestra, está íntimamente relacionado con la “Deuda Técnica”, más concretamente con el ratio de la “Deuda Técnica” (SonarQube, sf).

En un enfoque práctico esto sirve para transmitir de forma rápida, breve y visual la calidad del proyecto al responsable dándole una categoría entre la A y la E siguiendo la siguiente escala:

A. **Ratio de “Deuda Técnica” menor al 10%.** Se considera que el proyecto está sano y no habría que hacer nada especial.
B. **Ratio de “Deuda Técnica” entre el 10% y el 20%.** Se puede considerar que está en unos parámetros aceptables, pero hay que seguir de checheo periódico para vigilar que no empeora. Se puede empezar a plantearse tomar medidas preventivas
C. **Ratio de “Deuda Técnica” entre el 21% y el 50%.** Entra en unos valores en los que la salud ya no es buena, toca a tomar medidas correctivas. Si bien el carácter de urgencia puede venir marcado por la desviación de los datos respecto a los valores aceptables.
D. **Ratio de “Deuda Técnica” entre el 51% y el 100%.** Es urgente tomar medidas que lleven de vuelta el proyecto a zonas más saludables.
E. **Ratio de “Deuda Técnica” superior al 100%.** Se puede decir que se ha entrado en la zona de rescate, hay que tomar una decisión entre apostar por un rescate, con todos los esfuerzos que esto significa de tiempo y dinero, o bien dejar que el proyecto entre en banca rota para buscar otro camino fuera del mismo.

Esta categorización del SQALE Rating por letras y colores, similar al de certificación eléctrica, así como lo da a entender la imagen 13.

Imagen 13: Ratio deuda técnica

2.8.5 Nivel de cumplimiento de reglas

El análisis estático permite establecer reglas que faciliten la implantación de buenas prácticas a los desarrolladores para de esta manera reducir la aparición de errores cuando el sistema se lleva a producción.

El conjunto de reglas que se pueden establecer siempre estará condicionado a las herramientas que se utilicen para el análisis.

Los resultados del análisis se agrupan en cinco severidades diferentes. Cada una de estas severidades tiene un peso diferente que afecta al nivel de cumplimiento de las reglas:

- Bloqueante: la violación de algunas de las reglas requiere una solución inmediata pues se trata de un problema crítico (tiene un peso de 10)
- Crítico: se recomienda la solución del problema porque hay bastantes probabilidades de que genere un error posterior (tiene un peso de 5)
- Serio: quizás pueda derivar en un problema, o está en contra de las buenas prácticas, por lo que debería solucionarse (tiene un peso de 3)
- Leve: no parece derivar en un error, pero si se soluciona el desarrollo se acerca más a estándares (tiene un peso de 1)
- Informativo: sería bueno cumplirlo, pues se trata de aspectos que dan más consistencia al código, aunque no son fundamentales (tiene un peso igual a 0)

El nivel de cumplimiento se obtiene para cada regla en función del peso de la severidad y del número de veces que se detecte una evidencia de su incumplimiento.

Cada una de las reglas de buenas prácticas se categoriza en función de la característica de calidad a la que afecta, de manera que se puedan resolver problemas

concretos específicos en el menor tiempo posible. Esta categorización se lleva a cabo en conformidad con las características de calidad de la ISO25000**.**(Badenas,2014)

2.9 ESTADO DEL ARTE.

El rápido cambio de las tecnologías en los últimos años ha venido atravesando un sinfín de problemas en cuanto software, uno de los más importantes es la escases calidad de los productos. En la década de 1990, las principales corporaciones reconocieron que cada año se desperdiciaban miles de millones de dólares en software que no tenía las características ni la funcionalidad que se habían prometido. (Pressman, 2010, p.338)

Por este motivo se encontraron diferentes estudios relacionados con la calidad del producto software en cuanto a aplicación, modelos y estructura. A nivel internacional podemos encontrar la tesina de la Universidad Nacional de la plata Argentina realizada por **Tello, Pablo Gabriel,** denominada como "**Evaluación de Calidad de un Producto de Software**" y resume su trabajo de la siguiente manera "La presente tesina es un trabajo de investigación aplicada sobre la Evaluación de Calidad de un Producto de Software.
El trabajo para mejorar la calidad del software se fue fortaleciendo a medida que el uso de éste se fue integrando a la vida de las personas, ya que el software de alta calidad proporciona beneficios a la organización que lo produce y a la comunidad de usuarios finales. Además el uso de normas y estándares para lograr mayor competitividad es cada vez más habitual en las PyMEs desarrolladoras de software.
El objetivo de la tesina es analizar los estándares relacionados a la calidad para los productos de software:
ISO/IEC 9126 - Calidad del Producto, y para la evaluación de un producto de software: IRAM-ISO/IEC 14598 –Evaluación del Producto de Software. Realizar una evaluación de calidad sobre un producto de software concreto según los estándares mencionados, en particular evaluar las características funcionalidad y confiabilidad del producto, según las métricas definidas por estos estándares, y así detectar las debilidades del mismo para alimentar un proceso de mejora continua.
Se realiza además una presentación general sobre la familia de la norma ISO/IEC 25000, y se mencionan las ventajas y desventajas de los estándares estudiados. "(2016, p.1)
Se considera necesario incluir esta tesina debido a su estudio puntual y exacto de las normativas ISO 9126, ISO 14598, conjunto a orientaciones y explicaciones de términos implicados es las normas las cuales son base de este proyecto.

Por otro lado se cuenta con un trabajo realizado en **Instituto Politécnico Nacional** de la ciudad de Vallejo, México, a cargo de **Sandra Dinora Orantes Jiménez,** titulado "**Calidad de Software en el uso de Metodologías Ágiles para el Desarrollo de Software**" , resumiendo el proyecto como "El desarrollo de software no es tarea fácil y por ello, existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo; así, por un lado se encuentran propuestas tradicionales que se centran en el control del proceso, estableciendo de forma rigurosa

actividades involucradas, artefactos que deben producir y herramientas y notaciones que se emplearán.

Todas las propuestas han demostrado ser efectivas y necesarias en varios proyectos, pero también han presentado problemas en otros tantos. Se sugieren mejoras, como admitir en los procesos de desarrollo más actividades, artefactos y restricciones, en base a los puntos débiles detectados, como consecuencia se tendría un proceso de desarrollo complejo, que es posible que limite la propia habilidad del equipo para llevar a cabo el proyecto.
Es así que, en el proceso de buscar soluciones, hoy en día se encuentran metodologías que se centran en otros aspectos, como por ejemplo el factor humano o el producto de software, siendo esta la filosofía de las Metodologías Ágiles, donde se da mayor importancia a la colaboración con el cliente y al desarrollo incremental del software con iteraciones cortas, mostrando su efectividad en proyectos con requisitos cambiantes y donde se exige reducir drásticamente los tiempos de desarrollo, pero manteniendo una alta calidad. Por consiguiente, esta investigación se centra, en lograr mantener la Calidad de Software en ciclos de desarrollo cortos como los que plantean las Metodologías Ágiles.” (2007, p.1)

El trabajo de Dinora Orantes sirve como fundamento para entender la parte funcional de las normativas en diferentes de metodologías agiles y

Ahora a nivel local se encuentra “**modelo de calidad para evaluar el software desarrollado en el centro de investigación aplicada y desarrollo en tecnologías de información CIADTI**” presentado por **Y.S. Estévez** resumiéndolo se tiene “El presente artículo presenta un modelo de calidad de producto software para el Centro de Investigación Aplicada y Desarrollo en Tecnologías de Información, CIADTI, de la Universidad de Pamplona, que a diferencia de los modelos generales existentes pretende medir la calidad general de un producto software mediante la valoración de factores y características aplicadas a productos parciales dentro del proceso de desarrollo de software, es decir, a artefactos que se consideran relevantes dentro de la organización y a los cuales se les puede valorar mediante factores que fueron inicialmente pensados para productos finales del proceso de desarrollo. Se referencian antecedentes investigativos sobre modelos de calidad en el desarrollo de software, se realizó un diagnóstico al actual proceso de desarrollo del producto del CIADTI a partir de la norma ISO 9126.

También se aplicaron instrumentos para la recolección de información a los desarrolladores y coordinadores del proceso. A partir de allí se analizaron las categorías y atributos para la formulación del modelo, se formula el modelo final y se aplica a los artefactos existentes. Finalmente, se analizan los resultados y el modelo se valida frente a un grupo de expertos para una mayor objetividad.”(2014, p.1)
Éste artículo se toma por la veracidad de la información y relación con diversos modelos de calidad y su implementación en una institución de la vida real.

3. Análisis de SonarQube

En este capítulo se abarca el proceso de instalación de SonarQube, e implementación de plugins y devops necesarios para conseguir una configuración apropiada a la hora de realizar las diferentes pruebas de escáner, luego se mostraran los procesos de escáner, en la plataforma SonarQube, en el IDE de NetBeans y en el devops de Jenkins, junto con un análisis de las métricas arrojadas en cada uno de estos métodos.

3.1 Requisitos de instalación.

- **Requisitos de software.**

  El único requisito previo para ejecutar correctamente SonarQube es tener Java en su versión Oracle JRE 8 u OpenJDK 8 instalado previamente en la máquina, esto para el sistema operativo Windows de la versión 7 en adelante.

- **Requisitos de hardware.**

Tratándose de una escala individual o en equipo, el servidor de SonarQube requiere por los menos 2 GB de RAM para funcionar de manera eficiente y 1 GB de RAM libre para el sistema operativo (Guadin, 2018), en cuanto a disco duro no requiere una gran cantidad de almacenamiento (4GB).

3.2 Instalación local SonarQube.

Iniciando con el proceso de instalación y configuración, lo primero que se realizó fue escoger una versión de la página oficial que supla todas las características necesarias para conllevar los objetivos del proyecto.

La versión usada fue la 6.4, basándose en la mantenibilidad, y seguridad, correcciones de errores de versiones anteriores y siendo esta una de las más estables en cuanto a reglas, cabe destacar la gran cantidad de reglas de evaluación que tiene incluidas SonarQube, el grupo diseñador de esta plataforma considera casi innecesario la creación de reglas durante el proceso de análisis, esto debido a la gran cantidad de reglas nuevas generadas en cada una de sus versiones lanzadas, y el intercambio de comentarios y contribuciones de las empresas que usan SonarQube ,cubriendo así la mayoría de los posibles errores.

Al analizar código fuente SonarQube se basa en 3 herramientas para generar sus reglas y búsqueda de errores, basadas principalmente en mantenibilidad y seguridad, estas son Pmd, Cpd y findbugs.

- PMD es un analizador de código fuente. Encuentra fallas de programación comunes como variables no utilizadas, bloques de captura vacíos, creación innecesaria de objetos, etc. Es compatible con Java, JavaScript, Salesforce.com Apex y Visualforce, PLSQL, Apache Velocity, XML, XSL.
- CPD es un detector de copiar-pegar. CPD encuentra código duplicado en Java, C, C ++, C #, Groovy, PHP, Ruby, Fortran, JavaScript, PLSQL, Apache Velocity, Scala, Objective, entre otros. (PMD, 2019)
- Findbugs es un programa que utiliza el análisis estático para buscar errores en el código Java. Es un software libre, distribuido bajo los términos de la Licencia Pública GNU. (FindBugs, 2015).

Estas reglas están distribuidas en 4 grandes grupos: vulnerabilidad, bugs, seguridad y código smell, para obtener información más detallada sobre todas las reglas que maneja SonarQube, esta se puede encontrar en https://rules.sonarsource.com/java/RSPEC-5300

Por defecto SonarQube trae incluidas las reglas más usuales de los diversos lenguajes, en el caso de necesitar otro tipo de reglas estas se descargan como plugins extras en la plataforma.

3.2.1 Proceso de configuración.

a) Dirigirse a la página oficial de SonarQube (https://www.sonarqube.org), descargar la versión 6.4, conjunto a la última versión del scanner 3.2
b) Se descomprime los dos archivos **.rar** en el directorio **C:\.**
c) Por defecto SonarQube trabaja localmente en el puerto 9000, si se desea cambiar esta opción basta con ingresar al archivo **C:\sonarqube-6.4\conf\sonar.properties (ver imagen 14)** y editarlo a conveniencia.

```
# WEB SERVER
#
# Binding IP address. For servers with more than one IP address, this property specifies which
# address will be used for listening on the specified ports.
# By default, ports will be used on all IP addresses associated with the server.
#sonar.web.host=0.0.0.0

# Web context. When set, it must start with forward slash (for example /sonarqube).
# The default value is root context (empty value).
#sonar.web.context=
# TCP port for incoming HTTP connections. Default value is 9000.
#sonar.web.port=9000
```

Imagen 14 Archivo sonar.properties

En este mismo archivo podemos modificar también la conexión a la base de datos si se considera necesario. (Para este caso es omitida)

Si desea modificarlo basta con buscar y descomentar las siguientes líneas o las que se considere necesarias, la primera línea ubica el host donde se desea arrancar el servicio para el caso es local, y la siguiente línea es el puerto donde se encuentra por defecto es el 9000.

sonar.web.host=localhost

sonar.web.port=9000

d) A continuación se debe ejecutar el servidor SonarQube, esto depende del sistema operativo y las características del mismo. Para poder llevar a cabo este proceso, basta con ingresar a **C:\sonarqube-6.4\bin\windows-x86-64** y ejecutar el archivo **StartSonar.bat** dándole doble clic.

Por tratarse de la primera vez que se arranca este servidor, demorara un poco más de lo convencional, al finalizar la configuración muestra en pantalla el siguiente resultado. (Ver imagen 15).

Imagen 15 Ejecución del servidor SonarQube

Para poder ingresar al servidor lo que se debe observar es la última línea del terminal, esta debe tener **SonarQube is up,** lo cual quiere decir que el servicio está activo y listo para el uso.

e) Ahora se comprueba en el navegador de preferencia, que SonarQube se encuentra disponible, para ello introducir la siguiente url: **http://localhost:9000/**, donde su página de inicio es la mostrada en la imagen 16.

Imagen 16 Pantalla de inicio SonarQube

Las credenciales por defecto para el usuario y la contraseña, son la palabra “**admin**”.

Al ingresar estos valores se obtiene la información básica de proyectos escaneados con anterioridad, o la página lista para recibir los siguientes escaneos, como se observa en la **imagen 17.**

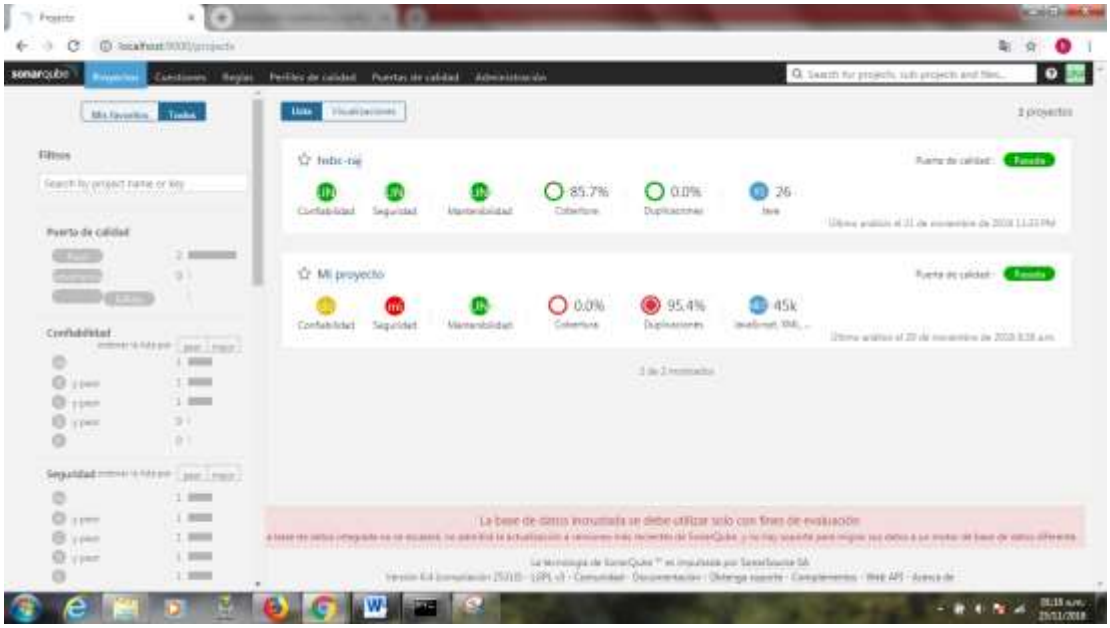


Imagen 17 inicio de sesión SonarQube

Teniendo claro el proceso de configuración de SonarQube de manera local se procede a configurar la plataforma dentro del IDE de NetBeans, junto con una breve explicación de su importancia para este trabajo.

3.2.2 SONARQUBE EN NETBEANS PLUGIN “RADAR”

SonarQube posee un alto grado de acoplamiento con diferentes IDE’s, los más conocidos usados por esta plataforma son NetBeans, Eclipse, IntelliJ IDEA, cada uno con un plugins especializado para poder realizar su respectivo escaneo.

Antes de iniciar con el proceso de scanner con ésta IDE, se considera necesario dar el concepto de NetBeans generado por su página oficial “es un entorno de desarrollo, una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el NetBeans IDE. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.” (NetBeans, sf)

Por otro lado se trabaja sobre el plugins Radar, este es un complemento únicamente para proyectos de java y éste poder realizar la inspección de calidad de SonarQube directamente sobre el IDE NetBeans.

La información que se debe tener en cuenta es la compatibilidad de las versiones de estas 4 herramientas expuestas en la imagen 18

HERRAMIENTA	VERSION
JAVA JDK	8
SONARQUBE	6.4
NETBEANS	8.2
PLUGIN RADAR	3.1

Imagen 18 Herramientas y versiones

Teniendo claro lo anterior, se procede a la instalación de “Radar”.

Nota: Tener previamente instalado y configurado NetBeans, y tener activo el servicio de SonarQube.

a. Por defecto el IDE trae consigo el plugins de “radar” en la versión nombrada, haciendo falta únicamente su activación.
b. Para esta activación, buscar la documentación respectiva en la siguiente página http://plugins.netbeans.org/plugin/51532/radar-netbeans

Posteriormente se implementará el escáner en esta herramienta con su respectivo análisis, con lo anterior se da paso, al proceso de creación de pruebas unitaria necesarias para las métricas de cobertura solicitadas por SonarQube.

3.2.3 PRUEBAS UNITARIAS JUNIT

Se sabe que a raíz de las diferentes pruebas unitarias se puede establecer el porcentaje de cobertura de un software, de esta manera poder determinar de cierto modo la calidad de este producto, en la actualidad existen diferentes herramientas que facilitan este trabajo, una de las más usadas es JUNIT, quien se basa en casos de pruebas y suites de prueba, los casos de prueba son clases o módulos que disponen de métodos para probar los métodos de una clase o módulo concreto. Así, para cada clase que quisiéramos probar definiríamos su correspondiente clase de caso de prueba. Mediante las suites podemos organizar los casos de prueba, de forma que cada suite agrupa los casos de prueba de módulos que están funcionalmente relacionados. (Universidad de Alicante, 2014)

Las pruebas generadas con esta herramienta poseen una facilidad en cuanto a diseño y tiempo de uso, debido a que el genera automáticamente el segmento de código de prueba a partir de cada uno de los módulos o métodos del proyecto, dejando únicamente al programador la configuración según los valores planteando por los requerimientos del software.

Como en la mayoría de los casos siempre se cuenta con ventajas y limitantes, las ventajas más importantes son el permitir corregir errores sin necesidad de afectar la funcionalidad del componente analizado, y contribuye al proceso de integración de componentes del sistemas al permitir el funcionamiento correcto de cada componente de manera individual, por otro lado contamos con las limitaciones tomando como punto de partida el no permitir identificar errores de integración ni errores de desempeño del sistema completo, debido a que solo trabaja pequeñas partes del código, además, son poco prácticas para probar todos los casos posibles para los componentes que se testean, existen también limitantes en cuanto a su compatibilidad con diversos modelos de desarrollo, siendo más compatible con aquellos dirigidos por pruebas como metodologías agiles tales como XP.(Rodríguez y López, sf, p.14)

3.2.4 JUNIT y NetBeans uso

a) Trabajando bajo el IDE de NetBeans, encontramos un plugin para Junit instalado por defecto desde la versión 8.0.2, lo cual ahorra esté proceso, en el caso de no encontrarse activo o instalado, se procede a instalar en plugin de manera convencional (el plugin se encuentra con el nombre Junit).

b) Para empezar con el proceso de creación de pruebas unitarias, se debe ubicar en el directorio donde se encuentre el archivo .java el cual contiene los métodos o modelos del software del software a probar, para este caso se llama métodos.java.

c) Se da clic derecho en el archivo nombrado, **>tools>create/update test,** esta ruta nos genera una pantalla para darle el nombre y los tipo de pruebas a usar así como se observa en la imagen 19.

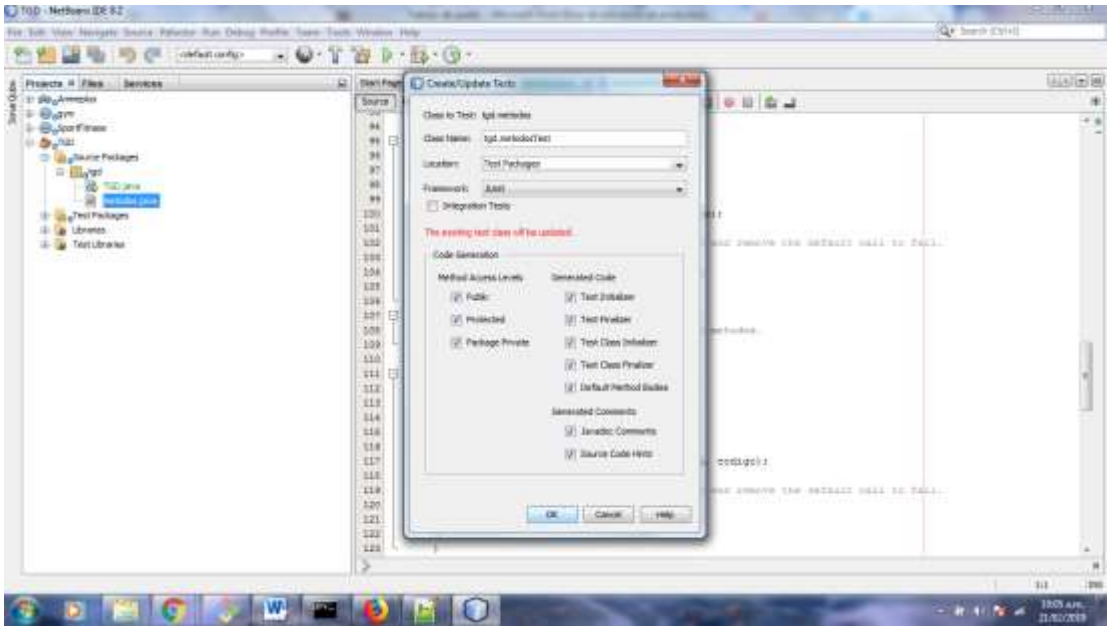


Imagen 19 Creación pruebas unitarias

d) Al crear la prueba, automáticamente este genera una nueva clase, con la posible prueba para cada uno de los métodos del archivo metodos.java, este nueva clase se encuentra alojada en el paquete test.

e) Por el momento solo se ha creado la estructura de las pruebas, falta la configuración interna de cada una de ellas, donde se determina un valor de entrada y la respuesta esperada para cada caso según los

requerimientos del software, debido a la facilidad del proyecto manejado, se trabaja lo básico de las pruebas unitarias con Junit, para estos casos es las pruebas van relacionadas con rangos de valores y validación de existencia de datos.

f) Para comprobar lo dicho anteriormente se tomara los métodos test llamados **testValidarNota** este método valida que se ingrese una nota válida y retorna un valor de true si es aceptado el valor **y TestBuscarPosicion** este método busca dentro de un vector si se encuentra un valor dado y retorna la posición del valor.

g) En el primer método se ingresa un valor doublé de 3.5 y se espera que retorne el valor de “true”, en el segundo método se ingresan dos datos un int con un valor de 3 y un vector con los valores (1, 2, 3, 4, 5) y se espera que retorne un valor de 3 (recordar que la posición de los vectores empiezan en 0), así como lo muestra la imagen 20

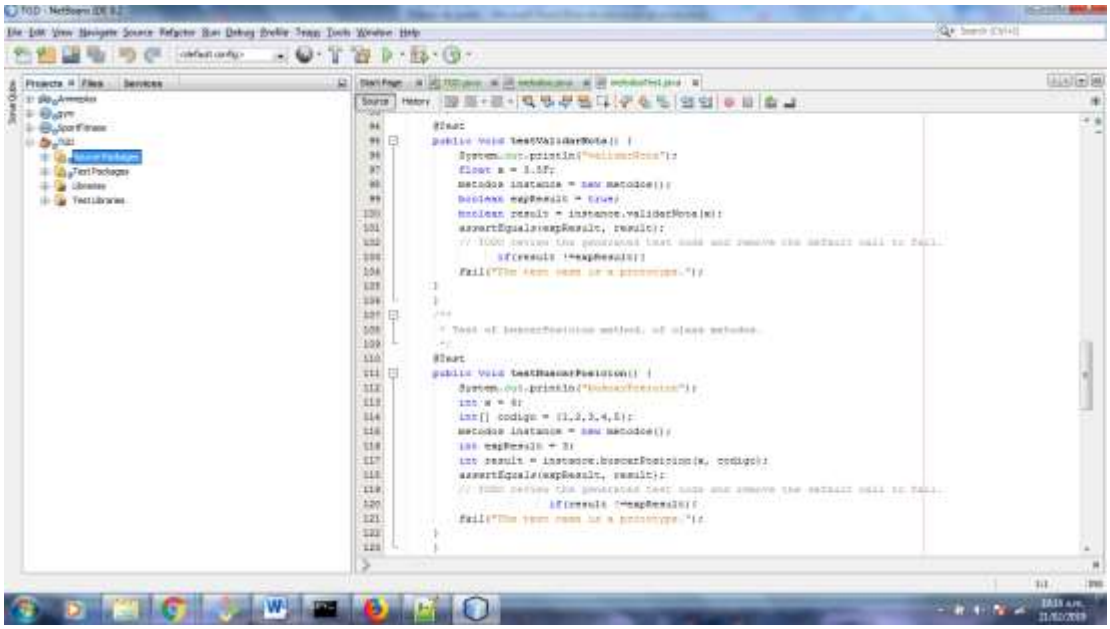


Imagen 20: Valores para pruebas unitarias

h) La mayoría de módulos de prueba son fallidos esto debido a que solo se usaron dos de los métodos test creados. (ver imagen 21)

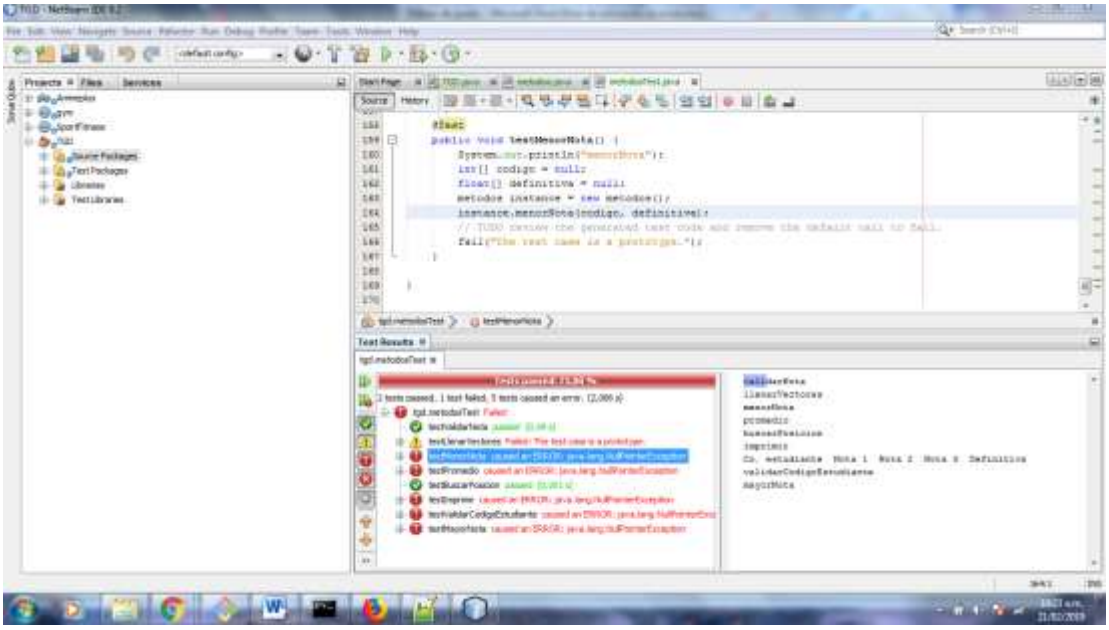


Imagen 21: ejecución de pruebas unitarias

Al observar el resultado de la ejecución se denotan los dos métodos test que se han modificado los cuales se muestran de color verde, el color rojo representa que existe una falla en los resultados esperados o un error en la configuración de la prueba.

Por el momento se obtienen las posibles fallas y aprobaciones de las pruebas unitarias, pero se necesita analizar la implementación de las pruebas de una manera más exacta y precisa por lo general en métricas, con el uso de la plataforma Jenkins podremos realizar este proceso de una manera fácil, además de vincularlo con SonarQube, para incluir la medición de severidad y demás métricas incluidas en esta herramienta.

3.3 JENKINS

Para hacer efectivas las pruebas unitarias dentro de las métricas de SonarQube en cuanto a cobertura es obligatorio la adaptación de estas dentro del devops de Jenkins quien al vincularse con SonarQube contempla las mediciones basadas en la cantidad de pruebas escaneadas.

3.3.1 Configuración.

a) Antes de proceder con la configuración, la descarga se basara en la versión 2.138.3 quien cuenta con un soporte a largo plazo, y se hace a través de la página oficial (https://jenkins.io/download/), dependiendo del sistema operativo y versión.
b) El procedimiento de instalación es la cotidiana de los sistemas operativos Windows, al terminar el proceso automáticamente abre el navegador predeterminado para continuar con la configuración, por defecto el panel de Jenkins trabaja en el puerto 8080 y en la ruta del host local (localhost:8080), en la imagen 22 se visualiza la primer pantalla al acceder al servidor local el cual solicita una contraseña presente en los archivos de configuración, dicha credencial se encuentra en una ruta generada por Jenkins la cual tiene un color rojo, dejando solo copiar y pegar esta contraseña.

Imagen 22 Primer acceso Jenkins

c) Al ingresar correctamente la información solicitada, este empieza a descargar los plugins que trae por defecto, este proceso demora dependiendo de la velocidad de internet y la cantidad de plugins a instalar como en la imagen 23.

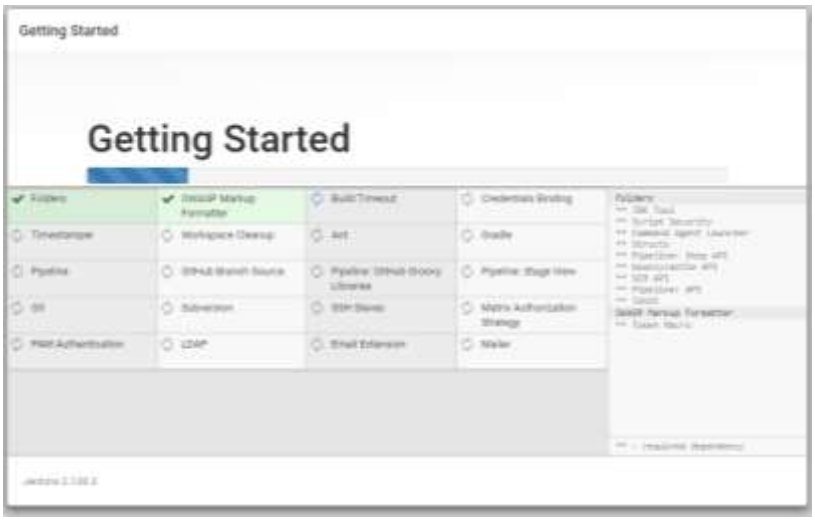


Imagen 23 Instalación de plugins

d) Terminada la descarga e instalación de los plugins, el paso a seguir es diligenciar un formulario teniendo en cuenta la imagen 24 con las credenciales que creamos convenientes.

Imagen 24 formulario de primer usuario.

e) En este punto se muestra el panel de configuración (ver imagen 25), con cada una de sus funciones, las mas usadas para este caso es “Nueva Tarea” y “Administrar Jenkins”

Imagen 25 Panel de configuración Jenkins

f) Ahora toca instalar y configurar Maven y el scanner de SonarQube, para SonarQube es necesario instalar su respectivo plugin, para Maven no es necesario, en el panel principal de Jenkins ubicarse en **>Administrar Jenkins >administrar plugins>todos los plugins**, en el slot de filtro escribir sonar e instalamos el plugin que aparece en la imagen 26.

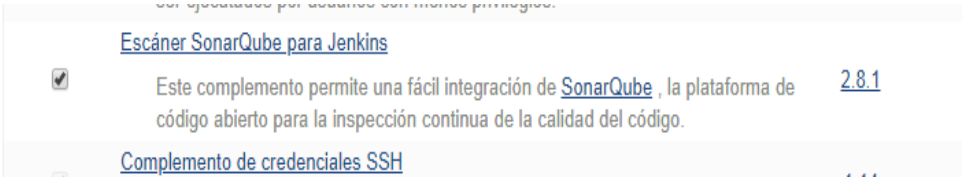


Imagen 26 Scanner SonarQube para Jenkins

g) El siguiente paso es configurar Maven para esto se especifica la siguiente ruta de acceso **>Administrar-Jenkins>configuración-de-herramientas-globales>Maven>añadir Maven**, en este punto colocar solamente en el nombre, “Maven”, para tenerlo como identificador como lo señala la imagen 27, este mismo proceso se realiza con la el Scanner de SonarQube pero en la siguiente ruta **>Administrar Jenkins>configuración de herramientas globales>SonarQube Scanner>Añadir SonarQube Scanner** con el nombre de sonar como identificador.

Imagen 27 configuración de Maven en Jenkins

h) Para terminar con el proceso de configuración solo queda vincular el servidor de SonarQube con el de Jenkins para esto, ubicar la siguiente ruta **>administrar Jenkins>configurar el sistemas>SonarQube servers> add SonarQube,** ingresando al anterior sitio de forma adecuada se genera un formulario con el nombre, URL del servidor y una autenticación, se llenan los datos al igual que en la imagen 28, exceptuando la autenticación debido a que es un proceso que se realiza desde la plataforma de SonarQube.

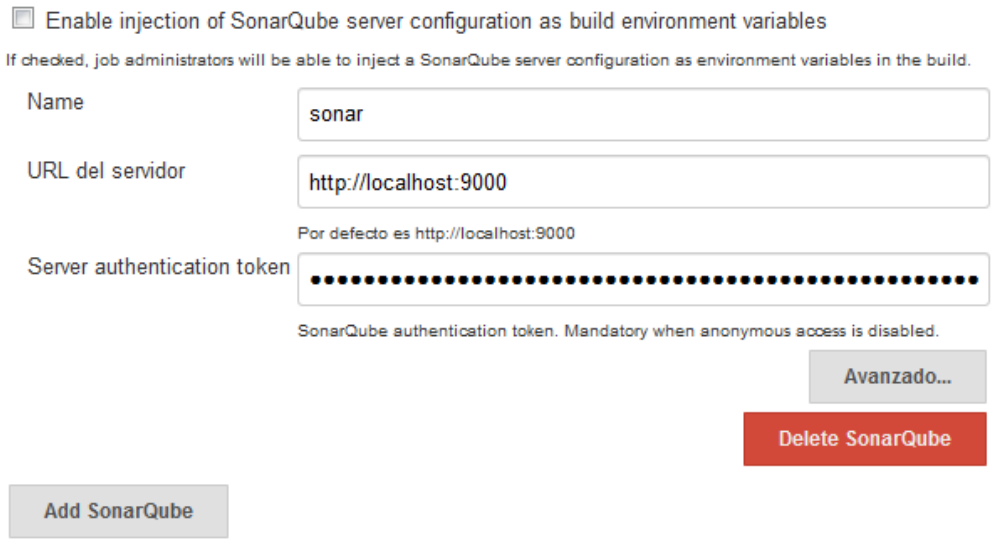


Imagen 28 vincular SonarQube con Jenkins

Para poder obtener este Token, se accede a la plataforma de SonarQube como administrador, en la parte superior derecha en el botón “UNA” buscar las siguientes instrucción **>mi cuenta >seguridad>generar nuevo**

Token , en esta parte asignar un nombre, preferiblemente algo relacionado con el tema (Jenkins) y por ultimo dar clic en el botón “generar”, dando como resultado lo mostrado por la imagen 29, al terminar el proceso se obtiene un código en letra de color verde este será el que se copia y pega en la autenticación del servidor del paso anterior.(dar guardar la plataforma Jenkins)

Imagen 29 Token SonarQube

3.4. Scanner y análisis de métricas.

Teniendo las anteriores herramientas configuradas de manera adecuada, se procede a realizar el montaje de proyectos para sus respectivos escaneos y posteriormente su análisis, para entender mejor la secuencia y la relación entre las diversas herramientas se plantea la imagen 30

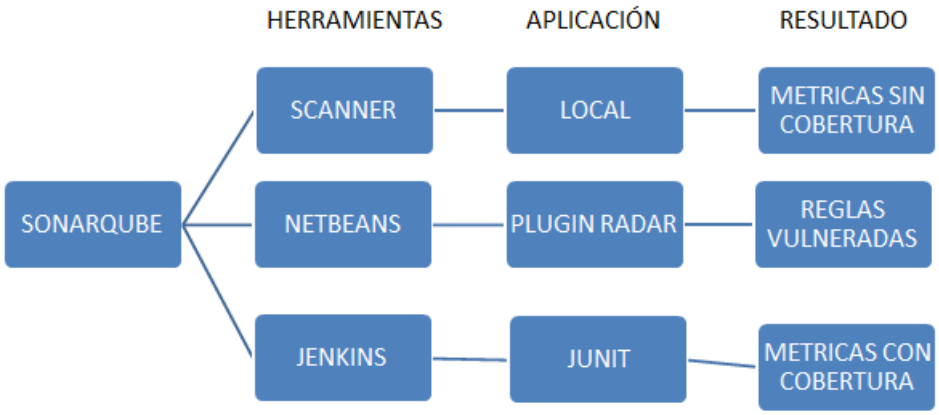


Imagen 30 relación entre herramientas y aplicación

Donde la primera parte se maneja a partir del scanner propio de SonarQube, éste se ejecuta de manera local y sus resultados son las métricas de errores y deuda técnica pero sin porcentaje de cobertura, la segunda parte se maneja el

scanner de SonarQube implementado en el plugins “Radar” instalado en el IDE de NetBeans, al efectuar el scanner arroja como resultado únicamente las reglas vulneradas durante en proceso de desarrollo del proyecto y la última parte realiza el escáner dentro del devops de Jenkins incluyendo el examen de pruebas unitarias realizadas en JUNIT este último scanner arroja como resultados las métricas de las reglas vulneradas en SonarQube incorporando los porcentajes de cobertura.

Para realizar escáner se utilizaron dos proyectos uno de creación propia en lenguaje java el cual se puede encontrar en el repositorio https://github.com/jhonson2jt/Tgd, este proyecto será la base de análisis de los diferentes métodos de scanner.

El proyecto llamado TGD es una creación básica pero que satisface las necesidades para ser analizada en los scanner, TGD se crea pensando en instituciones educativas, donde solicita la cantidad de estudiantes, el código de cada uno de ellos y 3 notas por cada uno con rango de 0 a 5 , a raíz de ellos genera una no nota definitiva promediada, también muestra la nota definitiva mayor y la menor, cuenta con una opción de búsqueda de notas por estudiantes usando el criterio del código, y otra opción que deja ver las notas y definitivas de todos los estudiantes, incluido en un menú de 4 opciones(mostrar notas por estudiante, nota mayor ,nota menor, y calcular promedio),este cuenta con su funcionalidad completa y de manera correcta, este proyecto cuenta con pruebas unitarias, a continuación se muestran los casos de uso según cada uno de las actividades planteadas anteriormente (ver imagen 31)

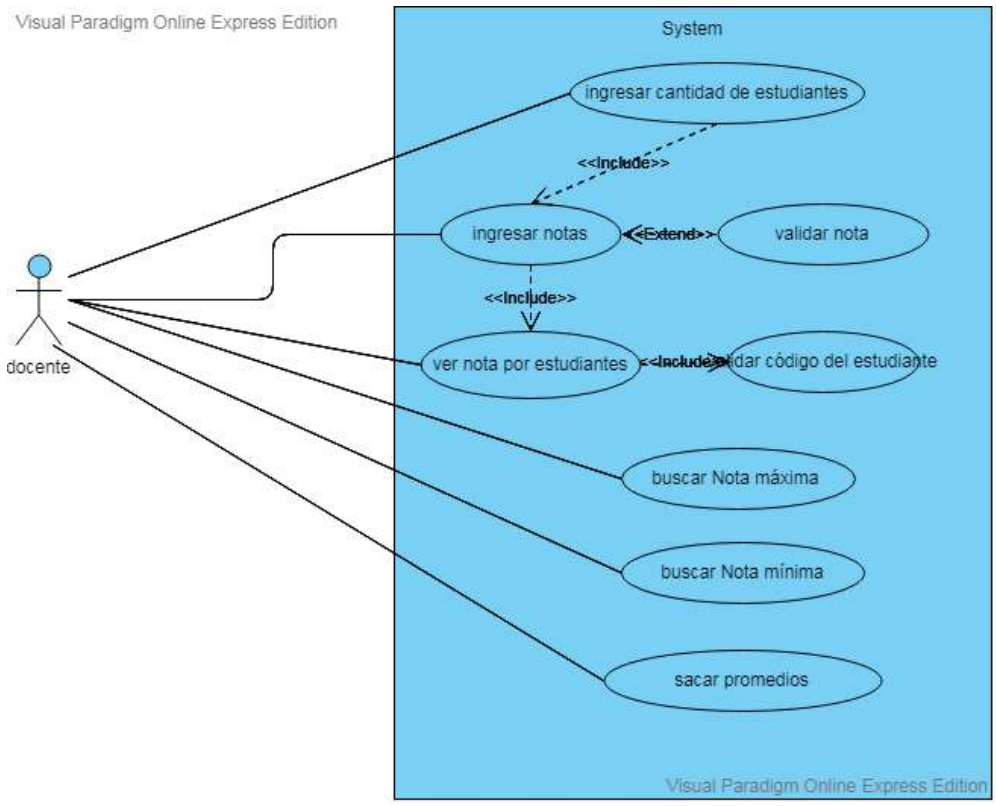


Imagen 31Casos de uso

Tabla 3 Caso de uno Ingresar cantidad de estudiantes

Caso de uso	Ingresar cantidad de estudiantes
Actores	Docente
Propósito	Capturar la cantidad de estudiantes
Resumen	El docente al ingresar al sistemas debe ingresar la cantidad de estudiantes del curso

Tabla 4 Caso de uso Ingresar notas

Caso de uso	Ingresar notas
Actores	Docente
Propósito	Capturar el código del estudiantes y 3 notas
Resumen	El docente ingresa el código de estudiantes y las tres notas de la materia una por una al terminar el proceso genera un menú

Tabla 5 Caso de uso Validar notas

Caso de uso	Validar notas
Actores	
Propósito	Validar que la nota ingresada sea valida
Resumen	El sistema valida que las notas ingresadas en el sistemas estén dentro de un rango de 0 a 5 en puntuación

Tabla 6 Caso de uso Ver nota por estudiante

Caso de uso	Ver nota por estudiante
Actores	Docente
Propósito	Capturar el código del estudiantes y mostrar en pantalla las 3 nota y la definitiva
Resumen	El docente ingresa el código de un estudiante el sistema muestra en pantalla las 3 notas y la definitiva del estudiante del código correspondiente.

Tabla 7 Caso de uso Validar código del estudiante

Caso de uso	Validar código del estudiante
Actores	
Propósito	Verifica que el código ingresado se encuentre registrado con anterioridad
Resumen	Al docente ingresar el código de un estudiante el sistema validad si este código fue registrado con anterioridad, de ser válido muestra en pantalla los 3 notas del estudiante y si definitiva

Tabla 8 Caso de uso Buscar nota máxima

Caso de uso	Buscar nota máxima
Actores	Docente
Propósito	Mostrar en pantalla la nota definitiva con mayor puntuación
Resumen	El sistema busca entre las notas definitivas de todos los estudiantes y muestra en pantalla aquella que sea la más alta, junto el código del estudiante y todas sus notas.

Tabla 9 Caso de uso Buscar nota mínima

Caso de uso	Buscar nota mínima
Actores	Docente
Propósito	Mostrar en pantalla la nota definitiva con menor puntuación
Resumen	El sistema busca entre las notas definitivas de todos los estudiantes y muestra en pantalla aquella que sea la más baja, junto el código del estudiante y todas sus notas.

Tabla 10 Caso de uso Sacar promedio

Caso de uso	Sacar promedio
Actores	Docente
Propósito	Saca el promedio entre las tres notas de cada uno de los estudiantes
Resumen	El sistema genera el promedio o nota definitiva a partir de las 3 notas ingresadas anteriormente, muestra en pantalla una matriz con el código de los estudiantes, las tres notas y la nota definitiva.

Tabla 11 Acciones del Actor y respuesta del sistema

Acción de los actores	**Respuesta del sistema**
1 El docente ejecuta el programa	2 Solicita la cantidad de estudiantes
3 El docente ingresa la cantidad de estudiantes que 5 tiene en el grupo	4 Solicita el código del estudiante junto con las tres notas, Valida que las notas estén dentro del rango de 0 a 5,
	6 Muestra en pantalla un menú. De 4 opciones.
7 Selecciona la primer opción “ver notas por estudiante”	8 Solicita ingresar el código del estudiante. luego válida que el código este registrado

	y muestra en pantalla las notas referentes a este código. Al terminar regresa al menú principal
9 Selecciona la segunda opción “Buscar nota mayor	10 Busca entre todas las notas definitivas y muestra aquella que sea la mayor. El código del estudiante y todas sus notas, al terminar regresa el menú principal
11 Selecciona la tercer opción “Buscar nota menor”	12 Busca entre todas las notas definitivas y muestra aquella que sea la menor. El código del estudiante y todas sus notas, al terminar regresa el menú principal
13 Selecciona la cuarta opción “sacar promedio	14 El sistema saca el promedio de las notas de cada uno de los estudiantes y muestra en pantalla todas las notas, la nota definitiva y el código del estudiante al terminar regresa a el menú de opciones
15 Selecciona la última opción “salir”	16 Finaliza el programa

Casos alternativos:

Línea 4: en caso de que la nota no sea válida, genera un error indicando el rango de las notas apropiadas (0 a 5), posteriormente solicita nuevamente ingresar la nota solamente en el lugar donde encontró el error.

Línea 8 en caso de que no sea válido el código del estudiante genera un error, dando la observación de que el estudiante no se encuentra y regresando automáticamente al menú principal

General: en el caso de acceder a una opción no validad del menú genera un error “opción no válida”, regresando al menú principal.

En cuanto al proyecto GYM no se cuenta con especificación de casos de uso, debido a que solo se tomara como muestra de comparación y aclaraciones de reglas y correcciones de errores que posiblemente el proyecto TGD no

contenga, Este se encuentra en el repositorio https://github.com/jhonson2jt/gym.

3.4.1. Scanner general y local de SonarQube

Por el momento lo único que se ha hecho es subir el servicio de SonarQube y acceder al servidor, pero falta lo más importante, el scanner al proyecto, se usó para este caso un código de creación propia en lenguaje java, solo para determinar el funcionamiento del scanner, teniendo en cuenta que el scanner local omite la evaluación de pruebas unitarias.

a. Lo primero que se hace es agregar la carpeta del proyecto en el directorio **C:\sonar-scanner-3.2.0.1227-windows\bin**, el proyecto agregado se llama TGD.
b. Ahora dentro de la carpeta "**TGD**" se creara un archivo llamado **sonar-project.properties** que contiene la configuración de la imagen 32.

```
# must be unique in a given SonarQube instance
sonar.projectKey=my:project
# this is the name and version displayed in the SonarQube UI. Was mandatory prior to SonarQube 6.1.
sonar.projectName=My project
sonar.projectVersion=1.0

# Path is relative to the sonar-project.properties file. Replace "\" by "/" on Windows.
# This property is optional if sonar.modules is set.
sonar.sources=./

# Encoding of the source code. Default is default system encoding
#sonar.sourceEncoding=UTF-8
```

Imagen 32: Archivo sonar-project .properties

El orden de esta configuración es dado por las líneas sin comentar partiendo con la llave del proyecto a ser escaneado el cual será el punto guía de la plataforma para ubicar dicho scanner, nombre de identificación del scanner, este nombre es el que dará SonarQube a su proceso, versión del escaneo en el caso de usarse múltiples versiones y ruta del archivo **.index** con respecto al **sonar-project .properties**

c. A continuación se ejecuta el scanner a través de la consola de comando, tomando como punto de partida la ruta del archivo **sonar-project .properties** de la siguiente manera **C:\sonar-scanner-3.2.0.1227-windows\bin** y se ejecuta el comando **sonar-scanner.bat**

d. Cuando termina este proceso arroja el mensaje que aparece en la imagen 33, diciendo que el ejecuto correctamente el proceso.

```
INFO: Task total time: 28.527 s
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
INFO: Total time: 31.481s
INFO: Final Memory: 12M/223M
INFO: ------------------------------------------------------------------------

C:\sonar-scanner-3.2.0.1227-windows\bin>
```

Imagen 33: Escaneo ejecución con éxito

e. Al terminar este proceso, es necesario dirigirse al panel de SonarQube donde se observa los resultados de una manera muy superficial del scanner tal como se muestra en la imagen 34.(es de aclarar que para mostrar la información del proyecto la primera vez es necesario esperar un tiempo mientras la plataforma realiza la aplicación de reglas)

Imagen 34: proyectos escaneados con éxitos

f. Si se desea tener una información más detalla basta solo con dar en el nombre del proyecto y despliega todo lo relacionado con el scanner (ver imagen 35).

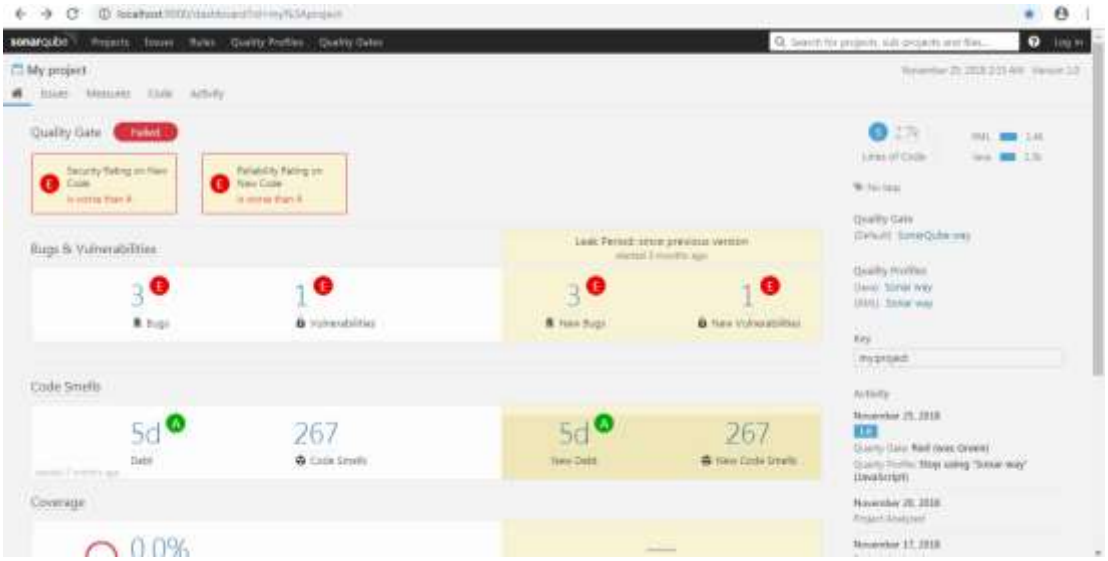


Imagen 35 métricas del scanner

g. Como se observó con anterioridad, este tipo de análisis arrogan como resultados métricas de relacionadas con mantenibilidad, confiabilidad y seguridad, así como lo muestra la imagen 36.

imagen 36 Resultado de Scanner

h. La imagen anterior da a conocer solo la métrica general, generada para la primera sección (confiablidad, seguridad y mantenibilidad) con valores del ranking de deuda técnica, y para los siguientes sección (cobertura y duplicaciones) en porcentaje dependiendo de las líneas de código presentes, en la siguiente sección con nomenclatura en k (miles) y por último, si el proyecto aprobó o no la puerta de calidad con dos posibles opciones fallida o pasada.

i. Para profundizar en las métricas basta con dar clic en el nombre del proyecto “Mi proyecto”, mostrando la ventana de la imagen 37.

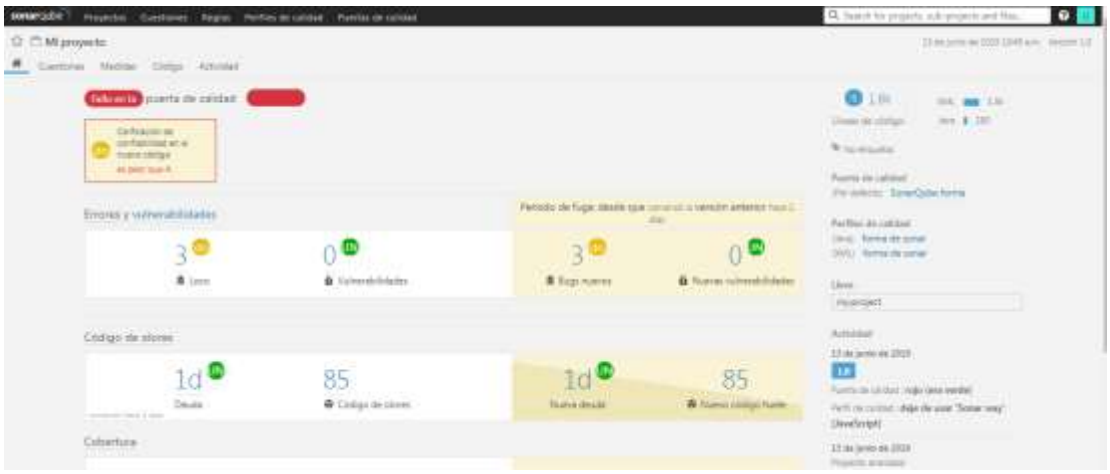


imagen 37 Métricas generales profundizadas

j. En la imagen se puede observar una división de cuatro secciones la primera de bugs y vulnerabilidades, el cual muestra la cantidad de errores tipo bugs que se han encontrado y el porcentaje de vulnerabilidad. La segunda sección corresponde a código smells o “código de olores”, donde se controla la deuda técnica y la cantidad de líneas de código que considera como código smell. La tercera sección equivale al porcentaje de cobertura a raíz de las pruebas unitarias. Y la última sección coincide con la duplicidad de código entregada en porcentaje y cantidad de bloques repetidos.

k. Además de esto, en la parte superior se encuentra un submenú con las opciones cuestiones, medidas, código y actividades; Al entrar a cuestiones este arroja todos los posibles errores y problemas encontrados en el escáner, (ver imagen 38)

Imagen 38 Listado de Bugs o errores encontrados

l. Al mismo tiempo se puede visualizar un menú de búsqueda basado en filtros donde se puede encontrar los problemas en dos ramas importantes según la cuestión, problema o según su esfuerzo de solución. Además puede buscar el problema según su tipo, resolución, gravedad, estado entre otras.

m. Cada uno de los problemas se encuentran muy bien detallados, muestra el nombre de la regla que está infringiendo, el tipo de error, la gravedad, el estado, el esfuerzo requerido para su solución y los comentarios a agregar, las anteriores se nombraron a tratarse de las más representativas.

n. Al escoger alguno de los problemas este nos remite directamente al archivo donde se encuentra el error como se observa en la imagen 39

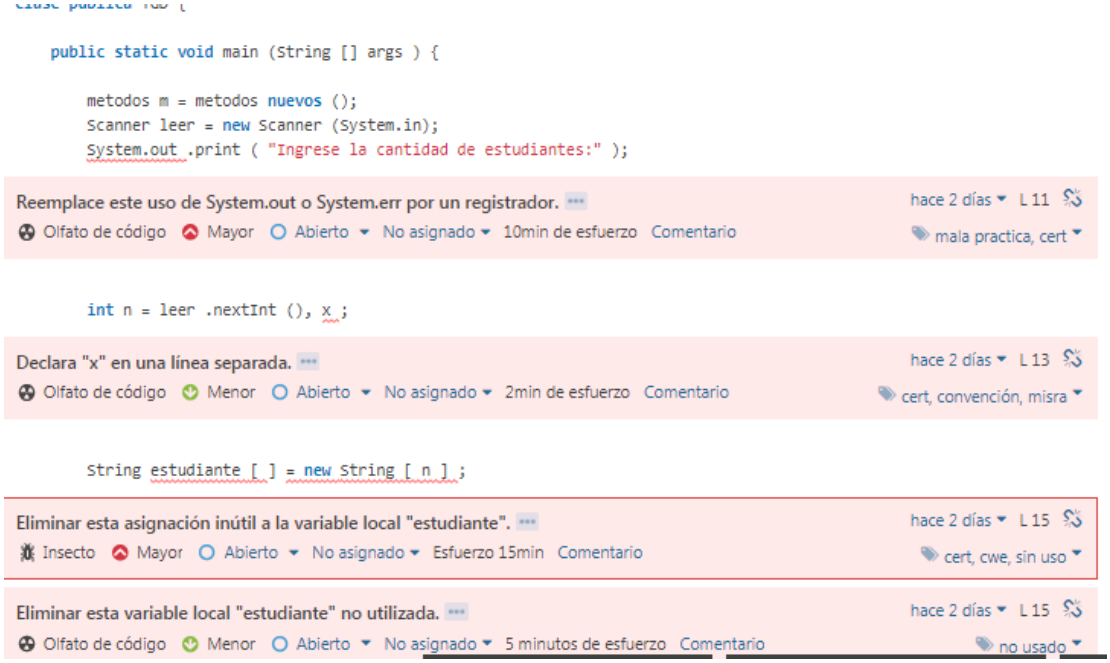


imagen 39 Errores mostrados en el código escaneado

o. Al Detallar el nombre de cada una de las reglas, se encuentran un icono de puntos suspensivos (imagen 40)

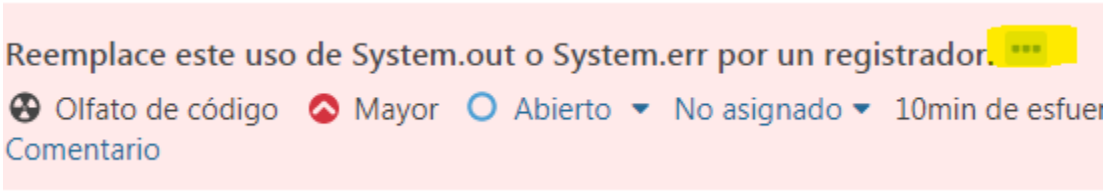


Imagen 40 Icono puntos suspensivos

p. Cuando se cliquea en estos puntos, en la parte inferior de la pantalla aparece la descripción del error, dos ejemplos uno con el error activo y

otro con la solución, y posibles excepciones que puede trabajar como lo detalla la imagen 41.

Imagen 41 Ejemplo de solución de errores

q. Pasando a la siguiente opción del menú encontramos “Medidas”, fundamentadas en las grandes categorías de evaluación: confiabilidad, seguridad, mantenibilidad, cobertura, duplicaciones, tamaño, complejidad y problemas.
r. Las 4 primeras opciones muestran de manera adicional un mapa cartesiano donde confrontan las líneas de código y tiempo de solución de cuestiones como se analiza en la imagen 42.

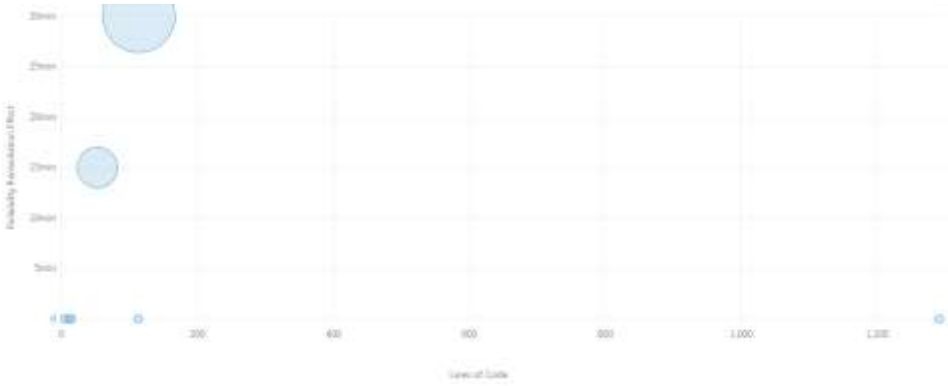


Imagen 42 Plano tiempo vs línea de código

s. Como tercera opción del menú principal se encuentra “código”, el cual muestra el código que se escaneo incluyendo todas las restricciones y errores, para una mejor ubicación y solución de problemas. Imagen 43.

imagen 43 Líneas de código vistas desde SonarQube

t. La última categoría que se encuentra es “actividad”, en la cual se evidencias las últimas acciones que se realizaron desde el último scanner, al realizar nuevamente el scanner y luego de haber corregido algún problema, aparecerá en este sector. (Ver imagen 44)

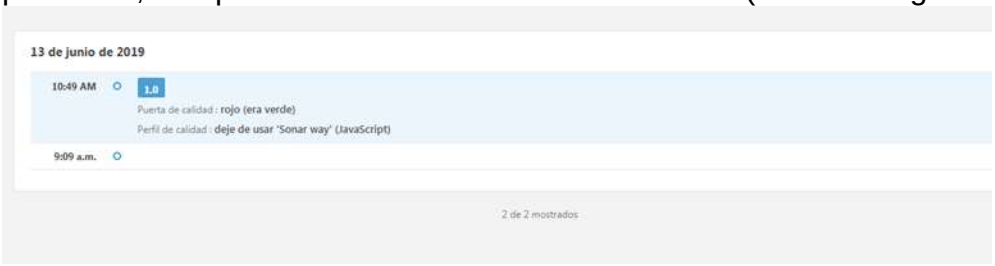


Imagen 44 Actividades realizadas

u. Teniendo claro la navegación dentro de la plataforma se procede a dar un análisis propio, basado en lo observado en éste procesos de escáner.

Observando el proyecto identificado como “my project” se observa que en primera instancia el proyecto no aprobó el scanner, esto se debe a que SonarQube genera las métricas de manera porcentual basándose en la cantidad de líneas de código escaneadas y la cantidad de reglas vulneradas, en este caso considera que encontró demasiados errores en las líneas de código, relacionadas con mantenibilidad, entre los errores encontrados se observan variables con asignaciones innecesarias, variables no usadas, variables declaradas consecutivamente, y la cual se considera la que genero la falla en el scanner, el uso de la sentencia “**System.out**”, debido que debido que el mensaje que arroja la plataforma es cito

“al registrar un mensaje hay varios requisitos importantes que deben cumplirse

- *El usuario debe ser capaz de recuperar fácilmente los registros*
- *El formato de todos los mensajes registrados debe ser uniforme para que el usuario pueda leer fácilmente el registro.*
- *Los datos confidenciales solo deben ser registrados de forma segura*

Si un programa escribe directamente en las salidas estándar, no hay absolutamente ninguna manera de cumplir con esos requisitos. Es por eso que es altamente recomendable definir y usar un registrador dedicado". (Documentación del error dentro de la plataforma),

SonarQube aconseja cambiar este modo de registrar el mensaje por un "**logger."** dado que este guarda los registros enviados. Concluyendo que al tener todas las salidas del programa con la sentencia mencionada al inicio cada una de estas genera un code smell, logrando un total de 39 errores de este tipo.

Por otro lado la plataforma da a entender que los bugs implican una mayor importancia que los códigos smell, esto se evidencia al dar solución a los 3 bugs presentados en el scanner, y al volver a ejecutar el scanner cambia de estado de fallida a pasada, modificando su valor de confiabilidad (Ver imagen 45)

Imagen 45 Calidad aprobada

A continuación se muestra el análisis del mismo proyecto pero esta vez aplicándolo en el IDE de NetBeans bajo el plugins "Radar", posteriormente se analiza y se compara con el scanner local realizado anteriormente en SonarQube.

Cabe aclarar que el proceso de escáner de este IDE no genera métricas, únicamente muestra las reglas vulneradas por el proyecto listadas por su nivel de severidad.

3.4.2 Scanner sonar radar

a. Se realiza el scanner desde NetBeans, recordar que antes de realizarlo es necesario tener activo el servicio de SonarQube scanner.
b. Para este caso usaremos el mismo proyecto usado con anterioridad llamado "TGD". Damos clic derecho en el proyecto y al desplegar un

menú en la parte inferior aparece un ítem especial para SonarQube, dar en “get Issues whit Sonar Runner”, mostrado en la figura 46.

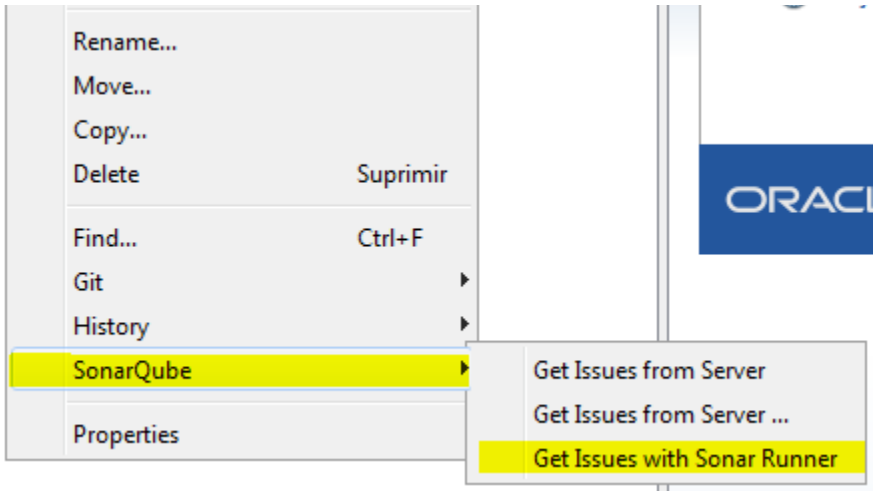


Imagen 46 inicio escaneo con radar

c. Aquí empieza el proceso de escaneo automáticamente, cargando las reglas usadas por “Sonar” este scanner toma el tiempo de ejecución a partir de las líneas de código del proyecto aproximadamente de 20 a 60 segundos para este caso, si el proceso termino satisfactoriamente muestra el análisis ejecutado correctamente y los errores encontrados por el escaneo teniendo en cuenta en grado de severidad, tal como se observa en la imagen 47.

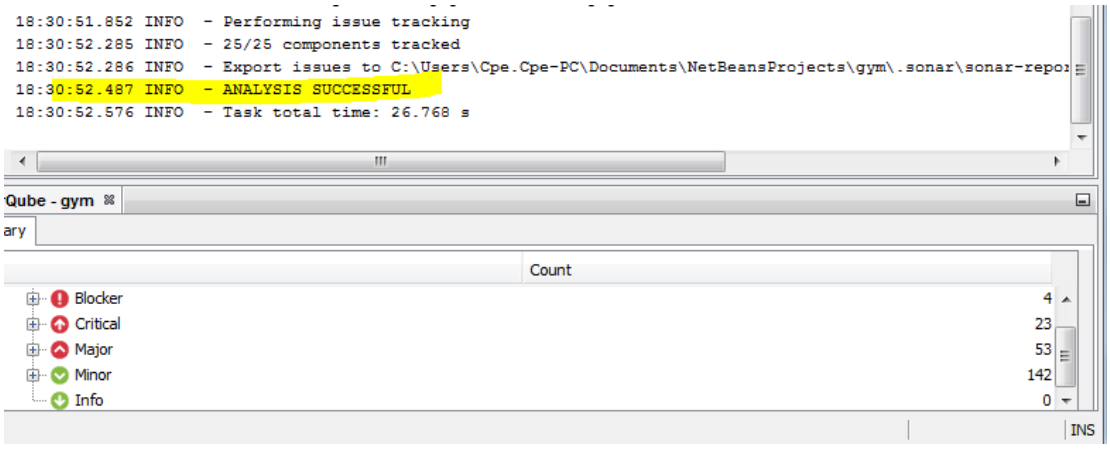
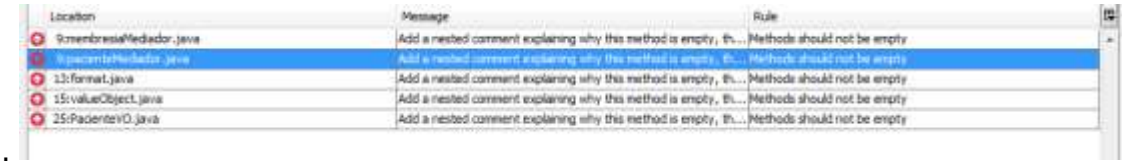


Imagen 47 final scanner radar

d. Al desplegar cada uno de los rangos de severidad, se observan las reglas vulneradas por el programador al realizar este proyecto como lo muestra la imagen 48

Imagen 48 Reglas grado de severidad vulnerada

e. Al dar doble clic en cualquiera de esta, muestra las diferentes incidencias que se obtuvieron con respecto a la regla y su problema, dando la localización, un mensaje de descripción del problema y la regla quebrantada(ver imagen 49)



.

Imagen 49 Detalles de error encontrado

f. Y por último al acceder a cualquiera de estas opciones, éste redirecciona directamente a la ubicación del error para una posible corrección, ver imagen 50.

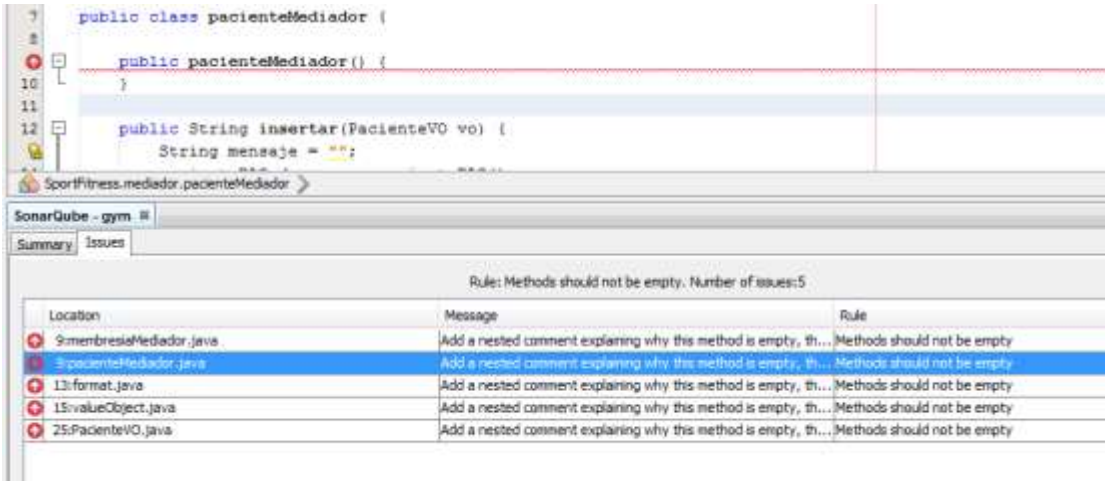


Imagen 50: ubicación del error en el proyecto

De la anterior manera se observa cómo también es posible realizar un scanner de SonarQube fuera de su plataforma convencional, es de aclarar que al realizar dicho scanner este no aparece en la página local de “sonar” debido a que arroja directamente las mediciones sobre el IDE.

Otra parte importante a destacar de este tipo de procedimiento es que solo aplica para las reglas de severidad anteriormente vistas, todo esto con el fin de generar una inspección continua sobre el cogido de una manera fácil dentro del mismo entorno de trabajo y en poco tiempo, para así ir mejorando el producto en cada uno de los entregables del ciclo de vida o metodología de desarrollo.

Se concluye que los dos métodos son valederos a la hora de realizar el scanner, debido a que generan exactamente las mismas vulnerabilidades de las reglas en los dos casos, se debe tener en cuenta que esta versión del plugins está acorde a la versión 6.4, puede presentarse el caso de realizar el scanner en una versión más avanzada de SonarQube donde se muestren más mediciones de reglas en la plataforma que en el IDE de NetBeans, debido a la retroalimentación de reglas en cada una de las nuevas versión.

Para concluir con este capítulo a continuación, se plantea el proceso y análisis del scanner aplicado en el devops de Jenkins, con esto se busca dar solución a la problemática de cobertura la cual fue omitida en los dos procesos anteriores por tratarse de un procedimiento el cual exige el uso de este devops para evaluar pruebas unitarias.

3.4.3 Scanner Jenkins

a. Teniendo configurado jenkins, se crea una nueva tarea, en esta parte solicita el nombre y tipo del proyecto para este caso el nombre sera SONARQUBE y el tipo sera de estilo libre como lo muestra la imagen 51 y por ultimo clic en **OK**.

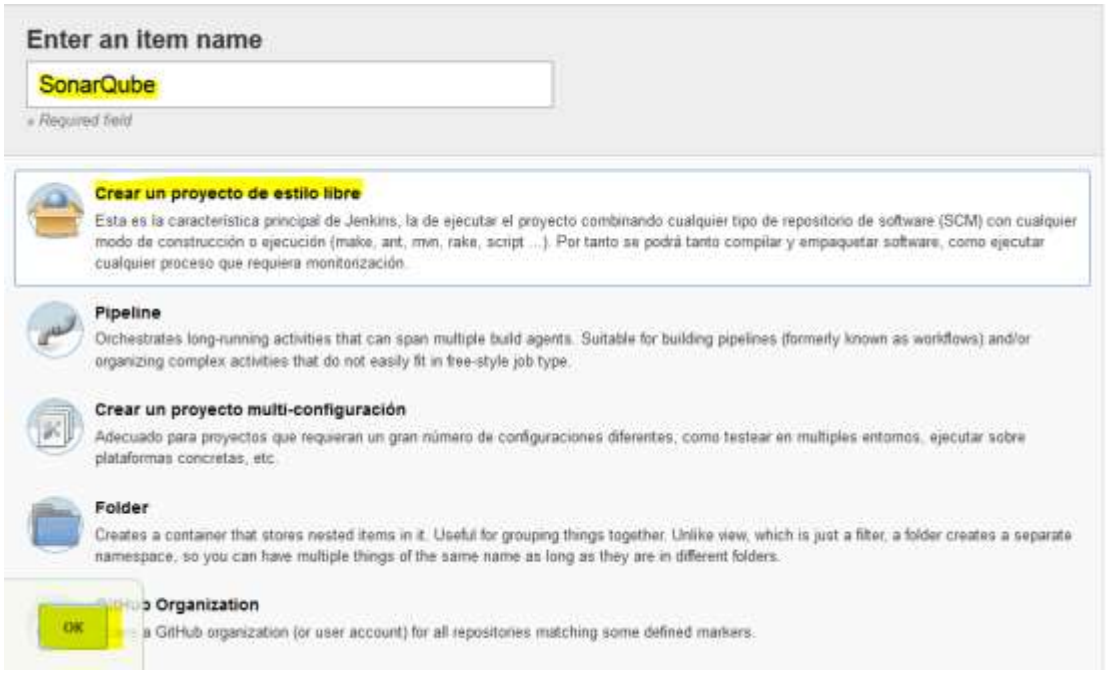


Imagen 51 Creación de proyecto en Jenkins

b. Al dar ok nos crea una ventana con diferentes tipos de configuraciones, las primordiales para poder implementar el scanner son “configurar el origen del código fuente” y “ejecutar”, la primera hace relación al lugar de donde proviene el proyecto que se va a escanear y la segunda las herramientas a ejecutar para esta prueba son Maven como gestor de programación y el scanner de SonarQube.(ver imagen 52).

Imagen 52 Ventana de configuración de proyecto

c. Para el lugar de origen es necesario tener el proyecto guardado en un repositorio de GITHUB con anterioridad, teniendo en cuenta estos, la url del proyecto en el repositorio será el cual usaremos. Dicha URL es https://github.com/jhonson2jt/SonarQube , ahora en la opción de "configurar código fuente" seleccionar el ítem de GIT y pegar la url en el espacio indicado por la imagen 53 y dejar el resto de configuración por defecto.

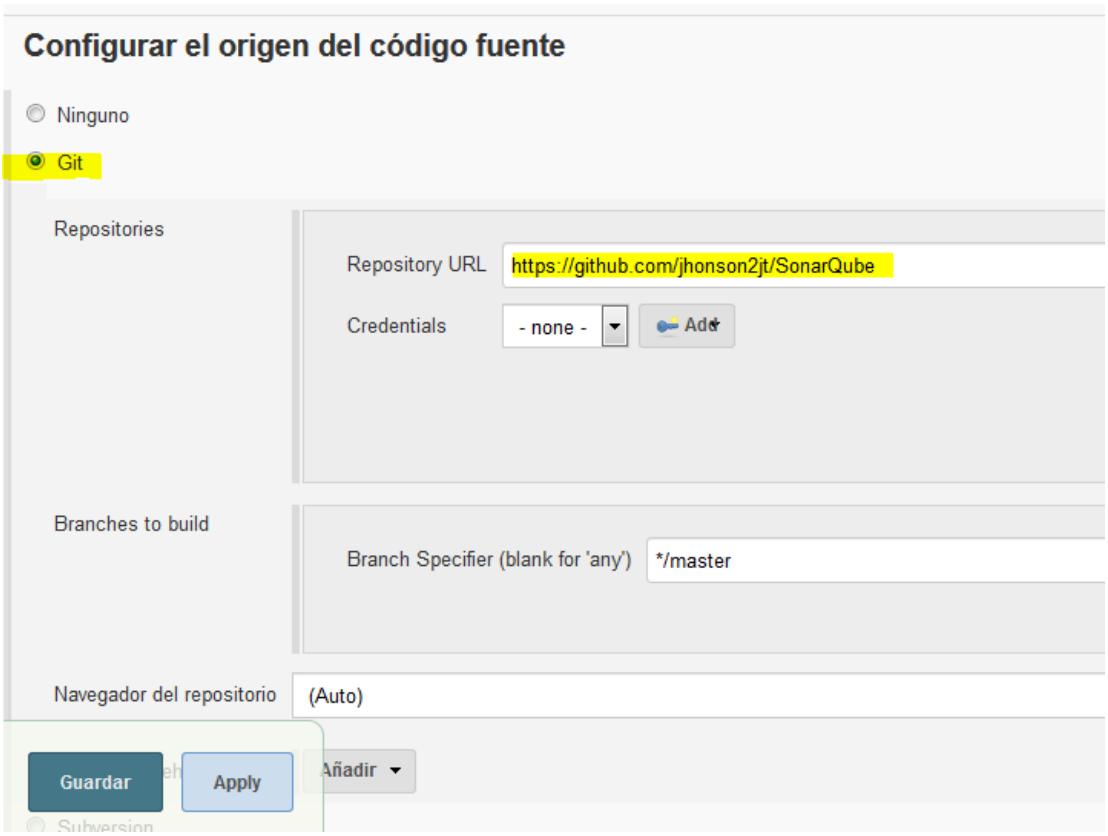


Imagen 53 vinculación GITHUB y Jenkins

d. Sin salir del la ventana de configuración dirigirse a la opción de "ejecutar" seleccionar el menú desplegable de "añadir un nuevo paso" y seleccionar la opción "ejecutar tareas 'Maven' de nivel superior" y colocamos en goals "install" por ultimo guardar.

e. Luego, muestra la configuración interna de cada proyecto creado, con las opciones de la imagen 54.

Imagen 54 configuración interna del proyecto.

f. Al tener esta configuración realizada se procede a incluir estos dos complementos dentro del proyecto creando anteriormente, para estos entramos al panel del proyecto dando clic en el nombre “SonaQube”, así como lo evidencia la imagen 55

.

Imagen 55 ingreso panel de configuración del proyecto

g. Estando ubicado en dicho panel seguir la siguiente ruta para adaptarlos complemento **>configurar>ejecutar>añadir nuevo un nuevo paso>scanner SonarQube**, solo basta con agregar y guardar paso, debido a que ya esta configurado, para Maven, este se había agregado con anterioridad, quedando como lo expone la imagen 56

Imagen 56 Complementos agregados

h. Ya con las configuraciones hechas anteriormente solo basta ejecutar las pruebas, para esto es necesario que en el proyecto se encuentre un archivo llamado pom.xml (incluido en GitHub) el cual descargara todos los complementos de sonar y Maven para vincular dichas plataformas, la opción construir ahora es la encargada de ejecutarlo.

i. Para verificar que si este bien configurado y vinculado Jenkins y SonarQube, basta con observar la parte inferior del menú del proyecto allí aparece el historial de tareas, en este lugar aparece el logo de SonarQube así como se evidencia en la imagen (57)

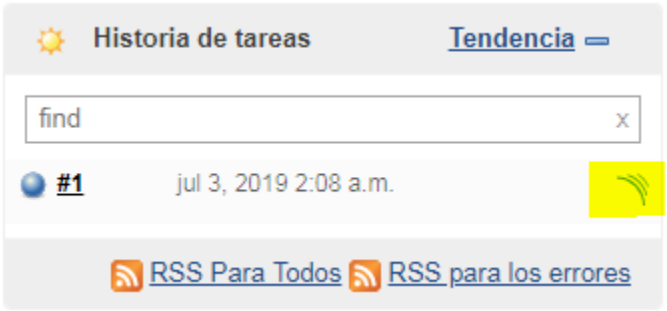


Imagen 57 Prueba de vinculación exitosa

j. Al realizar todo lo anterior de buena manera, se puede visualizar la información de cobertura en la plataforma de SonarQube, dando su porcentaje según las pruebas unitarias evaluadas.(ver imagen 58)

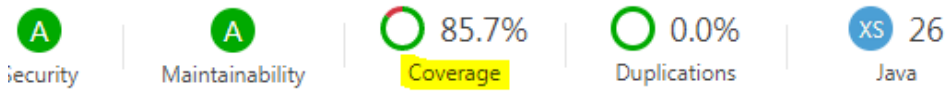


Imagen 58 implementación de cobertura en SonarQube

El análisis de las métricas arrojadas en este tipo de scanner se enfoca solamente en el porcentaje de cobertura, teniendo en cuenta que anteriormente se hizo el análisis sin evaluar las pruebas unitarias.

Las pruebas unitarias tienen a incrementar la deuda técnica al no cumplirse una cobertura en un 80%, esto viene dado por parámetros de cálculos de esta deuda, y afectara el tiempo el tiempo destinado a correcciones hasta que este porcentaje supere los 80.

Este porcentaje de cobertura es directamente proporcional a los métodos testados a mayor métodos del proyecto se prueben mayor será su nivel de cobertura e inversamente proporcional a la deuda técnica, a mayor cobertura menor deuda técnica.

Los anteriores ejemplos aplican para este trabajo, por su importancia para determinar la cobertura en la plataforma SonarQube, además demuestra una manera sencilla de detectar errores en las primeras etapas de desarrollo del ciclo de vida, previniendo posibles reestructuraciones o modificaciones por fallos en las entregas finales del proyecto las cuales generan pérdidas de recursos financieros y humanos.

Con la evaluación de pruebas unitarias y la cobertura del proyecto se da por finalizado el capítulo de análisis, configuración y escáner de la plataforma SonarQube, a continuación se presentan el paralelo entre la normativa ISO 25000 y SonarQube para determinar qué tan cercana es a ésta norma.

4. PARALELO ENTRE SONARQUBE E ISO25000

El objetivo de este paralelo es plasmar la relación que tiene la plataforma SonarQube con las características y subcaracterísticas de la ISO 25000, enfocada en el estudio y análisis de las métricas anteriormente expuestas en sus diferentes procesos de scanner, tomando como punto principal los tipos de errores y reglas vulneradas.

Tabla 12 Paralelo entre SonarQube e ISO25000

Característica ISO 25000	Integrado en SonarQube	Método de Aplicación
Funcionalidad	No	No aplica
Rendimiento	No	No aplica
Compatibilidad	No	No aplica
Usabilidad	No	No aplica
Fiabilidad **Madurez** **Tolerancia a fallos** **Recuperabilidad** **Conformidad**	Si	**Tolerancia a fallos**: Esta evaluada según la cantidad de Bugs encontrados, se muestra en grado de confiabilidad basado en el ranking de deuda técnica. **Recuperabilidad**: Se encuentra como recomendaciones en ciertas reglas basadas en el envió de datos. Ejemplo implementación del logger. **Conformidad**: categorización de reglas
Seguridad **Resistencia a la copia** **Resistencia al acceso** **Facilidad de cifrar** **Resistencia a la falsificación** **Robustez** **Conformidad**	Si	**Resistencia al acceso**: Generado principalmente por vulnerabilidades del sistema, basado principalmente en problemas de credenciales y posibles accesos prohibidos al código. Se encuentra como clasificación de seguridad, dado por ranking de deuda técnica. **Facilidad de cifrar**: se encuentra como

		recomendaciones y sugerencias en reglas de seguridad basadas en el tipo de cifrado.
Mantenibilidad **Capacidad de ser analizado** **Facilidad de cambio** **Estabilidad** **Facilidad de prueba** **Conformidad**	Si	**Cumple con todas las subcaracterísticas de mantenibilidad** Se basa principalmente en códigos smells y deuda técnica de la misma, se evidencia en problemas con funciones, variables, duplicación de código, y errores de coherencia de escritura, se encuentra como clasificación de mantenibilidad dada por ranking de deuda técnica
Portabilidad	No	No aplica

5. CONCLUSIONES

Al analizar y comparar las normativas de calidad de producto software ISO9126, e ISO25000 se entiende la importancia de éstas en el desarrollo software, así como sus aportes a la industria del software, abarcando como principal medida la reducción del déficit de calidad presente en los productos software presentes desde la década de los 70, en donde se implementa por primera vez la ISO 9126, basando sus características y subcaracterísticas en inconvenientes más comunes de la época, como la falta o fallas de funcionalidades, el que tan eficiente era el producto y facilidad de usar del mismo.

En cuanto a la normativa ISO25000 se encuentra nuevos términos y adaptaciones necesarias las cuales no estaban presentes en la ISO9126, estas adecuaciones se basan primordialmente en problemáticas que traen consigo los avances de la tecnología como inconvenientes de seguridad, portabilidad y desempeño existentes en la mayoría de los productos software desarrollados en la actualidad, está normativa busca generar un mejor manejo, comprensión y uso de cada las características y subcaracterísticas, depurando falencias halladas durante la aplicación y ejecución de la norma 9126 por empresas y grupos colaboradores de desarrollo.

Estas normativas (ISO 25000,ISO 9126) buscan aumentar la calidad de los productos teniendo en cuenta todos los posibles factores externos o internos involucrados en el desarrollo, etapas del ciclo de vida o ejecución del producto, algunos de estos factores son hardware, software y sistema operativo (compatibilidad), tiempo de ejecución apropiada y utilización de recursos(eficiencia), capacidad de ser modificado y probado (mantenibilidad), resistencia a accesos indeseados y robustez(seguridad) entre otros, de esta manera disminuir los gastos innecesarios de tiempo y dinero al tratar de dar solución a estos problemas.

Del uso de la plataforma SonarQube se infiere que sigue patrones de evaluación según estándares internacionales, esto se ve reflejado en la cantidad de reglas que maneja y sugerencias de solución al verse vulnerada cada una de estas.

Estudiando los resultado de los scanner se infiere que el uso de SonarQube debido a su facilidad de aplicación, uso y estudio de sus métricas puede ser incluido dentro del proceso de desarrollo o dentro de las diferentes etapas del ciclo de vida de un proyecto, facilitando a través de la deuda técnica un estimado del tiempo de resolución de problemas y de esta manera asegurar un porcentaje de calidad de producto software.

La inclusión de pruebas unitarias en este tipo de procesos se considera necesaria para generar los porcentajes de cobertura, dando aprobación a la subcaracterística de mantenibilidad “capacidad de ser probado”, se concluye que este tipo de subcaracterística genera un respaldo lógico a los programadores al estar basado en

pruebas unitarias y al mantener un porcentaje de aprobación alto en este aspecto, un 80% asegurando que la cobertura sea lo suficiente en la mayoría del proyecto

Analizando los modos de scanner de SonarQube se concluye que aplica procesos solamente a lo relacionado con la parte programable del proyecto (código fuente) como lo son fiabilidad, mantenibilidad y seguridad, dejando de lado factores externos como son el caso de la funcionabilidad rendimiento, compatibilidad, usabilidad, Operabilidad y portabilidad, esto implica que omite actores del sistema, características del software y hardware, manejo y uso de los proyectos.

Para culminar se con el análisis de la plataforma se concluye que no sigue en su totalidad los factores de estudio de las normativa ISO25000 adoptando solamente 3 de sus 8 característica, pero sus métodos de scanner son muy estricto a la hora de evaluar software, generando un grado de confianza alto reflejado en sus métricas, dando a entender que se puede incluir en cualquier tipo de proyecto software y obtener bueno resultados.

6. RECOMENDACIONES Y TRABAJOS FUTUROS

- Se recomienda realizar análisis de la plataforma SonarQube con uso de bases de datos distribuidas y en aplicativos con enfoque web.
- Debido al poco tiempo de uso de Herramientas tipo app para celulares es aconsejable realizar un estudio y análisis de diversas herramientas en la plataforma SonarQube para generar conocimiento enfocado a la calidad ya que es muy poco documentado.
- Otra herramienta que realiza un proceso parecido al de la plataforma SonarQube es kiuwan, se plantea un posible estudio generando un paralelo y análisis entre estas dos

7. BIBLIOGRAFÍA

- INTERNATIONAL STANDARD ISO/IEC/IEEE, A. A. (2017). Standards ISO. Recuperado de http://standards.iso.org/ittf/PubliclyAvailableStandards/c071952_ISO_IEC_IEEE_24765_2017.zip
- INTERNATIONAL STANDARD. ISO/IEC/IEEE, A. A. (2011a). ISO 25000. Recuperado de http://iso25000.com/
- The Standish Group, A. A. (2013). Instructional media + magic. Recuperado de https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/GENREF/S130301C.pdf.
- Orantes Jiménez, A. A. (2007). Biblioteca Virtual en Salud. Recuperado de http://www.bvs.hn/cu-2007/ponencias/CAL/CAL006.pdf
- Pilalonga, A. A. (2017).Modulo Evaluación RED. Recuperado de https://sites.google.com/site/moduloevaluacionred/modelo-mc-call
- Pilalonga, A. A. (2017).Modulo Evaluación RED. Recuperado de https://sites.google.com/site/moduloevaluacionred/modelo-de-calidad-boehm
- Ingeniería del software FURPS. A. A. (2008).Ingeniería del software. Recuperado de http://clases3gingsof.wikifoundry.com/page/FURPS
- Cataldi. A. (2000). Metodología de diseño, desarrollo y evaluación de software educativo (Tesis de maestría). Recuperado de http://recursosbiblioteca.utp.edu.co/tesisd/textoyanexos/0053L864e_anexo.pdf
- Largo y Marín, A. A. (2017).Guía técnica para la evaluación de software. Recuperado de https://jrvargas.files.wordpress.com/2009/03/guia_tecnica_para_evaluacion_de_software.pdf
- EcuRED, A. A. (2017). Norma ISO/IEC 14598. Recuperado de https://www.ecured.cu/Norma_ISO/IEC_14598
- Prada, A. A. (2013). Estándares, Métricas de Calidad Y Pruebas Del Software. Recuperado de http://evaluacion-software.blogspot.com/2013/03/
- INTERNATIONAL STANDARD. ISO/IEC/IEEE, A. A. (2011b). ISO 25000. Recuperado de: https://iso25000.com/index.php/normas-iso-25000?limit=4&limitstart=0
- VALENCIANO LÓPEZ. A. (2015). Auditoría mantenibilidad aplicaciones según LA ISO/IEC 2500 (Tesis de grado). Recuperado de: https://eprints.ucm.es/37485/1/AUDITOR%C3%8DA%20MANTENIBILIDAD%20APLICACIONES%20SEG%C3%9AN%20LA%20ISO_IEC%202500 0.pdf

- PMO informática, A. A. (sf). La oficina de proyectos de informática. Recuperado de http://www.pmoinformatica.com/p/pruebas-de-software.html
- APIUMHUB, A. A. (2017). Beneficios de las pruebas unitarias. Recuperado de https://apiumhub.com/es/tech-blog-barcelona/beneficios-de-las-pruebas-unitarias/
- Pressman, A. A. (2010). Ingeniería del software un enfoque práctico séptima edición. Recuperado de http://cotana.informatica.edu.bo/downloads/ld-Ingenieria.de.software.enfoque.practico.7ed.Pressman.PDF
- Tello. A. (2016). Evaluación de calidad de un producto de software (Tesis de grado). Recuperado de http://sedici.unlp.edu.ar/bitstream/handle/10915/58934/Documento_completo.pdf-PDFA.pdf?sequence=3
- Estévez A. (2014). Modelo de calidad para evaluar el software desarrollado en el centro de Investigación aplicada y desarrollo en tecnologías de información CIADTI (Tesis de maestría). Recuperado de http://www.unihorizonte.edu.co/revistas/index.php/TECKNE/article/download/129/125

- SonarQube. A. A. (sf). Acerca de SonarQube. Recuperado de https://www.sonarqube.org/about/
- Panel Testing. A. A. (2015). Calidad Software – Los 7 ejes de la calidad del código fuente. Recuperado de https://www.panel.es/blog/calidad-software-los-7-ejes-de-la-calidad-del-codigo-fuente/
- Ruiz. A. A. (2015). Qué es DevOps (y sobre todo qué no es DevOps). Recuperado de https://www.paradigmadigital.com/techbiz/que-es-devops-y-sobre-todo-que-no-es-devops/
- Guadin, A. A. (2018). Requerimientos. Recuperado de https://docs.sonarqube.org/display/SONARqube71/Requirements
- Universidad de los Andes. A. A. (2015). Métricas de calidad de código, Parte 2: Perfil de Calidad Recuperado de https://profesores.virtual.uniandes.edu.co/~isis2603/dokuwiki/lib/exe/fetch.php?media=principal:metricas-sonar-parte_2_perfil_de_calidad.pdf

- Borja. A. A. (2015). Configura e interpreta las métricas de SonarQube para conocer la calidad de tu código. Recuperado de https://www.adictosaltrabajo.com/2015/03/03/sonarqube-4-5-2/
- NetBeans A. A. (sf). NetBeans. Recuperado de https://netbeans.org/index_es.html
- Universidad de Alicante, A. A. (2014).Casos de prueba JUNIT. Recuperado de http://www.jtech.ua.es/j2ee/publico/lja-2012-13/sesion04-apuntes.html#Integraci%C3%B3n+de+JUnit+en+Eclipse

- Rodríguez y López, A. A. (sf). Pruebas de software. Recuperado de http://www.cc.uah.es/drg/docencia/Pruebas/Pruebas4x1.pdf
- García Oterino, A. A. (2015). ¿Qué es Jenkins. Recuperado de https://www.javiergarzas.com/2014/05/jenkins-en-menos-de-10-min.html
- López y Pedraza, A. A. (2017). La Objetividad en las Pruebas Estandarizadas. Recuperado de https://revistas.uam.es/index.php/riee/article/viewFile/7592/7891
- Wikipedia, A. A. (2018). Reproducibilidad y repetibilidad. Recuperado de https://es.wikipedia.org/wiki/Reproducibilidad_y_repetibilidad
- PMD, A. A. (2019). Analizador de código fuente PMD. Recuperado de https://pmd.github.io/
- FindBugs, A. A. (2015). Encuentra errores en los programas de Java. Recuperado de http://findbugs.sourceforge.net/ .
- Slidershare, A. A (2014). Calidad de software. Recuperado de https://es.slideshare.net/raaf0001/unidad-1calidad-del-software
- Borbón Ardila, A. A (2013). NORMA DE EVALUACIÓN ISO/IEC 9126. Recuperado http://actividadreconocimiento-301569-8.blogspot.com/2013/03/norma-de-evaluacion-isoiec-9126.html
- Camargo Manuel, A. A (2014) Aplicación de la calidad en el desarrollo del software. Recuperado de https://es.calameo.com/books/003651974633b7463b999
- Vegas Celso, A. A (2014). Ingeniería de software. Recuperado de https://slideplayer.es/slide/1087034/
- Marulanda López, A. A (2014). Aseguramiento de la calidad en el diseño del software. Recuperado de https://core.ac.uk/download/pdf/47246132.pdf.
- Betancur Luis, A. A(2014). Requisitos de calidad. Recuperado de https://es.scribd.com/document/163352809/Unidad-4
- Badenas Jesús A. A(2014). Nivel de cumplimiento de reglas. Recuperado de https://www.sonarqubehispano.org/display/DOC/Nivel+de+cumplimiento+de+reglas