# AUDITORÍA MANTENIBILIDAD APLICACIONES SEGÚN LA ISO/IEC 25000

JAVIER VALENCIANO LÓPEZ

FACULTAD DE INFORMÁTICA, UNIVERSIDAD COMPLUTENSE DE MADRID


TRABAJO FIN DE GRADO INGENIERÍA DEL SOFTWARE

JUNIO 2015

Directores:

Pedro García Repetto

M. Carmen Molina Prego

# Autorización de difusión y utilización

El abajo firmante, matriculado en el grado de ingeniería del software de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Grado: "AUDITORÍA MANTENIBILIDAD APLICACIONES SEGÚN LA ISO/IEC 25000", realizado durante el curso académico 2014-2015 bajo la dirección de Pedro García Repetto y María del Carmen Molina Prego, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Javier Valenciano López

Junio de 2015

# Índice

Autorización de difusión y utilización ..... ii

Índice de figuras ..... v

Índice de tablas ..... vii

Resumen ..... viii

Abstract ..... ix

1. Introducción ..... 11

1.1. Mantenibilidad ..... 11

1.2. Proceso Auditoría Mantenibilidad ..... 12

1.3. Objetivos/Finalidad TFG ..... 13

2. Fundamentos (análisis detallado de los antecedentes) ..... 14

2.1. ISO/IEC 9126 ..... 14

2.2. ISO/IEC 14598 ..... 17

2.3. ISO/IEC 25000 ..... 19

2.4. Calidad Software ..... 22

2.4.1. Dimensiones ..... 23

2.4.2. Deuda Técnica ..... 24

2.4.3. Dimensión Mantenibilidad ..... 25

2.4.4. Sub-Dimensiones Mantenibilidad ..... 27

2.4.4.1. Analizabilidad ..... 28

2.4.4.2. Modificabilidad ..... 28

2.4.4.3. Capacidad de ser probado ..... 30

2.4.4.4. Reusabilidad ..... 30

3. Proceso de auditoría ..... 33

3.1. Referente de mantenibilidad ..... 33

3.2. ISO 19011 e ISO/IEC 25040 ..... 48

4. Caso Práctico ..... 60

4.1 Analizabilidad ..... 64
Complejidad Ciclomática ..... 64
Densidad de código Repetido ..... 66
Densidad de Comentarios ..... 67
Densidad de defectos de la capacidad de ser analizado ..... 67
4.2 Modificabilidad ..... 71
Documentación ..... 71
Utilizar nombres con sentido ..... 72
Densidad de defectos de la capacidad de ser modificado ..... 72
4.3 Capacidad de ser probado ..... 76
Nombre de las variables ..... 76
Tamaño del programa ..... 76
4.4 Reusabilidad ..... 78
Estructura del programa ..... 78
Deuda técnica ..... 78
5. Conclusiones, principales aportaciones y líneas abiertas ..... 83
Conclusions ..... 86
Bibliografía ..... 87
Anexo 1: Tablas Resumen TextSecure ..... 89
Anexo 2: Laboratorio de Auditoría ..... 90
Anexo 3: Informe de Auditoría ..... 91

# Índice de figuras

Figura 1: Ciclo de vida Mantenibilidad ..... 12
Figura 2: Las tres perspectivas de calidad de ISO/IEC 9126 ..... 14
Figura 3: Características ISO/IEC 9126 ..... 15
Figura 4: Características de la vista en uso 9126 ..... 16
Figura 5: Estructura ISO/IEC 14598 ..... 17
Figura 6: Relación ISO/IEC 9126-14598 ..... 18
Figura 7: ISO/IEC 25000 ..... 19
Figura 8: Cuadrantes Deuda Técnica ..... 24
Figura 9: Mantenibilidad ISO/IEC 9126 ..... 27
Figura 10: Mantenibilidad ISO/IEC 25000 ..... 28
Figura 11: Responsabilidades software ..... 29
Figura 12: Acoplamiento software ..... 29
Figura 13: Cohesión software ..... 29
Figura 14: Diferencias ISO-25000/ISO_9126 ..... 32
Figura 15: Ejemplo Diagrama Flujo ..... 37
Figura 16: Ejemplo cálculo complejidad ciclomática ..... 38
Figura 17: Esquema Auditoría Mantenibilidad ..... 48
Figura 18: Actividades Auditoría Mantenibilidad ..... 53
Figura 19: Complejidad ciclomática TextSecure ..... 65
Figura 20: Complejidad ciclomática ArrayListCursor ..... 66
Figura 21: Código duplicado TextSecure ..... 66
Figura 22: Porcentaje de comentarios TextSecure ..... 67
Figura 23: Deuda técnica TextSecure ..... 69
Figura 24: Documentación TextSecure ..... 71
Figura 25: Ejemplo variables TextSecure ..... 72
Figura 26: Variable cc TextSecure ..... 72
Figura 27: Deuda técnica modificabilidad TextSecure ..... 73
Figura 28: Deuda técnica TextSecure ..... 74
Figura 29: Ejemplo nombre variables TextSecure ..... 76
Figura 30: Ejemplo 2 nombre variables TextSecure ..... 76
Figura 31: Tamaño TextSecure ..... 77
Figura 32: Tamaño TextSecure VerifySoft ..... 77

Figura 33: Deuda técnica modificabilidad TextSecure ..... 78
Figura 34: Deuda técnica TextSecure ..... 79
Figura 35: Pirámide deuda técnica TextSecure ..... 79
Figura 36: Deuda técnica VerifySoft ..... 80
Figura 37: Error excepción TextSecure ..... 81
Figura 38: Fases Auditoría Mantenibilidad ..... 84
Figura 39: Métricas Mantenibilidad ..... 84
Figura 40: Esquema ejes principales SonarQube ..... 90

# Índice de tablas

Tabla 1: Métricas mantenibilidad ..... 36
Tabla 2: Clasificación complejidad ciclomática ..... 39
Tabla 3: Clasificación defectos de la capacidad para ser analizado ..... 40
Tabla 4: Clasificación documentación publicada ..... 43
Tabla 5: Clasificación defectos de la modificabilidad ..... 44
Tabla 6: Clasificación deuda técnica ..... 46
Tabla 3: Clasificación defectos de la reusabilidad ..... 47
Tabla 8: Clasificación complejidad ciclomática TextSecure ..... 65
Tabla 9: Clasificación comentarios TextSecure ..... 68
Tabla 10: Clasificación complejidad ciclomática TextSecure ..... 68
Tabla 11: SQALE RATING sonarqube ..... 69
Tabla 12: Clasificación código repetido TextSecure ..... 70
Tabla 13: Resumen analizabilidad TextSecure ..... 70
Tabla 14: SQALE RATING sonarqube ..... 73
Tabla 15: SQALE RATING SonarQube ..... 74
Tabla 16: Clasificación código repetido TextSecure ..... 75
Tabla 17: Clasificación comentarios TextSecure ..... 75
Tabla 18: Resumen modificabilidad TextSecure ..... 75
Tabla 21: Clasificación capacidad de ser probado TextSecure ..... 77
Tabla 19: Clasificación deuda técnica VerifySoft ..... 80
Tabla 20: Clasificación estabilidad TextSecure ..... 81

## Resumen

Este trabajo de fin de grado tiene el objetivo de  proponer un proceso específico de auditoria de mantenibilidad para aplicaciones software. Actualmente se han desarrollado estándares para auditorías de calidad de aplicaciones pero no existe ningún proceso específico para medir la mantenibilidad de una aplicación.

Hoy en día existe una gran cantidad de software, pero no siempre los desarrolladores dan la importancia que se requiere a construir un software mantenible. Existen numerosos factores que impiden dar la importancia necesaria a la mantenibilidad del software. Esto aumenta la dificultad de que un componente software pueda ser modificado, mejorar su funcionamiento o adaptarse a cambios en el entorno.

En esta memoria, primero se realiza una introducción general del concepto de mantenibilidad en ella se describe el coste que supone desarrollar una aplicación poco mantenible. En el capítulo dos se incluye el estado del arte de la calidad software centrándose en la mantenibilidad de aplicaciones.

El capítulo tres constituye el núcleo principal del trabajo de investigación expuesto. En él se definen las métricas que van a ser utilizadas para medir la mantenibilidad de una aplicación y se propone un proceso detallado de auditoría de mantenibilidad.

En el capítulo 4 se realiza un caso práctico de auditoría de mantenibilidad, donde se audita una aplicación de dominio público en base al proceso de auditoría propuesto en este TFG.

Por último, en el capítulo cinco se presentan las principales conclusiones y aportaciones del presente trabajo. Así mismo, se proyectan las líneas abiertas de mejora que pueden ser tratadas en futuras investigaciones.



## Palabras Clave:

Aplicación, Auditoría, Calidad, Estándar, Mantenibilidad, Métricas, Software

## Abstract

The aim of this dissertation is to propose a specific audit process for the maintainability of software applications. Up to these days, standards for quality audits have been developed, however specific processes to measure the maintainability of an application still need to be improved.

Nowadays a great amount of software exists; nevertheless the importance of developing maintainable software is neglected by its developers. There several factors which lower the given importance to the maintainability of the software, and thus, the difficulty to modify software components are increased. In order to improve software functionality or to adapt to changes in the environment a high level of software maintainability should be required in all applications.

At the beginning of this memory a general introduction of the concept maintainability is accomplished, moreover, there is described the cost that involve developing a few maintainable application. Chapter two includes the state of the art for the quality software focusing on the maintainability.

Chapter three constitutes the main core of the dissertation exposed. Firstly, metrics that are going to be used to measure the maintainability of an application are described in detail. Secondly a full process for a maintainability audit is shown.

In chapter 4 a practical case of maintainability audit is accomplished. Here there is audited an application of public domain according to the process of audit proposed in this dissertation.

Finally, in chapter five the principal conclusions and contributions of the dissertation are commented. Likewise, there are outlined the lines opened of improvement that can be treated in future researches.



## KeyWords:

Application, Audit, Maintainability, Metrics, Quality, Standards, Software

# 1. Introducción

## 1.1.Mantenibilidad

La mantenibilidad es uno de los principales atributos de calidad de un sistema software, se conoce como una sub-dimensión de calidad y se evalúa durante la revisión del producto. Se define como “La facilidad con que un sistema software o componente puede ser modificado para corregir faltas, mejorar rendimiento u otros atributos, o adaptar a un entorno cambiante“, pero que se puede resumir como “fácil de entender y cambiar”.

El objetivo de auditar la mantenibilidad es definir un modelo para evaluar la mantenibilidad del software desde el punto de vista de la arquitectura global del sistema, facilitando el trabajo de localización de los componentes a modificar, y actualizar la documentación afectada por los cambios.

Una aplicación puede ser usable, rápida, eficiente, estética... todas ellas características relativas al estado o comportamiento actuales. Pero el software tiene una fuerte componente de evolución en el tiempo y ahí es donde entra la mantenibilidad.

Esta característica, que afecta enormemente al futuro de una aplicación, es obviada a menudo, a pesar de ser una de las que más diferencia el producto creado por un desarrollador aficionado del creado por uno profesional.

Si queremos adaptar la aplicación a una evolución del sistema operativo, a una nueva resolución de pantalla, añadir, cambiar o corregir una funcionalidad,... estamos poniendo a prueba la mantenibilidad del software.

Durante el desarrollo de una aplicación software se recomienda hacer las cosas sencillas, claras, que se puedan entender fácilmente.

## 1.2.Proceso Auditoría Mantenibilidad

A día de hoy existen estándares para auditar la calidad global de una aplicación software, sin embargo, apenas hay procesos para evaluar la mantenibilidad del software. Como se puede ver en la imagen, esta supone un 67% aproximadamente del tiempo invertido durante el ciclo de vida del software, es por ello que adquiere una gran importancia la creación de procesos que nos permitan evaluar y auditar esta característica de la mantenibilidad.

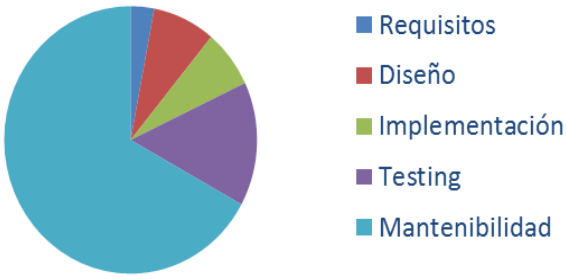


**Figura 1: Ciclo de vida Mantenibilidad**

La norma ISO/IEC 25040 que forma parte de la familia ISO/IEC 25000 propone un modelo de auditoría de calidad, a partir de este estándar podemos auditar la calidad global del software. Sin embargo esto no es suficiente dada la mencionada importancia que adquiere la característica de la mantenibilidad, por lo que es necesario dar una serie de pautas y acciones así como definir las métricas a evaluar cuando estamos auditando la mantenibilidad del software.

## 1.3. Objetivos/Finalidad TFG

Este trabajo de fin de grado tiene como objetivo proponer un modelo de auditoría de mantenibilidad para aplicaciones software. Para ello nos vamos a apoyar en estándares ya existentes como la norma ISO 19011 donde se propone un modelo general de auditoría y la ISO/IEC 25040 donde se proporcionan requisitos, recomendaciones y guías para llevar a cabo el proceso de evaluación de la calidad del producto software.

Además se aportan una serie de métricas divididas en cada sub-característica de la mantenibilidad (analizabilidad, modificabilidad, capacidad de ser probado y reusabilidad) y su función de medición. Algunas de estas métricas guardan relación con las publicadas en la tesis de Manuel Irrazábal "*Construcción de un Entorno para la Medición Automatizada de la Calidad de los Productos Software*", el resto son aportaciones propias.

Con estas métricas y el modelo propuesto se obtiene un detallado proceso de auditoría de mantenibilidad de aplicaciones.

Para completar la investigación, una vez detallado el modelo de auditoría, se va a auditar la mantenibilidad de la aplicación de software libre "TextSecure".

TextSecure es una aplicación para el intercambio privado de "SMS" que se puede descargar gratuitamente de los principales "markets" de aplicaciones.

Por último se exponen los resultados y conclusiones de este proyecto y se esbozan las líneas abiertas para futuras investigaciones.

# 2. Fundamentos (análisis detallado de los antecedentes)

En este punto se van a tratar los fundamentos que dan lugar a la medición de calidad del software, centrándonos en la dimensión de la mantenibilidad. Para ello es necesario hacer un repaso de los antecedentes históricos que han dado lugar a la ISO/IEC 25000 cuyo objetivo es medir la calidad del producto software.

## 2.1. ISO/IEC 9126

La ISO/IEC 9126 es un estándar internacional para la evaluación de la calidad del software. Actualmente se encuentra reemplazado por el proyecto SQuaRE (ISO 25000), el cual sigue los mismos conceptos.

Este estándar propone un modelo de calidad (ISO/IEC 9126-1) que se divide en 3 partes:

1. **Métricas internas (ISO/IEC 9126-3):** no dependen de la ejecución del software (medidas estáticas).
2. **Métricas externas (ISO/IEC 9126-2):** aplicables al software en ejecución.
3. **Métricas en uso (ISO/IEC 9126-4):** aplicables a la utilización del software por parte de los usuarios.

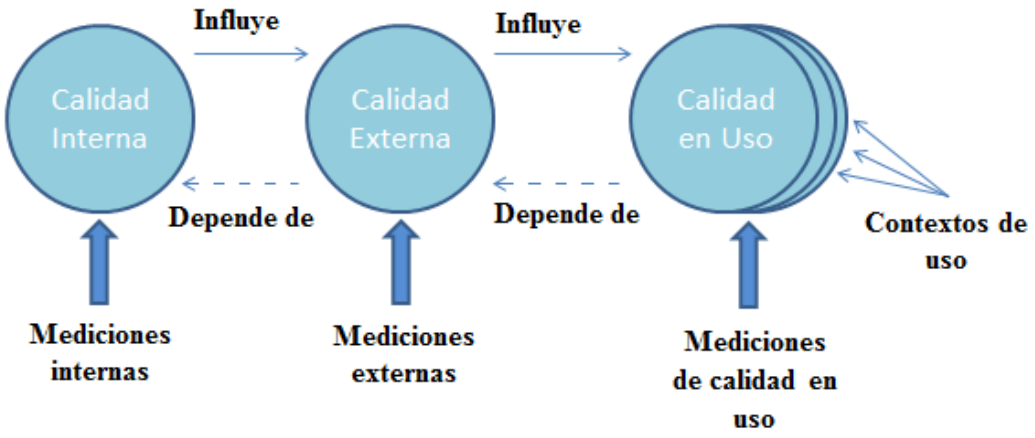


**Figura 2: Las tres perspectivas de calidad de ISO/IEC 9126**

Se necesita de un modelo de calidad bien definido para poder llevar a cabo la evaluación de calidad de un producto software. Un modelo de calidad está compuesto por características que se dividen en sub-características. Finalmente las Sub-características están compuestas por atributos, que obtienen sus valores de mediciones sobre el software.

En este caso el modelo de calidad según la ISO/IEC 9126-1 establece 10 características, seis de ellas comunes a las vistas internas y externas y las otras cuatro propias de las vistas en uso.


**Figura 3: Características comunes a las vistas internas y externas ISO/IEC 9126**

Las características comunes a las vistas internas y externas son las siguientes:

- **Funcionalidad**: capacidad del software de proveer los servicios necesarios para cumplir con los requisitos funcionales.
- **Fiabilidad**: capacidad del software de mantener las prestaciones requeridas del sistema, durante un tiempo establecido y bajo un conjunto de condiciones definidas.
- **Eficiencia**: relación entre las prestaciones del software y los requisitos necesarios para su utilización.

- **Usabilidad**: esfuerzo requerido por el usuario para utilizar el producto satisfactoriamente.
- **Mantenibilidad**: esfuerzo necesario para adaptarse a las nuevas especificaciones y requisitos del software.
- **Portabilidad**: capacidad del software para ser transferido de un entorno a otro.

Por otro lado, las cuatro características propias de la vista en uso son las siguientes:

- **Efectividad**: capacidad del software de facilitar al usuario alcanzar objetivos con precisión y completitud.
- **Productividad**: capacidad del software de permitir a los usuarios gastar la cantidad apropiada de recursos en relación a la efectividad obtenida.
- **Seguridad**: capacidad del software para cumplir con los niveles de riesgo permitidos tanto para posibles daños físicos como para posibles riesgos de datos.
- **Satisfacción**: capacidad del software de cumplir con las expectativas de los usuarios en un contexto determinado.

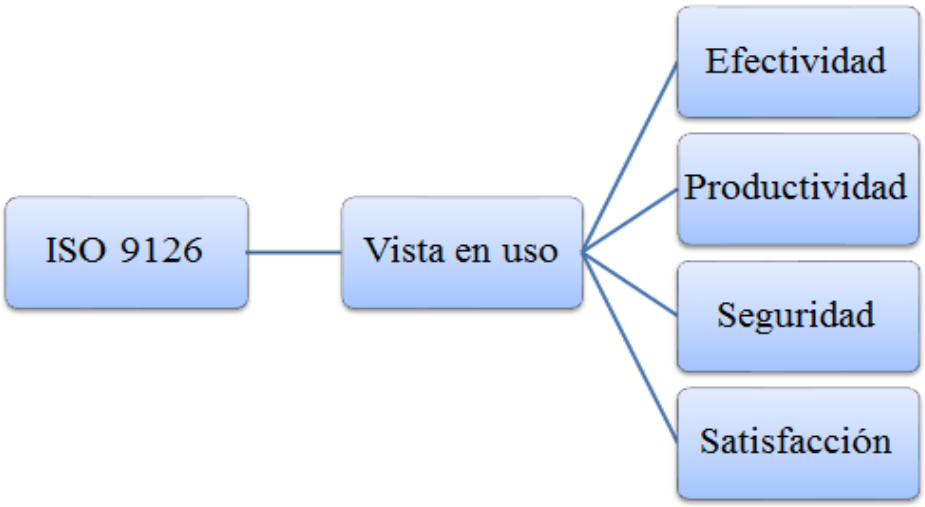


**Figura 4: Características de la vista en uso 9126**

## 2.2. ISO/IEC 14598

La norma ISO/IEC 14598 es un estándar que proporciona un marco de trabajo para evaluar la calidad de todo tipo de producto software e indica los requisitos para los métodos de medición y el proceso de evaluación.

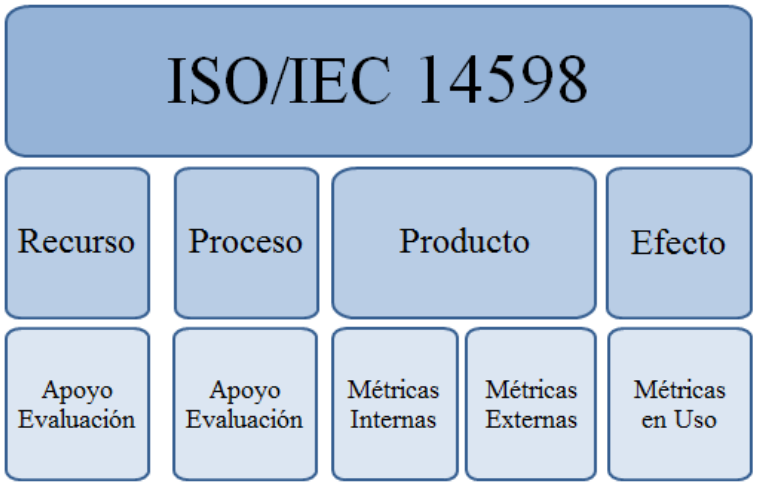


**Figura 5: Estructura ISO/IEC 14598**

La ISO/IEC 14598 proporciona métricas y requisitos para los procesos de evaluación, a través de 6 etapas.

1. **ISO/IEC 14598-1 Visión General**: establece un resumen de las otras cinco etapas, explica la relación entre la evaluación del producto software y el modelo de calidad.

   Actividades: (Establecer los requisitos de evaluación, Especificar la evaluación, Planear la evaluación, Ejecutar la evaluación).

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

   Actividades: (Preparación de políticas, Definición de objetivos, Identificación de la tecnología, Asignación de responsabilidades, Evaluación de software desarrollado y adquirido).

3. **ISO/IEC 14598-3 Proceso de desarrolladores**: lo utilizan las organizaciones que planean desarrollar un producto o mejorar uno existente, realiza evaluaciones de producto utilizando indicadores que puede predecir la calidad de los productos finales.

   Actividades: (Organización, Planeamiento, Especificaciones, Diseño, Montaje)

4. **ISO/IEC 14598-4 Proceso de comparadores:** lo utilizan las organizaciones que pretenden comparar o rehusar un producto de software existente, se aplica con el propósito de aceptación de un producto.

   Actividades: (Requerimientos, Especificación evaluación, Diseño evaluación, Ejecución evaluación).

5. **ISO/IEC 14598-5 Proceso evaluadores**: este proceso es utilizado por organizaciones encargadas de evaluar, provee los requisitos y guías para la evaluación del producto software.

   Actividades: (Trazabilidad, Resultados, Problemas, Mejoras, Conclusiones)

6. **ISO/IEC 14598-6 Modulo evaluación:** especifica las mediciones que van a ser tomadas sobre los atributos de calidad que se definieron en la etapa anterior, provee las guías para la documentación de la evaluación.

   Actividades: (Introducción, Alcance, Entradas, Resultados)

Se recomienda el uso conjunto de las normas ISO/IEC 9126 y ISO/IEC 14598. La norma ISO/IEC 9126 define un modelo de calidad de propósito general, describe un conjunto de características de calidad y brinda ejemplos de métricas. Mientras que la norma ISO/IEC 14598 da una descripción general de los procesos para la evaluación de productos de software así como también guías y requerimientos para la evaluación.

En el siguiente esquema se describe la relación entre estas dos normas.

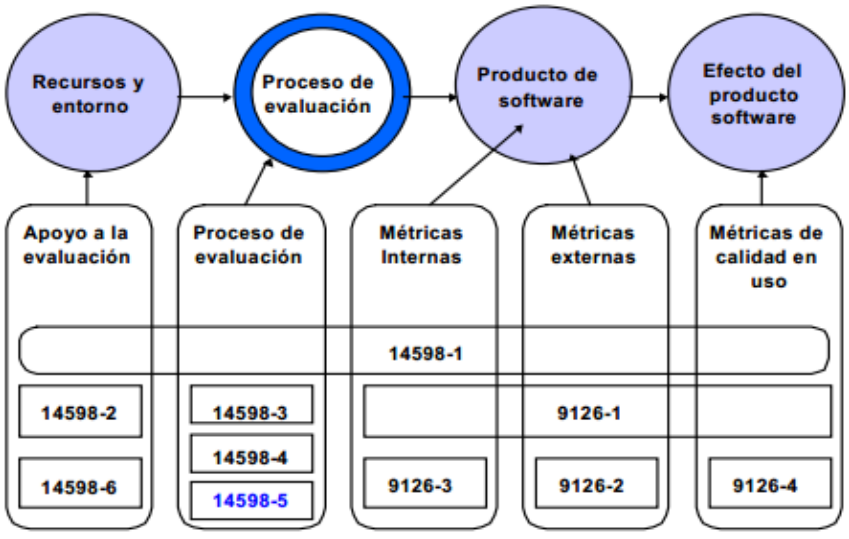


**Figura 6: Relación ISO/IEC 9126-14598**

## 2.3. ISO/IEC 25000

A lo largo de los últimos años se han elaborado trabajos de investigación, normas y estándares, con el objetivo de crear modelos, procesos y herramientas de evaluación de la calidad del propio producto software. Para dar respuesta a estas necesidades surge la nueva familia de normas ISO/IEC 25000 conocida como SQuaRE (Software Product Quality Requirements and Evaluation), que tiene por objetivo la creación de un marco de trabajo para definir los requisitos y evaluar la calidad del producto software, sustituyendo a las anteriores ISO/IEC 9126 e ISO/IEC 14598 y convirtiéndose así en el referente a seguir.

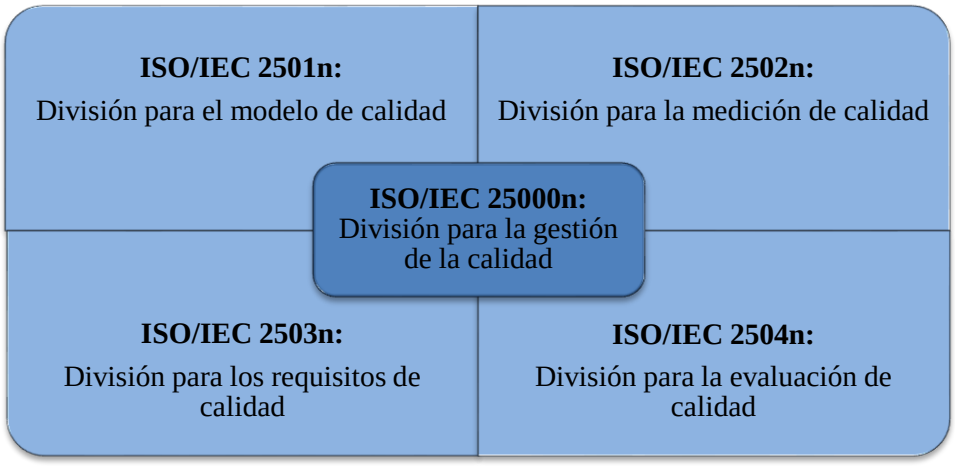


**Figura 7: ISO/IEC 25000**

La ISO/IEC 25000 se encuentra compuesta de varias partes o divisiones, entre las que podemos destacar:

- **La ISO/IEC 2500n-División de gestión de calidad:** las normas que forman este apartado definen todos los modelos, términos y definiciones referenciados por todas las otras normas de la familia 25000. Actualmente esta división de encuentra formada por:
    - ISO/IEC 25000, contiene el modelo de la arquitectura de SQuaRE, la terminología de la familia, un resumen de las partes, los usuarios previstos y las partes asociadas, así como los modelos de referencia.
    - ISO/IEC 25001, establece los requisitos y orientaciones para gestionar la evaluación y especificación de los requisitos del producto software.

- **La ISO/IEC 2501n-División de modelo de calidad**: las normas de este apartado presentan modelos de calidad detallados incluyendo las características para calidad interna, externa y en uso del producto software. Esta división está formada por:
    - ISO/IEC 25010, determina las características de calidad del producto software que se pueden evaluar. En total son 8 las características de calidad que identifica: funcionalidad, rendimiento, compatibilidad, usabilidad, fiabilidad, seguridad, mantenibilidad y portabilidad.
    - ISO/IEC 25012, 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.

- **La ISO/IEC 2502n-División de mediciones de calidad**. las normas pertenecientes a esta división incluyen un modelo de referencia de calidad del producto software, definiciones matemáticas para las mediciones de calidad y una guía práctica para su aplicación. Presenta aplicaciones de métricas para la calidad externa, interna y en uso del producto software. Actualmente esta división está formada por:
    - ISO/IEC 25020, 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, 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, define específicamente las métricas para realizar la medición de la calidad en uso del producto.
    - ISO/IEC 25023, define específicamente las métricas para realizar la medición de la calidad de productos y sistemas software.
    - ISO/IEC 25024, define específicamente las métricas para realizar la medición de la calidad de datos.

- **ISO/IEC 2503n-División de requisitos de calidad**. Las normas que forman este apartado ayudan a especificar requisitos de calidad. Está compuesto por :
    - ISO/IEC 25030, las normas que forman esta división ayudan a especificar los requisitos de calidad. Estos requisitos pueden ser usados en el proceso de especificación de requisitos de calidad para un producto software que va a ser desarrollado o como entrada para un proceso de evaluación.

- **La ISO/IEC 2504n-División de evaluación de calidad:** Incluye normas que proporcionan requisitos, recomendaciones y guías para llevar a cabo el proceso de evaluación del producto software. Esta división está formada por:
    - ISO/IEC 25040, define el proceso de evaluación de la calidad del producto software, compuesto por cinco actividades:
        - Establecer los requisitos: para determinar cuáles son los requisitos de calidad que se deben considerar a la hora de evaluar el producto.
        - Especificar la evaluación: indicando las métricas, criterios de medición y evaluación a tener en cuenta.
        - Diseñar la evaluación: definiendo el plan de actividades que se realizarán para evaluar el producto.
        - Ejecutar la evaluación: realizando las actividades de medición y evaluación del producto, considerando los criterios identificados en las fases previas.
        - Concluir la evaluación: elaborando el informe de evaluación y realizando la disposición de resultados e ítems de trabajo.
    - ISO/IEC 25041, 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.

## 2.4. Calidad Software

La calidad del producto software se puede interpretar como el grado en que dicho producto satisface los requisitos de sus usuarios aportando de esta manera un valor.

Hoy en día las aplicaciones software tienden a crecer, tanto en complejidad como en funcionalidad por lo que es necesario realizar un mantenimiento periódico para, por lo menos, conservar su calidad actual.

La calidad es un concepto complejo y multidimensional y puede ser descrita desde diferentes perspectivas.

- **Perspectiva trascendental:** donde la calidad es reconocida, pero no descrita. Se realizan definiciones subjetivas y no cuantificables.
- **Perspectiva del usuario:** la calidad está directamente relacionada con la satisfacción de las necesidades del usuario. Características como la fiabilidad, el rendimiento o la eficiencia son las tenidas en cuenta en esta perspectiva.
- **Perspectiva de la fabricación o del proceso:** se centra en la conformidad con las especificaciones y la capacidad para producir software de acuerdo con el proceso de desarrollo implementado en la empresa. En este caso se tienen en cuenta características como la tasa de defectos o los costes de re-trabajo.
- **Perspectiva de producto:** especifica que las características de calidad del producto se definen a partir de las características de sus partes. Por ejemplo líneas de código, complejidad, diseño.
- **Perspectiva basada en aspectos económicos:** miden la calidad de acuerdo a costes, precios, productividad, etc.

En el caso del desarrollo software, la calidad puede estudiarse desde el punto de vista de la calidad del producto software y la calidad del proceso de desarrollo software.

### 2.4.1. Dimensiones

La calidad software es un concepto muy amplio, que está compuesto por varias dimensiones.

Estas dimensiones según el modelo de calidad descrito en la **ISO/IEC 25010** son las siguientes:

- Adecuación Funcional: grado en que el producto proporciona las funciones que satisfacen las necesidades implícitas y explícitas cuando el producto se utiliza bajo determinadas condiciones.
- Eficiencia de desempeño: rendimiento relativo a la cantidad de recursos utilizados bajo determinadas condiciones.
- Compatibilidad: condición que hace que un programa y un sistema, arquitectura o aplicación logren comprenderse correctamente tanto directamente o indirectamente (mediante un algoritmo).
- Usabilidad: esfuerzo requerido por el usuario para utilizar el producto satisfactoriamente.
- Fiabilidad: capacidad del software de mantener las prestaciones requeridas del sistema, durante un tiempo establecido y bajo un conjunto de condiciones definidas.
- Seguridad: capacidad del software para cumplir con los niveles de riesgo permitidos tanto para posibles daños físicos como para posibles riesgos de datos.
- Mantenibilidad: esfuerzo necesario para adaptarse a las nuevas especificaciones y requisitos del software.
- Portabilidad: capacidad del software para ser transferido de un entorno a otro.

Entre estas dimensiones de la calidad software cabe destacar que dos de ellas están a su vez compuestas por otras Sub-Dimensiones. Estas son la Mantenibilidad y la Compatibilidad.

En el siguiente apartado nos centraremos en la dimensión de la mantenibilidad, ya que es la dimensión en la que se basa este TFG. Pero antes de ello resulta interesante introducir el concepto de deuda técnica, ya que está muy relacionado con la mantenibilidad de los productos software.

### 2.4.2. Deuda Técnica

La deuda técnica es el coste y los intereses a pagar por hacer mal las cosas. El sobre esfuerzo a pagar para mantener un producto software mal hecho, es decir es algo que resta valor al producto software y que, además, no se ve.

Con el tiempo, el término deuda técnica se ha perfeccionado y ampliado, principalmente por Steve McConnell con su taxonomía y Martin Fowler con sus cuatro cuadrantes.

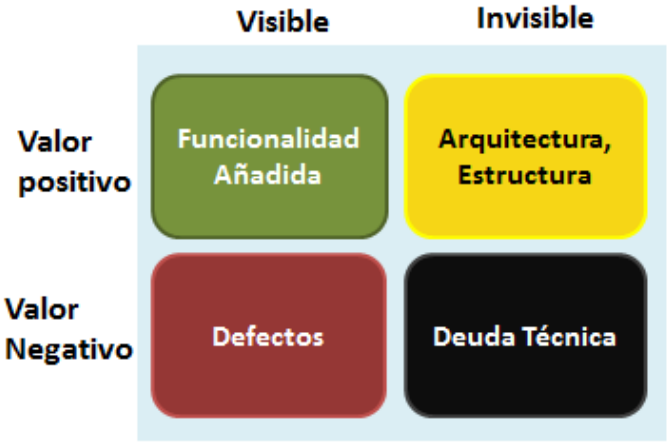


**Figura 8: Cuadrantes Deuda Técnica**

La figura muestra cuatro tipos de posibles mejoras o tareas a realizar en el futuro para aumentar el valor del producto software, como pueden ser ampliar funcionalidades (color verde, que es en lo que suelen fijarse las empresas), o invertir en arquitectura (amarillo), invertir reducir los defectos (rojo) o la deuda técnica (negro), que es invisible y tiene un efecto negativo.

- No hay deuda técnica, si hay retrasos, recortes, etc., que no requieren el pago de intereses. No todo el trabajo incompleto es deuda.
- Si hay deuda técnica puede ser:
  - Deuda incurrida involuntariamente debido a trabajos de baja calidad
  - Deuda incurrida intencionalmente.

La deuda técnica al final siempre alguien la paga. O la paga el proveedor que desarrolla el software o la paga el cliente que lo usa o compra. La principal razón de la deuda técnica es la presión de las fechas a la que están expuestos los desarrolladores, sin embargo, hay muchas otras causas, como la falta de cuidado, falta de educación, procesos pobres, la no verificación de la calidad, o la incompetencia.

### 2.4.3. 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.

Hoy en día la mantenibilidad es de las partes más importantes a la hora de llevar a cabo un desarrollo software, ya que es el atributo de calidad del software que más directamente influye en los costes y necesidades del mantenimiento, a mayor mantenibilidad menores costes de mantenimiento, y viceversa.

La mantenibilidad debe establecerse como objetivo en las fases iniciales del ciclo de vida para reducir las posteriores necesidades de mantenimiento.

Existen unos pocos factores que afectan directamente a la mantenibilidad, de forma que si alguno de ellos no se satisface adecuadamente, ésta se resiente.

Los tres más significativos son:

- **Proceso de desarrollo:** la mantenibilidad debe formar parte integral del proceso de desarrollo del software. Las técnicas utilizadas deben ser lo menos intrusivas posible con el software existente. Los problemas que surgen en muchas organizaciones de mantenimiento son de doble naturaleza: mejorar la mantenibilidad y convencer a los responsables de que la mayor ganancia se obtendrá únicamente cuando la mantenibilidad esté incorporada intrínsecamente en los productos software.
- **Documentación:** en múltiples ocasiones, ni la documentación ni las especificaciones de diseño están disponibles, y por tanto, los costes del mantenimiento del software se incrementan debido al tiempo requerido para que un mantenedor entienda el diseño del software antes de poder ponerse a modificarlo. Las decisiones sobre la documentación que debe desarrollarse son muy importantes cuando la responsabilidad del mantenimiento de un sistema se va a transferir a una organización nueva.
- **Comprensión de Programas:** la causa básica de la mayor parte de los altos costes del mantenimiento es la presencia de obstáculos a la comprensión humana de los programas y sistemas existentes.

La utilización de algunas prácticas de desarrollo (técnicas de programación estructurada, paquetes software estándares y generadores de listados) suponen la reducción del esfuerzo de mantenimiento ya que conllevan la generación de un código limpio y fácil de entender, mientras que otras prácticas (uso de generadores de código) implican un aumento de la necesidad de mantenimiento debido a que suelen generar código con una menor calidad.

La mantenibilidad se puede considerar como la combinación de dos propiedades diferentes: Reparabilidad y Flexibilidad.

Un sistema software es reparable si permite la corrección de sus defectos con una cantidad de trabajo limitada y razonable.

Un sistema software es flexible si permite cambios para que se satisfagan nuevos requerimientos.

.

### 2.4.4. Sub-Dimensiones Mantenibilidad

La mantenibilidad es un concepto muy amplio, que como hemos dicho anteriormente se puede dividir en distintas Sub-Dimensiones.

El concepto mantenibilidad software ha ido evolucionando, por ello la ISO/IEC 9126 distingue distintas sub-dimensiones para la Mantenibilidad con respecto al ISO/IEC 25010.

La mantenibilidad según el modelo de calidad recogido por la norma ISO/IEC 9126 está formada por las siguientes sub-características:

- **Analizabilidad**, facilidad para analizar el software en busca de deficiencias e identificar sus componentes y artefactos.
- **Cambiabilidad**, capacidad de permitir modificaciones en el producto software.
- **Estabilidad**, capacidad de evitar efectos inesperados tras la realización de modificaciones en el software.
- **Capacidad de ser probado**, capacidad para validar los cambios en el software.
- **Adherencia a las normas**, cumplimiento de los estándares y convenciones de mantenibilidad. Hace referencia a todas las anteriores.

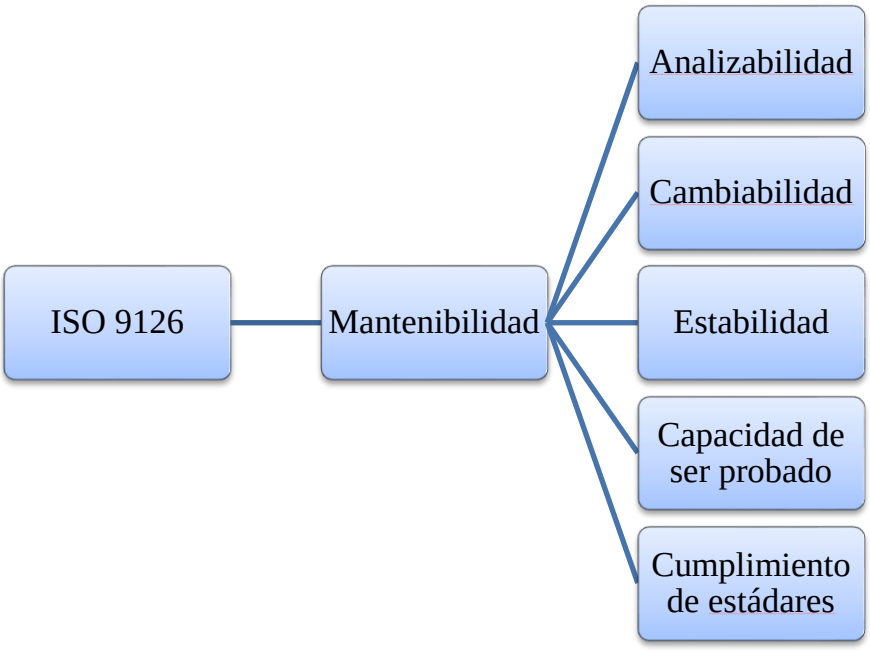


**Figura 9: Mantenibilidad ISO/IEC 9126**

Por otro lado con respecto a la ISO/IEC 25000 y en concreto la división 25010, distingue las siguientes sub dimensiones del concepto mantenibilidad:

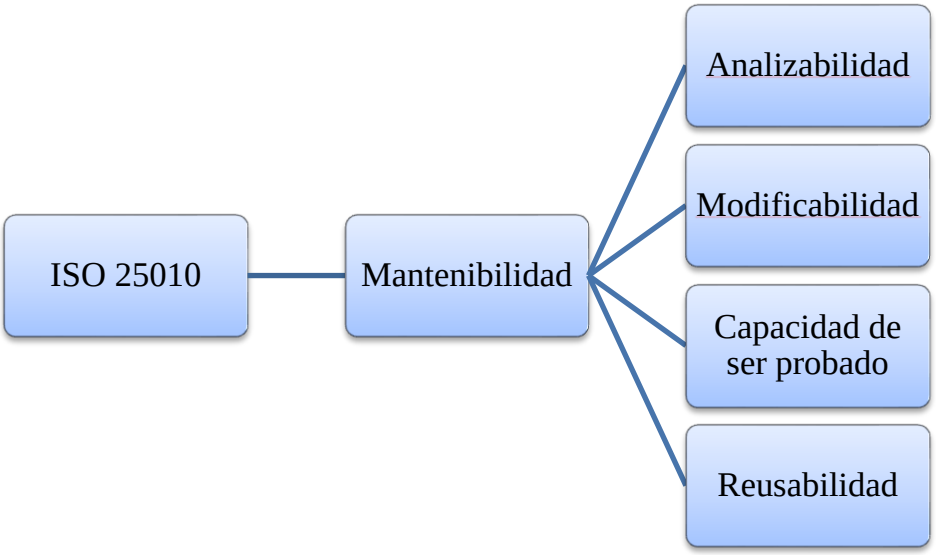


**Figura 10: Mantenibilidad ISO/IEC 25000**

#### 2.4.4.1. 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.

#### 2.4.4.2. 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:** reducir el costo de modificar una responsabilidad A y que esto no implique cambios en otras responsabilidades, reduce el costo de cualquier modificación que implique cambios en A.

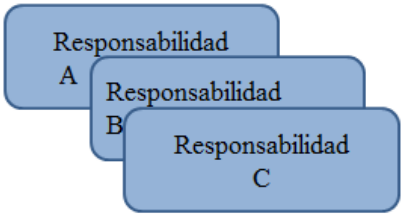


**Figura 11: Responsabilidades software**

**Grado de acoplamiento:** reducir el acoplamiento entre dos responsabilidades A y B reduce el costo de una modificación en la responsabilidad A.

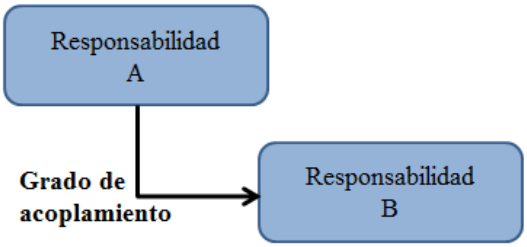


**Figura 12: Acoplamiento software**

**Cohesión:** las responsabilidades son asignadas a módulos del sistema. Si una responsabilidad A tiene un alto grado de acoplamiento con la responsabilidad B, el costo de cambiar A es menor si las dos responsabilidades son asignadas al mismo módulo.

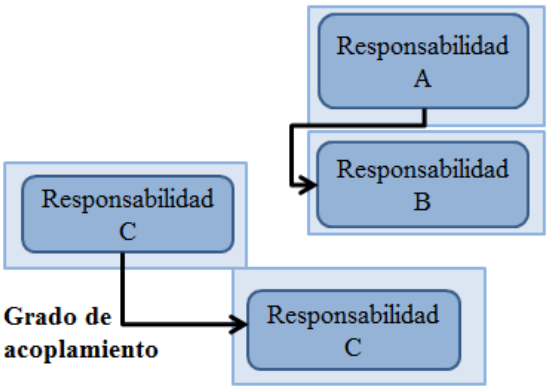


**Figura 13: Cohesión software**

Los cambios tempranos en el sistema son menos costosos que los cambios hechos más adelante

#### 2.4.4.3. 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.

#### 2.4.4.4. Reusabilidad

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 desarrollar 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.

Entre los elementos que intervienen en la reusabilidad de un producto software podemos encontrar los siguientes:

- Especificaciones de requisitos previamente concebidas.
- Diseños previamente definidos (Estructuras de datos, algoritmos, etc.)
- Código probado y depurado con anterioridad.
- Personal cualificado (aprovechamiento de la experiencia de los ingenieros de un proyecto a otro).
- Paquetes de software de propósito general.

Poner en práctica la reutilización del producto software aporta ventajas como puede ser el de reducir tiempos y costes de desarrollo y una mayor fiabilidad ya que estamos usando software que ya ha sido probado con anterioridad.

Por otro lado podemos distinguir varios tipos de reutilización:

- Oportunista: el ingeniero de software reutiliza piezas que él sabe que se ajustan al problema
- Sistemática
    - Esfuerzo a nivel organizacional y planificado de antemano.
    - Todo componente reutilizado ha de ser ideado, a priori, para ser reutilizado.
    - Implica inversiones iniciales para recoger frutos en el futuro.
    - Diseñar componentes genéricos para que sean reutilizados con facilidad.
    - Se desarrollan pequeños componentes para una determinada aplicación.
    - Se incorpora a un repositorio
- Top-Down
    - Se determinan las piezas necesarias que encajan unas con otras
    - Se van desarrollando poco a poco
    - Requiere alta inversión a comienzo
    - Se recogerán beneficios en el futuro

Tras analizar las sub-dimensiones de mantenibilidad de las ISO/IEC 9126 y 25010 se puede ver el como el concepto de Mantenibilidad ha ido cambiando ligeramente de un estándar al otro con el paso del tiempo.

- Hay dos subcaracterísticas nuevas: la reutilización y la modificabilidad.
- La subcaracterística de modificabilidad combina dos subcaracterísticas de las norma ISO/IEC 9126: cambiabilidad y estabilidad.
- El cumplimiento de estándares, que es una subcaracterística en ISO/IEC 9126, está ahora fuera del alcance del modelo de calidad en ISO/IEC 25010.

En la siguiente imagen se pueden apreciar las diferencias anteriormente descritas

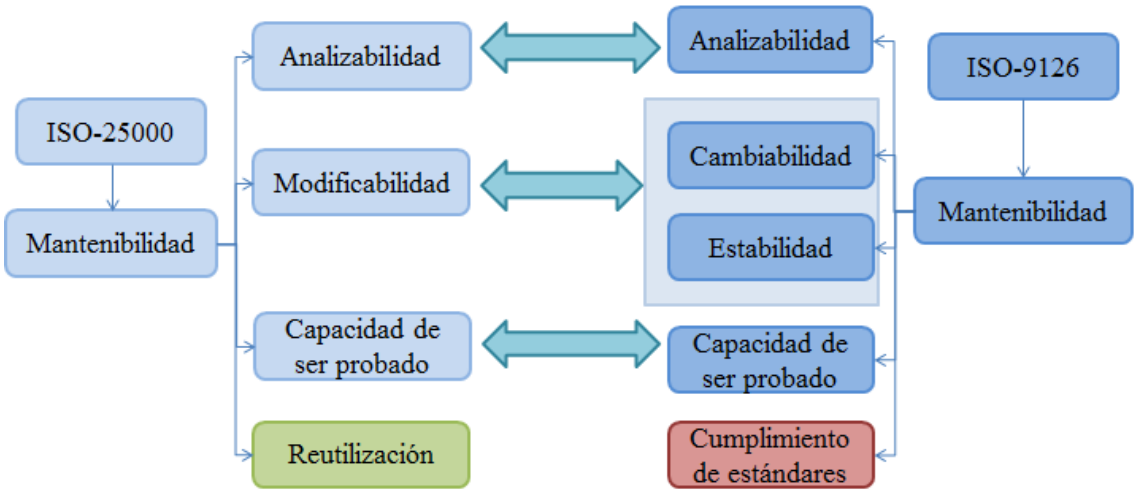


**Figura 14: Diferencias ISO-25000/ISO_9126**

# 3. Proceso de auditoría

## 3.1. Referente de mantenibilidad

Como se comentó anteriormente en el punto 2.1 la norma ISO/IEC 9126 propone un modelo de calidad que se divide en tres aproximaciones: interna (calidad del código), externa (calidad en la ejecución) y en uso.

Dentro de estas divisiones podemos encontrar informes técnicos internacionales que proporcionan un conjunto sugerido de métricas de calidad de software para ser utilizados con el modelo 9126-1. Apoyándonos en estos estándares vamos a proponer un conjunto de métricas que nos permitan medir la mantenibilidad de los productos software.

**9126-2 y 9126-3**

El término métrica en el campo de la informática puede definirse como cualquier medida o conjunto de medidas destinadas a conocer o estimar el tamaño u otra característica de un software.

Las métricas internas se pueden aplicar a un producto de software no ejecutable durante sus etapas de desarrollo (por ejemplo, la definición de requisitos, especificación de diseño o el código fuente). Las métricas internas proporcionan a los usuarios la capacidad de medir la calidad de los entregables intermedios y por lo tanto predecir la calidad del producto final. Esto permite al usuario detectar problemas de calidad y tomar acciones correctivas durante las primeras etapas de la proceso del ciclo de vida de desarrollo.

Las métricas externas se pueden usar para medir la calidad del producto de software mediante la medición del comportamiento del sistema del que forma parte. Las métricas externas sólo pueden ser utilizadas durante las etapas de prueba del ciclo de vida del proceso y durante cualquier etapa operacional. Esto se logra mediante la ejecución del producto de software en el sistema que está destinado a utilizarse.

La métrica de la calidad en uso mide el grado en que un producto cumple con las necesidades de los usuarios. Esto solo se consigue en un entorno de sistema realista.

Se recomienda el uso de métricas internas con una relación tan fuerte como sea posible con las métricas externas de producto. Sin embargo, a menudo es difícil diseñar un riguroso modelo teórico que proporcione una fuerte relación entre métricas internas y métricas externas.

Como se dijo en el apartado 2.1 el modelo de calidad según la ISO/IEC 9126-1 establece 10 características, seis de ellas comunes a las vistas internas y externas y las otras cuatro propias de las vistas en uso.

A continuación vamos a describir cada métrica con respecto a estas 6 características comunes a las vistas internas y externas.

- **Métricas de Funcionalidad:** mide el comportamiento funcional del sistema.
- **Métricas de Eficiencia:** mide el consumo del tiempo y recursos de comportamientos del sistema.
- **Métricas de Usabilidad:** mide cuando el software puede ser entendido, aprendido, operativo y atractivo.
- **Métricas de Mantenibilidad:** mide atributos tales como el comportamiento del usuario del sistema, persona encargada del mantenimiento, cuando el software se mantiene o modifica durante las pruebas o mantenimiento.
- **Métricas de Fiabilidad:** mide atributos relacionados con el comportamiento del sistema durante la ejecución de los test para indicar el grado de fiabilidad.
- **Métricas de Portabilidad:** mide atributos como el comportamiento del sistema durante una actividad de portabilidad.

En este TFG estamos tratando la característica de la mantenibilidad, por ello nos vamos a centrar en las métricas de miden esta característica según la norma ISO/IEC 25000.

Estás métricas se dividen en las siguientes:

- **Métricas de Analizabilidad:** mide atributos tales como el esfuerzo de la persona encargada del mantenimiento o el usuario, también mide los recursos necesarios para diagnosticar deficiencias o partes a modificar.
- **Métricas de Modificabilidad:** mide el esfuerzo de la persona encargada del mantenimiento y el usuario cuando intentan implementar una modificación.
- **Métricas de Capacidad de ser probado:** mide el esfuerzo de la persona encargada del mantenimiento midiendo su comportamiento cuando trata de probar el software modificado.

- **Métricas de Reusabilidad:** mide la capacidad que tiene el programa para poder reusarse para nuevos propósitos.

A continuación se detallarán las métricas que hemos definido para medir la mantenibilidad. En todas las métricas derivadas se utilizarán funciones de densidad, calculando el ratio, entre, por ejemplo, el tamaño y la complejidad ciclomática.

A continuación se detallarán las métricas que hemos definido para medir la mantenibilidad y se clasificarán en función de la sub-característica de mantenibilidad a la que pertenecen. Además estas métricas proporcionan unas tablas para clasificar los valores obtenidos en un ratio obteniendo el valor de las medidas de mantenibilidad.

Esto es fundamental, ya que los valores de las medidas de calidad software que componen una sub-característica pertenecen a distintos rangos numéricos. En el momento de componer la sub-característica es necesario que sus valores pertenezcan a los mismos rangos, por ejemplo valores entre 0 y 100.

La selección de medidas de calidad para la métrica de mantenibilidad se muestra en la siguiente tabla:

Aportación

| Subcaracterística | Medida de calidad |
|---|---|
| **Analizabilidad** | Complejidad ciclomática |
| | Código repetido |
| | Densidad Comentarios |
| | Densidad de defectos de la capacidad para ser analizado |
| **Modificabilidad** | Densidad de ciclos |
| | Documentación Publicada |
| | Nombres de las variables |
| | Densidad de defectos de la modificabilidad |
| **Capacidad de ser probado** | Densidad de pruebas |
| | Tamaño del programa |
| **Reusabilidad** | Estructura del programa |
| | Deuda técnica |
| | Densidad de defectos de Reusabilidad |

**Tabla 1: Métricas mantenibilidad**

A continuación se detallarán las métricas listadas en la Tabla anterior para la subcaracterística de analizabilidad.

## 1. Densidad de la complejidad ciclomática

La densidad de la complejidad ciclomática mide la relación entre la complejidad ciclomática de un producto y su tamaño (sin contar comentarios). De esta forma se puede obtener un indicador que es independiente del tamaño del proyecto.

La complejidad ciclomática es una métrica de calidad software basada en el cálculo del número de caminos independientes que tiene nuestro código.

Estos caminos se identifican a partir de las estructuras de control (condicionales, bucles, …) incluidas en la mayoría de los lenguajes de programación, y más concretamente, a partir del diagrama de flujo de cada uno de nuestros métodos.

La idea es sencilla, cuanto más compleja sea la lógica de un código, más difícil será de entender, mantener y probar.

Estas funciones de densidad favorecen la independencia de las métricas respecto de los proyectos. Así, por ejemplo, un proyecto con 10.000 líneas de código fuente y una complejidad ciclomática acumulada de 5.000 tendrá menor densidad de complejidad ciclomática que un proyecto con 500 líneas de código y una complejidad ciclomática acumulada de 400.

Para calcular la complejidad ciclomática, si nos atenemos a la definición de complejidad ciclomática, como detección y recuento de caminos independientes. La idea más "purista" es acudir al diagrama de flujo, tratarlo como un grafo, con sus nodos y aristas, y buscar cuántos caminos diferentes hay desde el nodo inicial al final.

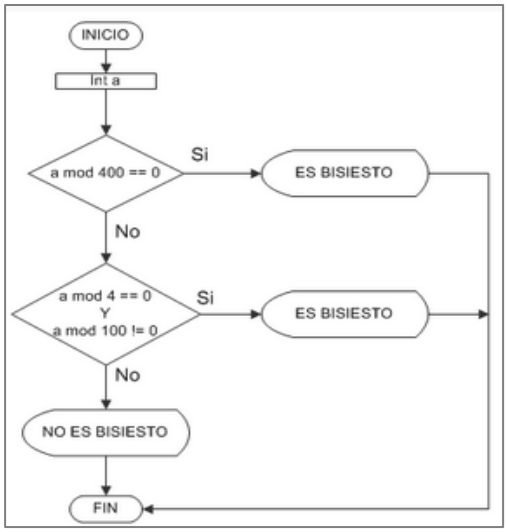


**Figura 15: Ejemplo Diagrama Flujo**

En el ejemplo anterior sobre calcular si un año es bisiesto o no, habrían 3 posibles caminos (SI, NO-SI, NO-NO).

Otra manera de calcular la complejidad ciclomática sería con la siguiente fórmula: *Complejidad Ciclomática = e – n + 2*, donde e representa el número de aristas y n el número de nodos.

Si para nuestro ejemplo numeramos cada instrucción (los nodos) y las transiciones entre las mismas (las aristas) nos queda:

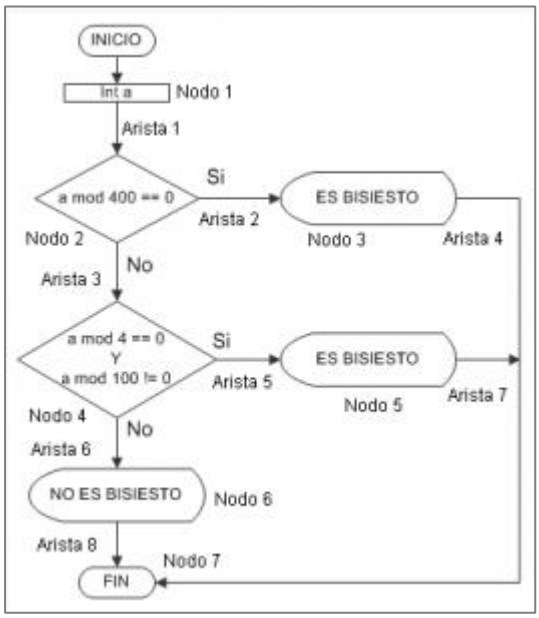


**Figura 16: Ejemplo cálculo complejidad ciclomática**

*v(G) = 8 -7 + 2 = 3, siendo e = 8; n = 7*

Las aplicaciones con una mayor densidad de la complejidad ciclomática son más difíciles de entender y más difíciles de probar (tienen una mayor densidad de caminos de ejecución) y, por lo tanto, más difíciles de mantener.

El objetivo de esta métrica es evaluar la complejidad funcional del código para conocer la capacidad del mismo para ser analizado y el esfuerzo requerido para revisar dicho código.

Función de medición: CCD = E – N + 2,

Elementos de la medida de calidad

- E: Aristas
- N Nodos.
- CCD: Densidad de complejidad ciclomática

En la siguiente tabla se ve una clasificación general para evaluar la complejidad ciclomática.

| Complejidad Ciclomática | Evaluación del Riesgo |
|---|---|
| **1-10** | Programa Simple, sin mucho riesgo |
| **11-20** | Más complejo, riesgo moderado |
| **21-50** | Complejo, programa de alto riesgo |
| **>=50** | Programa no testeable, muy alto riesgo |

**Tabla 2: Clasificación complejidad ciclomática**

## 2. Densidad de código repetido

La densidad de código repetido mide la relación entre la cantidad de código repetido de un producto y su tamaño (sin contar comentarios). De esta forma se puede obtener un indicador que es independiente del tamaño del proyecto.

A la hora de duplicar código, se duplican también los errores. Una densidad de código repetido alta, indica que la probabilidad de que un error esté repetido en varios sitios sea alta.

Su objetivo es evaluar la cantidad de código duplicado para conocer su capacidad para ser analizado, modificado y para evitar efectos inesperados a la hora de realizar una modificación.

Función de medición: DCR=LD/LT *100

Elementos de la medida de calidad:

- LD: Líneas de Código Duplicadas.
- LT: Líneas de totales del código – Líneas comentadas.
- DCR: Densidad de código repetido

Aportación

## 3. Densidad de comentarios

La densidad de comentarios mide la relación entre la cantidad de comentarios de un producto y su tamaño (obviamente, incluyendo comentarios). De esta forma se puede obtener un indicador que es independiente del tamaño del proyecto.

El objetivo es evaluar los comentarios en el código para conocer su capacidad para ser analizado y modificado.

Como se explica en números libros acerca de la calidad de los productos softwares en concreto en el libro *"Clean Code(Código Limpio)"* los comentarios nos

ayudan a explicar nuestras intenciones en el código y pueden ser bastante útiles, sin embargo no debemos colapsar nuestro código con comentarios intranscendentes.

El uso correcto de los comentarios compensa nuestra incapacidad para expresarnos en el código, ya que no es una labor fácil sobre todo para los programadores sin mucha experiencia. El formato de los comentarios puede ayudar al lector a comprender el objetivo del código y cómo se alcanza dicho objetivo.

Es mucho peor escribir un comentario impreciso que no escribir el comentario puesto que se puede confundir al usuario. Los comentarios tampoco compensan el código incorrecto, el código claro y expresivo sin apenas comentarios es muy superior al código enrevesado y complejo con multitud de comentarios.

La idea consiste en intentar explicarse en el código.

Función de Medición: $\mathrm{COMD} = \frac{\mathrm{LC}}{\mathrm{LT}} * 100$

Elementos de la medida de calidad:

- COMD: Densidad de Comentarios
- LC: Líneas comentadas
- LT: Total Líneas

Aportación

### 4. Densidad de defectos de la capacidad de ser analizado

La densidad de defectos de la capacidad de ser analizado mide la relación entre el número de defectos de analizabilidad y el tamaño del proyecto. Este indicador nos muestra lo complejo que es el software a la hora de ser analizado. Se calcula la densidad de violaciones al realizar un análisis estático del código fuente.

Función de Medición: Los resultados obtenidos se clasifican en función de la siguiente tabla, con valor de 1 a 100:

| Defectos de la capacidad para ser analizado | Resultado defectos analizabilidad |
|---|---|
| 1-25 | Muy Buena |
| 26-50 | Buena |
| 51-75 | Mala |
| 76-100 | Muy Mala |

**Tabla 3: Clasificación defectos de la capacidad para ser analizado**

Elementos de la medida de calidad

Estos se calculan a partir de los defectos que consideremos importantes a la hora de medir la calidad en nuestro software. Se hará una clasificación según la importancia de cada defecto sobre la mantenibilidad, la suma de todos ellos será 100. Los defectos de analizabilidad son:

- Ausencia de comentarios. (10)
- Elevada complejidad ciclomática. (40)
- Incoherencias en el código. (30)
- Código Repetido. (20)

Al igual que lo realizado con la subcaracterística de analizabilidad en este apartado se detallarán las métricas listadas en la tabla 1, para la subcaracterística de modificabilidad.

**1. Densidad de ciclos**

La densidad de ciclos de un producto mide la relación entre el número de dependencias cíclicas que presenta el producto y la cantidad de módulos existentes.

El objetivo es Evaluar la existencia de ciclos entre módulos para conocer las dependencias que pueden traer problemas entre dichos módulos.

Función de Medición: $CYCD = \frac{\Sigma CEM}{Módulos} * 100$

Elementos de la medida de calidad

- CYCD: Densidad de Ciclos.
- CEM: Ciclos entre módulos.
- Módulos: Módulos existentes.

Aportación

**2. Documentación**

Hay que distinguir dos tipos de documentación en un proyecto:

Documentación Interna:

Esta documentación está dirigida a quienes leerán el código fuente del programa. Incluye información para identificar el programa y describir algoritmos, estructuras de datos y flujos de control.

Normalmente esta información se ubica al comienzo de cada componente en forma de comentarios y se denomina bloque de encabezamiento.

El encabezamiento se escribe al principio de los programas y contiene la siguiente información acerca de cada componente:

- Qué denominación tiene el componente
- Quién ha escrito el componente
- Dónde va ubicado el componente en el diseño general del sistema
- Cuándo fue escrito, revisado o modificado

Documentación externa

La documentación externa está orientada para ser leída incluso por quienes tal vez nunca verán el código real. Por ejemplo los diseñadores o usuarios finales. Además aporta una explicación más amplia que los comentarios acerca de una aplicación. Responde a las cuestiones – quién, qué, cuándo, dónde y cómo desde la perspectiva del sistema.

Dado que un sistema de software se construye a partir de componentes interrelacionados, la documentación externa a menudo incluye una vista general de los componentes del sistema.

Al momento de escribir el código de un componente, debemos explicar su estructura y flujo en los documentos de diseño. Para ello la documentación externa del componente debe incluir las siguientes descripciones:

Descripción del problema: en la primera sección de la documentación se debe explicar el problema que vamos a tratar.

Descripción de algoritmos: una vez que se ha esclarecido el porqué de la existencia del componente, se debe respaldar la elección del algoritmo. D

Descripción de los datos: en la documentación externa, los usuarios y los programadores deberán ser capaces de ver el flujo de datos a nivel de componente. Para componentes con un diseño orientado a objetos, la visita general de objetos y clases debe explicar la interacción general entre dichos objetos.

Función de Medición:

*Densidad de API documentado público = (API Pública - API indocumentada Pública ) / API Pública * 100*

La siguiente tabla muestra una clasificación según la documentación publicada:

| % **Documentación** | **Valoración** |
|---|---|
| **95% - 100%** | Muy buena |
| **70%-95%** | Buena |
| **40%-70%** | Mala |
| **0%-40%** | Muy mala |

**Tabla 4: Clasificación documentación publicada**

Aportación

### 3. Nombres de las Variables

En el software, los nombres son omnipresentes. Aparecen en variables, funciones, clases, etc .Usamos nombres constantemente, por ello necesitamos usar nombres que revelen nuestras intenciones, elegir nombres correctos lleva tiempo pero a la vez ahorra trabajo y comentarios innecesarios.

Ej: *Int e;// coste en Euros*

El nombre "e" no revela nada por lo que necesitamos escribir el comentario para aclarar el uso de la variable, y cada vez que hagamos uso de esta variable necesitaríamos recordar el comentario anterior. Por lo tanto es recomendable usar un nombre de variable más explicativo (en este caso Euros, aunque no es útil utilizar palabras redundantes).

Otro consejo útil es el de utilizar nombres que se puedan pronunciar. Un ejemplo se da al utilizar nombres de variables de una sola letra. Un contador de bucles se podría bautizar como i,j o k (pero nunca l) si su ámbito es muy reducido y no hay conflictos con otros nombres, ya que los nombres de una letra son tradicionales en contadores de bucles. Sin embargo, en otros contextos, un nombre de una letra es una opción muy pobre: es como un marcador de posición que el lector debe asignar mentalmente a un concepto real. No hay peor motivo para usar el nombre c que a y b ya estén seleccionados.

Función de Medición: DNCS=NVCS/NVT*100

Elementos de la medida de calidad

- DNCS: Densidad de nombres con sentido.
- NVCS: Número de variables con nombre con sentido.
- NVT: Número de variables totales.

Aportación

### 4. Densidad de defectos de la modificabilidad

La densidad de defectos de la capacidad de ser cambiado un producto se mide como la relación entre el número de defectos de modificabilidad que presenta el producto y su tamaño.

Función de Medición Los resultados obtenidos se clasifican en función de la siguiente tabla, con valor del 1 al 100:

| | |
|---|---|
| DCA=1-25 | Muy Buena |
| DCA=26-50 | Buena |
| DCA=51-75 | Mala |
| DCA=76-100 | Muy Mala |

**Tabla 5: Clasificación defectos de la modificabilidad**

Elementos de la medida de calidad

Estos se calculan a partir de los defectos que consideremos importantes a la hora de modificar nuestro software. Se hará una clasificación según la importancia de cada defecto sobre la modificabilidad, clasificando el resultado total del 1 al 100. Estos defectos son:

- Acoplamiento entre clases. (30)
- Incoherencias en el código (30)
- Código Repetido (25)
- Ausencia de comentarios. (15)

Al igual que lo realizado con las otras subcaracterísticas en este apartado se detallarán las métricas listadas en la tabla 1 para la subcaracterística de la capacidad de ser probado.

### 1. Densidad de pruebas

A la hora de hacer pruebas, hay que prestar especial atención a la complejidad ciclomática. Las pruebas unitarias deben recorrer todos los caminos ejecutables definidos en el código fuente. En este sentido, sería deseable que el número de pruebas unitarias fuese aproximadamente igual a la complejidad ciclomática, es decir una prueba por cada camino.

La densidad de pruebas mide la relación entre el número de pruebas unitarias y la complejidad ciclomática, es decir, el número de pruebas unitarias codificadas en

relación con el número de pruebas unitarias deseable. Este es un buen indicador de la amplitud y profundidad con la que se han realizado las pruebas.

Función de Medición $UTD = \Sigma UN / \Sigma CC$

Elementos de la medida de calidad

- UTD: Densidad de Pruebas Unitarias.
- UN: Número de Pruebas Unitarias.
- CC: Complejidad Ciclomática.

**2. Tamaño**

Mide el tamaño de cada parte de nuestro programa, como pueden ser el número total de líneas, número de clases, número de directorios, número de archivos etc.

- Descriptores de acceso: número de funciones "get" y "set" utilizados como lectura, escritura o propiedad de una clase.
- Clases: número de clases.
- Directorios: número de directorios.
- Archivos: número de archivos.
- Líneas: número de líneas físicas que contienen al menos un espacio en blanco o una tabulación o parte de un comentario.
- Métodos: número de funciones.

Al igual que lo realizado con las otras subcaracterísticas en este apartado se detallarán las métricas listadas en la tabla 1 para la subcaracterística de reusabilidad.

Aportación

**1. Estructura del programa**

Para cualquier tipo de diseño, es importante que la estructura del programa refleje la estructura de control planteada a nivel diseño. Es importante que el lector pueda establecer el camino correcto de un código sin necesidad de desplazarse a saltos a través del código. En consecuencia, nuestro código debe estar escrito de tal manera que un componente pueda leerse fácilmente desde lo general a lo particular (enfoque top-down).

En el siguiente ejemplo se demuestra cómo cambia la facilidad de comprensión del código gracias a una buena restructuración

```
Benefit=mínimum
If(age < 75) goto A;
Benefit = maximum;
Goto C;
If (age < 65 ) go to B;
If (age < 55) go to C;
A:          if  ( age < 65) goto B;
 Benefit = benefit * 1.5 + bonus;
 Goto C;
B:          if (age < 55) goto C;
 Benefit = benefit * 1.5;
C:          next statement
```

Se podría conseguir lo mismo, en un formato mucho más fácil de seguir con reordenamiento del código:

```
If ( age < 55) benefit = mínimum;
Else if ( age < 65 ) benefit = mínimum + bonus;
Elseif ( age <75) benefit= minimum * 1.5 + bonus;
Else benefit=maximum;
```

Siempre debemos poner la acción inmediata después de la decisión que la genera, aunque hay veces que no es posible conseguir tener un flujo top-down.

Aportación

## 2. Deuda Técnica

Esta métrica proporciona la ratio entre la deuda técnica actual y el esfuerzo que se debería de invertir para reescribir todo el código desde el principio. Esto último se estima basándose en el número de líneas de código o la complejidad global. Esto significa que el valor de esta métrica depende del tamaño del proyecto. Para el cálculo de la deuda técnica nos basaremos en la metodología SQALE (Software Quality Assessment based on Lifecycle Expectations).

Función de Medición:
Nos apoyaremos en la siguiente tabla:

| Clasificación SQALE | Resultado |
|---|---|
| **A** | Muy buena |
| **B** | Buena |
| **C** | Media |
| **D** | Mala |
| **E** | Muy mala |

**Tabla 6: Clasificación deuda técnica**

Aportación

### 3. Densidad de defectos de Reusabilidad

La densidad de defectos de la capacidad de ser analizado mide la relación entre el número de defectos de reusabilidad y el tamaño del proyecto. Este indicador nos muestra lo complejo que es el software a la hora de ser reutilizado.

Función de Medición: Los resultados obtenidos se clasifican en función de la siguiente tabla, con valor de 1 a 100:

| Defectos de la capacidad para ser analizado | Resultado defectos analizabilidad |
|---|---|
| 1-25 | Muy Buena |
| 26-50 | Buena |
| 51-75 | Mala |
| 76-100 | Muy Mala |

**Tabla 7: Clasificación defectos de la reusabilidad**

Elementos de la medida de calidad

Estos se calculan a partir de los defectos que consideremos importantes a la hora de medir la reusabilidad en nuestro software. Se hará una clasificación según la importancia de cada defecto sobre la reusabilidad, la suma de todos ellos será 100. Los defectos de analizabilidad son:

- Uso de excepciones (100)

## 3.2.ISO 19011 e ISO/IEC 25040

En este apartado vamos a desarrollar la norma ISO 19011 (que describe las directrices para la auditoría de los sistemas de gestión) adaptándola a una auditoría de mantenibilidad de aplicaciones software.

El objetivo es proponer un proceso específico de auditoría de mantenibilidad de aplicaciones software, para ello, nos apoyaremos las norma ISO 19011 y ISO/IEC 25040 descritas anteriormente, y teniendo como referente el conjunto de métricas definidas.

Para nuestro TFG nos vamos a centrar en las actividades de mantenibilidad, proporcionando orientación sobre la planificación y forma de llevar a cabo dichas actividades como parte de un programa de auditoría. La siguiente figura proporciona una visión general de las actividades de auditoría típica.

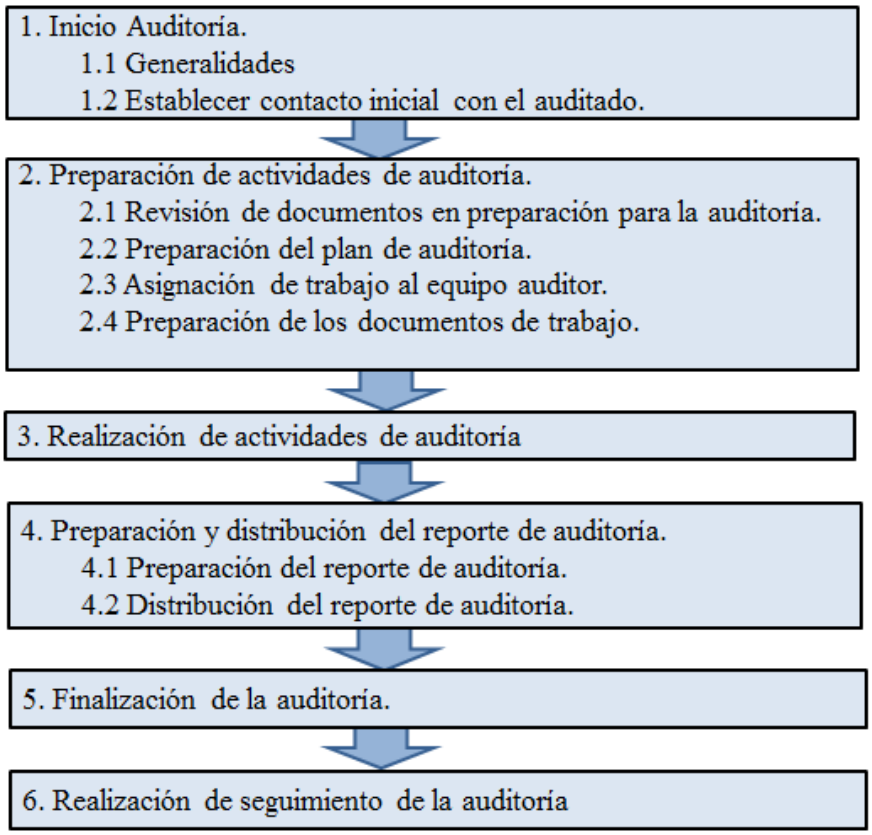


**Figura 17: Esquema Auditoría Mantenibilidad**

Nos apoyaremos en estas actividades para plantear un nuevo proceso de auditoría especializado en la auditoría de mantenibilidad de aplicaciones software. A partir de este proceso de auditoría vamos a poder aplicar las métricas definidas en el apartado 3.1 y conseguir un proceso de auditoría de mantenibilidad completo.

Vamos a pasar a describir punto por punto las actividades para una auditoría de mantenibilidad.

## 1. Inicio de la auditora

Aportación

### 1.1. Generalidades

Cuando se da inicio a una auditoría de mantenibilidad, hay que definir el equipo de auditoría que va a desarrollar el proceso de auditoría. La responsabilidad de llevar a cabo dicha auditoría es del líder del equipo auditor hasta que la auditoría se haya finalizado.

Para una auditoría de mantenibilidad es importante dejar claro el rol de cada persona del equipo auditor y exige un gran nivel de compenetración entre ellos y de competencia por su parte.

La asignación debería hacerse con suficiente tiempo antes de la fecha de la auditoría, a fin de asegurar una planeación efectiva de la misma.

Para asegurar la conducción efectiva de la auditoría, se debería entregar la siguiente información al líder del equipo auditor:

a) Objetivos de auditoría de mantenibilidad; el principal objetivo es evaluar el grado de la mantenibilidad de aplicaciones software a ser auditadas.

b) Definir criterios o referente de auditoría y cualquier documento de referencia. Para realizar una auditoría de mantenibilidad debemos entender las normas ISO/IEC 9126,14598 y 25000, en ellos se describe la dimensión de mantenibilidad. Estos estándares han sido los documentos de referencia para desarrollar el proceso de auditoría de mantenibilidad, gracias a ellos hemos obtenido las métricas a tener en cuenta a la hora de realizar el caso práctico.

c) Alcance de auditoría, se definirán las partes que cubre el alcance de nuestra auditoría, que es medir la mantenibilidad de la aplicación. Además al principio del proceso de auditoría deben quedar claras las partes que no cubre nuestro proceso de auditoría. En nuestro caso es importante dejar claro que nuestro TFG se centra en una auditoría de mantenibilidad, excluyendo todas aquellas dimensiones de la calidad del software que no sean la mantenibilidad.

d) Métodos y procedimientos de auditoría, serían los pasos que vamos a seguir para auditar la aplicación definidos en nuestra memoria del TFG. Al tratarse de un nuevo proceso de auditoría vamos a combinar los estándares 19011 y 25040 con el fin de poder incluir el proceso concreto de mantenibilidad en un proceso como el 19011 que describe las directrices para la auditoría de los sistemas de gestión.

e) Composición del equipo auditor, el equipo de auditoría que evaluará la aplicación. Los integrantes del equipo deberán tener un gran conocimiento y experiencia en el proceso de auditoría en general. Así mismo es necesario que hayan tenido en periodos laborales anteriores experiencia en el análisis y desarrollo de aplicaciones.

f) Adjudicación de recursos apropiados para llevar a cabo la auditoría; para evaluar la mantenibilidad del software necesitamos toda la documentación posible de la aplicación a auditar, su código fuente, y las herramientas para medir la mantenibilidad, como pueden ser: SonarQube o VerifySoft.

### 1.2. Establecer contacto inicial con el auditado

Para llevar a cabo la auditoría de mantenibilidad es importante el contacto inicial con el auditado, puede ser de forma formal o informal y debería hacerlo el líder del equipo auditor. Los propósitos del contacto inicial son los siguientes:

a) Establecer comunicación con los representantes del auditado ya que a ellos va dirigida la auditoría de mantenibilidad.

b) Confirmar la autoridad para la realización de la auditoría; los responsables del proyecto han de dar el consentimiento para que su aplicación pase una auditoría de mantenibilidad. Es fundamental analizar el código fuente de las aplicaciones a auditar ya que la mantenibilidad depende en gran parte de dicho código, para tener acceso al código necesitamos confirmar la autoridad.

c) Proveer información sobre los objetivos, alcance y métodos de auditoría, así como la composición del equipo auditor, incluyendo los expertos técnicos; el objetivo de nuestra auditoría será auditar un proyecto software para que cumpla los estándares de mantenibilidad definidos en las métricas 9126 y 25000, es importante dejar claro que esta auditoría solo va a medir la mantenibilidad de la aplicación y no otras características como pudieran ser la fiabilidad, eficiencia, usabilidad etc.

d) Solicitar acceso a documentos y registros relevantes para propósitos de planeación; como sería el código fuente de la aplicación que vamos a auditar y toda la documentación relativa a ella.

e) Confirmar el acuerdo del auditado en lo referente al grado de divulgación y tratamiento de la información confidencial.

f) Hacer arreglos para la auditoría, incluyendo la programación de fechas.

**2. Preparación de actividades de auditoría.**

**2.1. Revisión de documentos en preparación para la auditoría.**

La documentación relevante del proyecto software que vamos a auditar debería ser revisada con el fin de:

a) Reunir información para preparar las actividades de una auditoría de mantenibilidad.

b) Establecer una visión general del grado de documentación del sistema de gestión para detectar posibles vacíos. Toda aquella documentación a la que no tengamos acceso quedará fuera del alcance de nuestra auditoría de mantenibilidad.

Para poder auditar la mantenibilidad necesitaremos preparar internamente las actividades de auditoría apoyándonos en las ISO 19011 y 25040. Además debemos recopilar toda la documentación posible relativa al proyecto a auditar, y preguntar al auditado por la documentación que consideramos importante.

### 2.2. Preparación del plan de auditoría.

El líder del equipo auditor deberá preparar un plan de auditoría de mantenibilidad basado en la información contenida en el programa de auditoría y en la documentación entregada por el auditado. El plan debería facilitar la programación y coordinación eficiente de las actividades de auditoría a fin de alcanzar efectivamente los objetivos.

El plan de auditoría consiste en medir la mantenibilidad de un proyecto software ayudándonos de las herramientas de análisis de código y documentación, en base a las métricas definidas en el apartado 3.1. Se deben redactar las conclusiones que ha obtenido el equipo auditor durante el proceso de auditoría.

### 2.3. Asignación de trabajo al equipo auditor.

Se debe asignar el trabajo entre los componentes del equipo auditor. Para la auditoría de mantenibilidad hay que tener en cuenta que se va a auditar tanto el código fuente del software como toda la documentación externa y externa relativa al proyecto.

### 2.4. Preparación de los documentos de trabajo.

Los miembros del equipo auditor deberían recolectar y revisar la información pertinente a las tareas asignadas y preparar los documentos de trabajo que sean necesarios para evaluar la mantenibilidad.

Estos documentos de trabajo deben incluir las métricas que se van a seguir para la evaluación.

Los documentos de trabajo, incluyendo los registros que resultan de su uso, deberían retenerse al menos hasta que finalice la auditoría, o de acuerdo con lo especificado en el plan de auditoría. Aquellos documentos que contengan información confidencial o de propiedad privada deberían ser guardados con la seguridad apropiada en todo momento por los miembros del equipo auditor.


## 3. Realización de actividades de auditoría.

### 3.1. Generalidades.

Las actividades de auditoría normalmente son llevadas a cabo en una secuencia definida, en nuestro caso nos vamos a apoyar en el estándar 25040 que define el proceso de evaluación de la calidad del producto software. A partir de este estándar vamos a desarrollar un proceso de evaluación de la mantenibilidad de productos software.

El estándar 25040 forma parte de la serie de los estándares SQuaRE de las normas 25000 que fue descrito en el punto 2.2 de esta memoria y contiene los requisitos generales para la evaluación dela calidad de los productos software.

Para el proceso de auditoría de nuestro proyecto, nos vamos a ayudar de la siguiente figura definida en la norma ISO/IEC 25040:

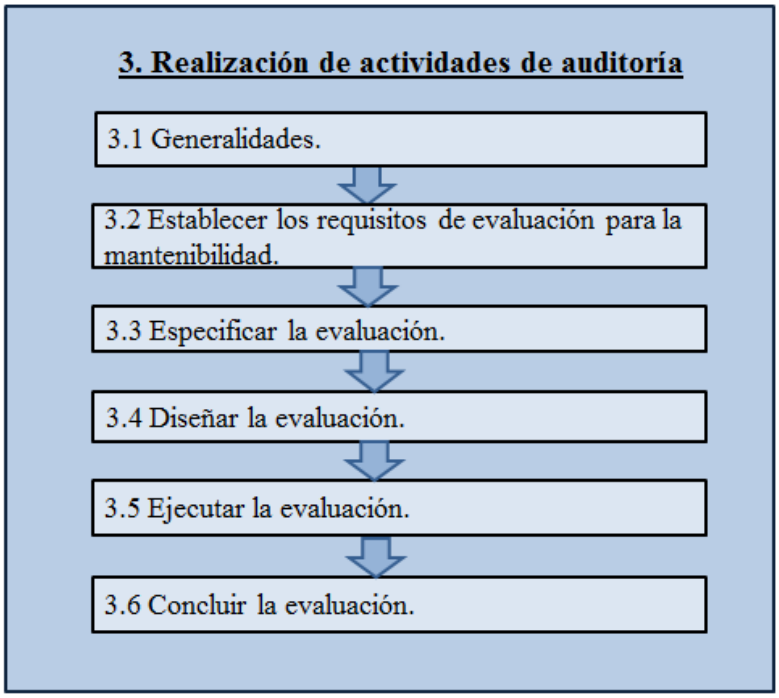


**Figura 18: Actividades Auditoría Mantenibilidad**

### 3.2. Establecer los requisitos de evaluación para la mantenibilidad.

Al inicio de la auditoría es necesario realizar una reunión para dejar claro el alcance de la auditoría y establecer los requisitos para la evaluación.

Se han de obtener los requisitos que ha de cumplir una aplicación para que sea mantenible, estos requisitos son las métricas que hemos definido en el apartado 3.1 de esta memoria. Es necesario que se respeten cada una de las sub-dimensiones de la mantenibilidad. Estas son: analizabilidad, modificabilidad, estabilidad, capacidad de ser probado y reusabilidad.

Los puntos siguientes deben ser tenidos en cuenta a la hora de elaborar los requisitos de mantenibilidad del software:

a) El propósito de la evaluación; en el apartado 1 de esta memoria se ha definido el concepto de mantenibilidad, la importancia que tiene esta en una aplicación software y el coste que tendría para un equipo de proyectos el desarrollo de una aplicación sin tener en cuenta el concepto de la mantenibilidad.

b) Especificación de los requisitos de mantenibilidad del producto; definidos en las sub-características de mantenibilidad descritas en los estándares 25000 y 9126.

c) Herramientas de medición aplicable y metodología; hoy en día hay muchas herramientas que nos permiten medir y evaluar la mantenibilidad de un producto software entre ellas están SonarQube y VerifiSoft.

d) Identificar las partes del producto que pueden afectar a la mantenibilidad del mismo. Como es el código fuente, la documentación y el control de versiones por ejemplo.

Además en este punto es imprescindible acotar el alcance de la auditoría. Para una auditoría de mantenibilidad tenemos que auditar los documentos y el código fuente de la aplicación, ya que estos afectan notoriamente a la mantenibilidad del software. Pero podrían quedar fuera documentos como el manual de usuario de la aplicación, el documento de entrega para su pase a producción y todos aquellos que no afecten la mantenibilidad del software.

### 3.3. Especificar la evaluación.

Se ha de dejar claro cómo se va a evaluar la mantenibilidad de la aplicación. Se deben seleccionar las métricas de mantenibilidad que se van a examinar y definir los criterios de decisión que se van a aplicar a estas métricas. Para ello se han definido las tablas de evaluación en el apartado de métricas.

Es necesario evaluar la documentación del programa ya que la mantenibilidad depende en cierta manera del grado de documentación que tenga el software a auditar. Esta documentación debería ser revisada para:

- Recopilar información para soportar las actividades de auditoria.
- Identificar todos los componentes del proyecto que puedan alterar la mantenibilidad.

La revisión puede estar combinada con otras actividades de auditoría y puede continuar a lo largo de la misma, en tanto esto no vaya en detrimento de la efectividad en la realización de la auditoría.

### 3.4. Diseñar la evaluación.

Para diseñar la evaluación el equipo auditor puede apoyarse en las normas 9126 y 25000. En ellas se define el concepto de calidad del software y se pueden aplicar para evaluar la mantenibilidad, ya que es una dimensión de la calidad. El equipo de auditoría debería reunirse periódicamente para intercambiar información, preparar el plan de evaluación, evaluar el progreso de la auditoría y re-asignar trabajo entre los miembros del equipo auditor, según resulte necesario.

Los siguientes puntos deben ser los resultados de esta actividad:

a) Especificación de los propósitos de evaluación de la mantenibilidad del software.

b) Especificación del plan de evaluación de la mantenibilidad, para ello hemos adaptado las normas ISO 19011 y 25040 a un proceso concreto para evaluar la mantenibilidad.

El plan de evaluación debe definir criterios de aceptación o rechazo. Esto se debe hacer con el fin de disminuir el riesgo de errores y reducir el esfuerzo de la evaluación prevista, teniendo en cuenta al menos los siguientes elementos:

a) El presupuesto de la evaluación.

b) Los métodos de evaluación y estándares adaptados.

c) Herramientas de evaluación.

d) Las actividades de evaluación, incluyendo el calendario y los recursos involucrados.

### 3.5. Ejecutar la evaluación.

Para evaluar la mantenibilidad de un producto software es recomendable contar con alguna herramienta que analice el código fuente y la documentación. Hay varias herramientas que se pueden utilizar para la evaluación del software como pueden ser VerifySoft o SonarQube entre otras.

Estas herramientas nos mostrarán los resultados más destacables que pueden afectar a la mantenibilidad del software, a partir de estos resultados se evaluarán las métricas de mantenibilidad definidas y se determinará en qué medida estas se cumplen.

Durante una auditoría de mantenibilidad, la información relevante a los objetivos, alcance y criterios de mantenibilidad debería ser recolectada por medio de

muestreo apropiado y debería ser verificada. Solo información verificable debería ser aceptada como evidencia de auditoría. La evidencia de auditoría que conduce a hallazgos de auditoría debería ser registrada. Si durante la recolección de evidencia el equipo auditor conoce de circunstancias o riesgos nuevos o cambiantes, estos deberían ser tratados por el equipo de manera concordante.

En un informe de evaluación de la mantenibilidad de productos software se preparará la documentación de las actividades de evaluación y los resultados de la evaluación.

Se requieren medidas rigurosas para hacer comparaciones fiables.

### 3.6. Concluir la evaluación de mantenibilidad.

Una vez que hemos ejecutado la evaluación de mantenibilidad debemos de interpretar los resultados obtenidos y elaborar un informe donde se describan los procedimientos realizados y se analicen los resultados.

La evidencia de auditoría debería ser evaluada contra los criterios de la auditoría a fin de determinar los hallazgos de la auditoría. Los hallazgos de auditoría pueden indicar si una aplicación cumple los criterios de mantenibilidad.

La evidencia recolectada durante la auditoría que surgiera un riesgo significativo para el auditado debería ser reportado sin demora al auditado, y cuando sea apropiado, al cliente de auditoría. Por ejemplo una muy elevada complejidad ciclomática, o un gran porcentaje de código repetido supondrían un elevado riesgo de que no se vaya a pasar la auditoría.

Las no conformidades y su soporte de evidencia de auditoría deberían ser registradas. Las no conformidades pueden estar clasificadas. Estas deberían ser revisadas con el auditado a fin de obtener reconocimiento de que la evidencia de auditoría es correcta y que las no conformidades son entendidas. Se debería realizar todo intento de resolver opiniones divergentes relacionadas con la evidencia o hallazgos de auditoría; cualquier punto sin resolver debería ser registrado.

El equipo auditor debería reunirse antes de la reunión de cierre con el fin de reflejar los hallazgos de la auditoría antes de exponerlos al equipo auditado:

a) Revisar los hallazgos de la auditoría y cualquier otra información apropiada recopilada durante la auditoría frente a los objetivos de la misma.

b) Llegar a un acuerdo respecto a las conclusiones, teniendo en cuenta la incertidumbre inherente en el proceso de auditoría.

c) Preparar recomendaciones para mejorar la mantenibilidad de la aplicación, si esto está especificado en el plan de auditoría.

d) Discutir el seguimiento a la auditoría, ya que cada cambio en el software supone un cambio en la mantenibilidad del mismo.

Los Criterios de decisión son umbrales numéricos utilizados para determinar la necesidad de una acción o de una mayor investigación. Han sido establecidos con respecto a los requisitos de mantenibilidad y los correspondientes criterios de evaluación. En nuestro caso estos criterios de decisión son las métricas definidas anteriormente para la característica de mantenibilidad, los umbrales numéricos son calculados según las fórmulas descritas en el apartado anterior.

La evaluación de la mantenibilidad debe ser llevada a cabo una vez tenemos desarrollada la aplicación para poder analizar completamente todas las métricas.

En una auditoría de mantenibilidad no se puede evaluar cada ciclo de vida del proyecto, ya que es el conjunto final lo que determina si un software es más o menos mantenible.

## 4. Preparación y distribución del informe de auditoría.

### 4.1. Preparación del informe de auditoría

El líder del equipo auditor debería reportar los resultados de acuerdo con los procedimientos del programa de auditoría.

El informe de auditoría debería proveer un registro completo, exacto, conciso y claro de la auditoría y debería incluir o hacer referencia a lo siguiente:

a) Los objetivos de la auditoría; que son medir la mantenibilidad de un producto software.

b) El alcance de la auditoría, particularmente la identificación de las unidades de la organización y de las unidades funcionales relacionadas con la mantenibilidad.

c) Identificación del cliente de auditoría; en nuestro caso serían los responsables del proyecto auditado.

d) Identificación del equipo auditor y los participantes del auditado en la auditoría.

e) Criterios de mantenibilidad; que han sido desarrollados en las métricas.

f) Hallazgos de la auditoría y la evidencia relacionada.

g) Conclusiones de la auditoría.

### 4.2. Distribución del informe de auditoría

El informe de auditoría debería ser emitido dentro de un periodo de tiempo acordado.

El evaluador, debe revisar los resultados de la evaluación y la validez del proceso de evaluación, indicadores y medidas aplicadas.

El reporte de la auditoría debería entonces ser distribuido a los receptores designados en los procedimientos o plan de auditoría. Los responsables del proyecto.

## 5. Finalización de la auditoría

La auditoría finaliza cuando todas las actividades relacionadas con la medición de la mantenibilidad del producto hayan sido llevadas a cabo.

Se debería llevar a cabo una reunión de cierre, facilitada por el líder del equipo auditor, para presentar los hallazgos y conclusiones de la auditoría.

Si es necesario, el líder del equipo auditor debería prevenir al auditado de las situaciones encontradas durante la auditoría que pudieran disminuir la confianza en las conclusiones de la auditoría.

Cualquier opinión divergente relativa a los hallazgos de la auditoría y/o a las conclusiones entre el equipo auditor y el auditado deberían discutirse y, si es posible, resolverse. Si no se resolvieran, las dos opiniones deberían registrarse.

Se pueden presentar recomendaciones para mejorar la mantenibilidad del producto.

Las conclusiones de auditoría se tratarán en esta reunión de cierre y tienen en cuenta aspectos como los siguientes:

a) El grado de conformidad con los criterios de la auditoría y la mantenibilidad del software, incluyendo en forma de resumen los aspectos que más están afectando a la mantenibilidad del proyecto.

b) Logro de los objetivos de auditoria, cubriendo todos los aspectos relacionados con la mantenibilidad.

c) Causas raíz de los hallazgos, es decir evidencias claras para cada hallazgo relativo a la mantenibilidad del producto.

Las conclusiones de la auditoría pueden llevar a recomendaciones para la mejora o futuras actividades de auditoría.

El evaluador y el solicitante deberán realizar un examen conjunto de los resultados de la evaluación, para ello nos debemos reunir con las personas que han solicitado la auditoría y revisar los resultados obtenidos. Cuando la evaluación se completa los datos y elementos de evaluación, deberán ser dispuestos de acuerdo con los requisitos del solicitante.

Las lecciones aprendidas a raíz de la auditoría deberían ser incluidas en el proceso de mejora continua del sistema de gestión de las organizaciones auditadas.

### 6. Realización de seguimiento de la auditoría

Las aplicaciones software tienden a cambiar continuamente añadiendo nuevas funcionalidades al producto. Estos nuevos desarrollos afectan a la mantenibilidad de la aplicación. Dependiendo de los objetivos de la auditoría, las conclusiones de la auditoría pueden indicar la necesidad de acciones correctivas, preventivas, o de mejora para nuevos desarrollos.

# 4. Caso Práctico

En este apartado vamos a realizar el caso práctico de una auditoría de mantenibilidad de productos software. Para ello nos vamos a basar en el proceso de auditoría de mantenibilidad definido en el punto anterior.

En el proceso de auditoría descrito se definió que era importante contar con alguna herramienta de análisis de código para poder interpretar los resultados obtenidos y determinar su mantenibiliad, para este caso práctico vamos a utilizar las herramientas SonarQube y Verifysoft. Estas se describen en el Anexo 1

Una vez descargada instaladas y configuradas las herramientas SonarQube y VerifySoft, pasamos a buscar algún proyecto software escrito en JAVA en la plataforma Github. GitHub es una plataforma de desarrollo colaborativo de software muy conocida para alojar proyectos utilizando el sistema de control de versiones Git. En ella encontramos el proyecto que vamos a analizar.

**TextSecure**

TextSecure es una aplicación de mensajería para la comunicación privada.

TextSecure puede utilizar cualquiera de los datos (WiFi / 3G / 4G) o SMS para comunicarse de forma segura, y todos los mensajes TextSecure también pueden ser encriptados localmente en el dispositivo. Esta aplicación se puede obtener en la plataforma de distribución digital de aplicaciones móviles Google Play.

Esta es una aplicación escrita en JAVA que es el lenguaje configurado por defecto en SonarQube.

Para llevar a cabo el proceso de auditoría de mantenibilidad realizaremos todas las actividades que se definieron en el apartado anterior.

# 1. INICIO AUDITORÍA

## 1.1. Generalidades

Para empezar una auditoría de mantenibilidad lo primero ha sido definir el equipo de auditoría. Este está formado por el tutor que será el que gestione el programa de auditoría y el alumno, que será el líder del equipo auditor. Una vez que se definió el equipo de auditoría se asignaron las tareas y se hizo una planificación. Esta parte se realizó en septiembre calculando que el tiempo de finalización de la auditoría sería Junio.

En estas primeras reuniones se definieron los objetivos de la auditoría, el principal objetivo es medir la mantenibilidad de la aplicación TextSecure.

Posteriormente hemos definido el alcance de la auditoría, para ello se ha dejado claro en todo momento que esta auditoría solo cubre la mantenibilidad del programa.

Además al inicio de la auditoría se analizaron los estándares 9126,14598 y 25000 que serán la base para entender la dimensión de mantenibilidad. Fue durante las primeras reuniones cuando se analizaron las herramientas existentes que se podrían utilizar para medir la mantenibilidad de una aplicación.

## 1.2. Establecer contacto inicial con el auditado

Como dice el proceso de auditoría descrito en el apartado 3, al comienzo de la auditoría se debe conocer a los auditados, en nuestro caso vamos a auditar la herramienta TextSecure, no hemos tenido contacto directo con los auditados al tratarse de un TFG, pero hay mucha documentación en internet incluso tienen su propia web. Además en el portal Githup hemos encontrado el código fuente de la aplicación y documentación.

Al ser una aplicación pública se confirma que tenemos autoridad para la realización de la misma, aunque esta auditoría no tenga ningún efecto sobre los creadores de la aplicación.

No ha sido necesario establecer una comunicación con los auditados ya que ponen toda la documentación en el dominio público.

## 2. Preparación de actividades de auditoría.

### 2.1. Revisión de documentos en preparación para la auditoría.

Se ha realizado una revisión detallada de todos los documentos de la aplicación TextSecure a los que tenemos acceso. En la página: https://whispersystems.org/ podemos encontrar gran parte de la documentación externa de la aplicación.

Por otro lado en github hemos encontrado documentación interna del producto junto con el código fuente.

Se ha encontrado algún vacío en cuanto a la documentación publicada en internet. Por ejemplo no hemos podido encontrar el análisis funcional del programa o los diagramas de secuencia. Esto es un aspecto a tener en cuenta ya que estos documentos pueden afectar a la hora de analizar la mantenibilidad de la aplicación, si se tratase de una auditoría donde tuviéramos contacto con los auditados en este punto deberíamos solicitar esta documentación al líder del equipo auditado.

### 2.2. Preparación del plan de auditoría.

El plan de auditoría de mantenibilidad consiste en analizar TextSecure, viendo en qué medida se cumplen las métricas definidas en el punto 3. Nos estamos basando en el proceso de auditoría de mantenibilidad definido en el apartado anterior, por ello estamos desarrollando cada actividad del proceso de auditoría en este caso práctico.

Una vez que tengamos estas evidencias procederemos a realizar un informe detallado de las evidencias de auditoría.

### 2.3. Asignación de trabajo al equipo auditor.

La asignación de tareas en nuestra auditoría es fácil ya que el equipo auditor está formado por el alumno del TFG y sus tutores, por lo que el trabajo de realizar la auditoría lo realizará el alumno y será supervisado por sus tutores.

### 2.4 Preparación de los documentos de trabajo.

Los documentos de trabajo consisten en toda la documentación a la que tengamos alcance y el código fuente de la aplicación. Además se han preparado las métricas con las que vamos a auditar la aplicación. Estos documentos son los que vamos a seguir en el apartado de realización de las actividades de auditoría.

## 3. Realización de actividades de auditoría.

Para analizar esta aplicación vamos a apoyarnos en las métricas que hemos definido en el punto 3. Veremos en qué medida esta aplicación cumple con las métricas de mantenibilidad definidas. Para ello vamos a analizar cada una de las sub-métricas de

la mantenibilidad definidas a lo largo de esta memoria. Por último vamos a elaborar un informe de auditoría con los resultados obtenidos en la evaluación de la mantenibilidad.

### 3.1 Establecer los requisitos de evaluación para la mantenibilidad.

Para que TextSecure sea mantenible ha de cumplir los requisitos de mantenibilidad que son las métricas que hemos definido en la tabla 1 del apartado 3 de esta memoria. Es necesario que se respeten cada una de las sub-dimensiones de la mantenibilidad, que son la analizabilidad, modificabilidad, capacidad de ser probado y reusabilidad.

El análisis de los requisitos se va a realizar en función de la información obtenida al ejecutar SonarQube y VerifySoft en TextSecure, a partir de estos resultados evaluaremos en qué medida se respetan las métricas definidas.

Además se ha de analizar la documentación disponible ya que la documentación es un requisito importante de mantenibilidad.

### 3.2. Especificar y diseñar la evaluación.

El alumno de este TFG va a realizar la evaluación de la mantenibilidad en la herramienta SonarQube. Para ello va a realizar un informe detallado donde se describirán los resultados obtenidos con SonarQube para cada métrica de mantenibilidad obtenida a partir de las normas ISO/IEC 25000 y 9126. Se van a interpretar estos resultados en función de los umbrales descritos en cada métrica. Al tratarse de una auditoría de mantenibilidad no se van a estudiar los resultados obtenidos para otra dimensión de la calidad software que no sea la mantenibilidad.

Para diseñar la evaluación el equipo auditor se ha reunido periódicamente durante el tiempo que ha durado este proceso de auditoría para definir los requisitos de manteniblidad y analizar los resultados que hemos ido obteniendo del análisis de la mantenibilidad. Durante esta comunicación se han realizado cambios en el plan de auditoría como descartar el uso de las métricas de 25023 por las del 9126 ya que su disponibilidad era mayor.

### 3.3. Ejecutar la evaluación.

Esta es la actividad principal del proceso de auditoría de mantenibilidad ya que en este apartado vamos a realizar el análisis detallado del resultado de ejecutar la evaluación de cada métrica de mantenibilidad definidas en el apartado 3.

La iremos dividiendo según las sub-dimensiones de mantenibilidad:

## 4.1 Analizabilidad

#### Complejidad Ciclomática

SonarQube al analizar nuestro código para calcular la complejidad ciclomática mantiene un contador. Cuando el flujo de una función se altera, es decir, se produce un salto, este contador de la complejidad aumenta en 1. Cada función tiene como mínimo una complejidad de 1, aunque no hayamos programado nada en ella (se suma 1 por la llamada al método).

En JAVA por ejemplo, estas son las reglas para medir la complejidad:

- Estas palabras incrementan la complejidad: "*if, for, while, case, catch, throw, return (that is not the last statement of a method), &&, ||, ?"*
- *"Else, default"* no incrementan la complejidad.

Un método simple con una sentencia switch y un enorme bloque de sentencias de casos puede tener un valor sorprendentemente alto de complejidad.

Ejemplo: el siguiente método tiene una complejiad de 5.

```
public void process(Car myCar){         // +1
if(myCar.isNotMine()){              // +1
return;                     // +1
}
car.paint("red");
car.changeWheel();
while(car.hasGazol() && car.getDriver().isNotStressed()){   // +2
car.drive();
}
return;
```

SonarQube diferencia tres tipos de complejidad: la de método, la de clase, la de fichero y la total.

- **Complejidad/método:** es la media de la complejidad por método, es decir, la suma de la complejidad/método de cada clase dividido entre el número de métodos totales de nuestro software.

- **Complejidad/clase:** es la media de la complejidad de todas las clases, es decir, la suma de la complejidad ciclomática total de cada clase (obtenida como la suma de la complejidad de cada uno de sus métodos), dividida entre el número de clases.
- **Complejidad/fichero:** es la media de la complejidad por fichero.
- **Complejidad total (Total):** es la complejidad ciclomática total del sistema (es decir, el sumatorio de la complejidad ciclomática de todos los métodos de nuestro software)

En el panel de control del proyecto TextSecure podemos ver la complejidad de este proyecto.

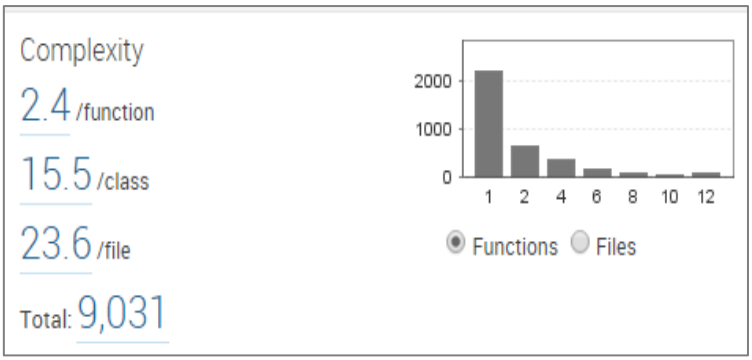


**Figura 19: Complejidad ciclomática TextSecure**

Como definimos en las métricas, la densidad de la complejidad ciclomática se calcula a partir de la siguiente fórmula: CCD = Aristas – Nodos + 2,

Como se puede ver en la imagen anterior la complejidad ciclomática del proyecto es de 9,031. Además podemos ver el desglose de la complejidad por funciones, clases y ficheros.

| Complejidad Ciclomática | Evaluación del Riesgo |
|---|---|
| **1-10** | Programa Simple, sin mucho riesgo |
| **11-20** | Más complejo, riesgo moderado |
| **21-50** | Complejo, programa de alto riesgo |
| **50** | Programa no testeable, muy alto riesgo |

**Tabla 8: Clasificación complejidad ciclomática TextSecure**

Apoyándonos en la tabla que definimos en la métrica de complejidad ciclomática, vemos que la complejidad del proyecto TextSecure(9,031) indica que nuestro programa es simple y sin mucho riesgo.

Para analizar la complejidad ciclomática en detalle podemos ver uno a uno la complejidad de cada fichero JAVA de nuestro proyecto. En la siguiente imagen se puede ver la complejidad del fichero: ArrayListCursor.

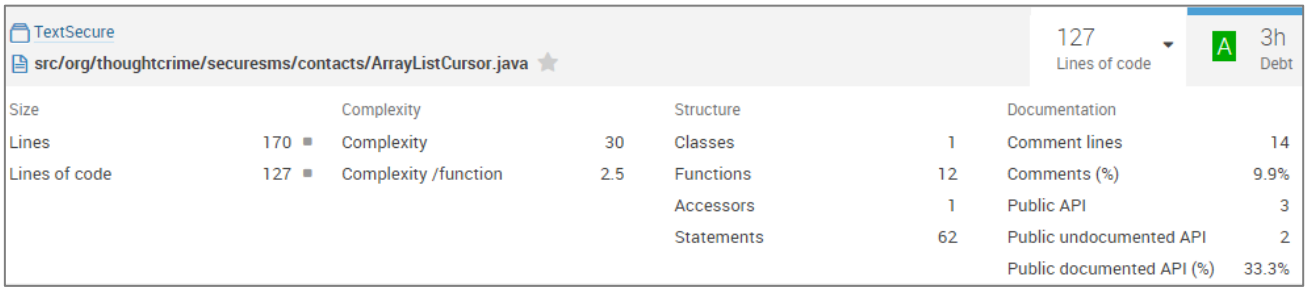


**Figura 20: Complejidad ciclomática ArrayListCursor**

Como se puede ver la complejidad en este fichero es más elevada que la complejidad media del proyecto.

### Densidad de código Repetido

El código repetido es una métrica básica de calidad, y junto con la complejidad ciclomática, es el mayor enemigo de la mantenibilidad.

En la siguiente figura se puede ver el porcentaje de código repetido en el proyecto TextSecure:

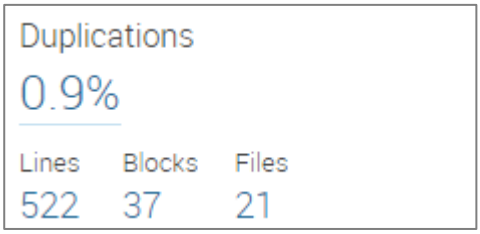


**Figura 21: Código duplicado TextSecure**

Para calcular la densidad de código repetido SonarQube aplica la fórmula definida en el apartado de la métrica de código repetido:

*DCR=Líneas duplicadas/Líneas totales *100*

Como se puede ver, solo el 0.9% de las líneas de código están duplicadas, este es otro aspecto que favorece claramente la mantenibilidad del producto software ya que un alto porcentaje de código repetido:

– Aumenta innecesariamente el número de líneas de código (a más líneas de código más complejo es el mantenimiento, y más costoso).

– Dispara los costes (si hay que cambiar ese código repetido… hay que cambiarlo en muchos sitios).

– Aumenta los riesgos (hay que buscar todas las repeticiones y si se nos olvida cambiarlo en algún sitio… el software acaba siendo incoherente).

### Densidad de Comentarios

SonarQube calcula el porcentaje de líneas comentadas del proyecto, así como el número total de comentarios.

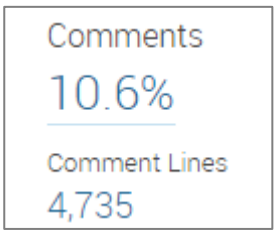


**Figura 22: Porcentaje de comentarios TextSecure**

La forma de calcular la densidad de comentarios, como definimos en el apartado de métricas, es bastante sencilla:

*COMD=Líneas comentadas/Líneas totales * 100*

El porcentaje de comentarios del proyecto es de 10,6%. Este no es un número muy elevado de comentarios, la ausencia de comentarios supone un problema de mantenibilidad ya que es más costoso entender el código fuente. Sin embargo un 10,6% no supone un problema crítico ya que un gran número de líneas contienen comentarios explicativos, para hacernos una idea, si el valor fuese de un 50%, el número de líneas de código es igual al número de líneas comentadas. Si el valor fuese de un 100%, significa que el proyecto solo contiene comentarios.

### Densidad de defectos de la capacidad de ser analizado

Los defectos de la capacidad de ser analizado un programa se calculan a partir de los defectos que consideremos importantes a la hora de medir la analizabilidad de nuestro software. En nuestras métricas definimos los siguientes puntos:

- **Ausencia de comentarios.** Como hemos dicho anteriormente hay un 10.6% de líneas comentadas en este proyecto. Es decir no es un número muy elevado pero tampoco muy reducido. Si todas las líneas estuvieran comentadas habría un 50% de líneas comentadas, aunque no es necesario comentar cada línea, es decir el 40-50% de líneas comentadas sería el valor óptimo, al haber un 10,6% definimos la ausencia de comentarios con un valor de 8 sobre 10.

| % de comentarios | Valor (0-10) |
|---|---|
| **1-10%** | 9-10 |
| **10-20%** | 7-8 |
| **20-30%** | 5-6 |
| **30-40%** | 1-4 |
| **40-50%** | 0 |

**Tabla 9: Clasificación comentarios TextSecure**

El peso de la ausencia de comentarios, como se definió en el apartado de métricas, es de 10 sobre 100 debido a que hay aspectos más importantes a la hora de evaluar los defectos de la capacidad de ser analizado un programa.

- **Elevada complejidad ciclomática**. La complejidad ciclomática del proyecto es de 9,031 que como hemos dicho anteriormente está por debajo de 10 y significa que la complejidad ciclomática es muy buena, por tanto aplicando la siguiente tabla daremos un valor de 2 sobre 10.

| Complejidad ciclomática | Evaluación de riesgo | Valor (0-10) |
|---|---|---|
| **1-10** | Programa simple sin mucho riesgo | 0-2 |
| **11-20** | Más complejo, riesgo moderado | 3-5 |
| **21-50** | Complejo, programa de alto riesgo | 6-8 |
| **50** | Programa no testeable, muy alto riesgo | 9-10 |

**Tabla 10: Clasificación complejidad ciclomática TextSecure**

El peso de la complejidad ciclomática, como se explicó en el apartado de las métricas, es de 40 sobre 100, debido a que es un factor fundamental a la hora de analizar un programa.

- **Incoherencias en el código**. Para medir esta característica nos fijamos en el "SQALE RATING" que se mide de la A a la E.

| Clasificación SQALE | Resultado |
|---|---|
| **A** | Muy buena |
| **B** | Buena |
| **C** | Media |
| **D** | Mala |
| **E** | Muy mala |

**Tabla 11: SQALE RATING sonarqube**

Como se ve en la imagen el ratio de TextSecure es A, esto quiere decir que tiene un valor entre 0 y 1 por lo que otorgamos 0,5.

Por otro lado apenas hay aspectos críticos o bloqueantes en el código, pero si hay un elevado número de "problemas Mayores" como se puede ver en la siguiente figura, por lo que otorgamos un valor de 4 sobre 10.

El peso de ambas características es de 0,5 ya que ambas tienen la misma importancia a la hora de analizar las incoherencias en el código. Por lo que si hacemos los cálculos obtenemos que el

valor final es: 0,5 * 0,5 + 0,5 * 4= 2,25

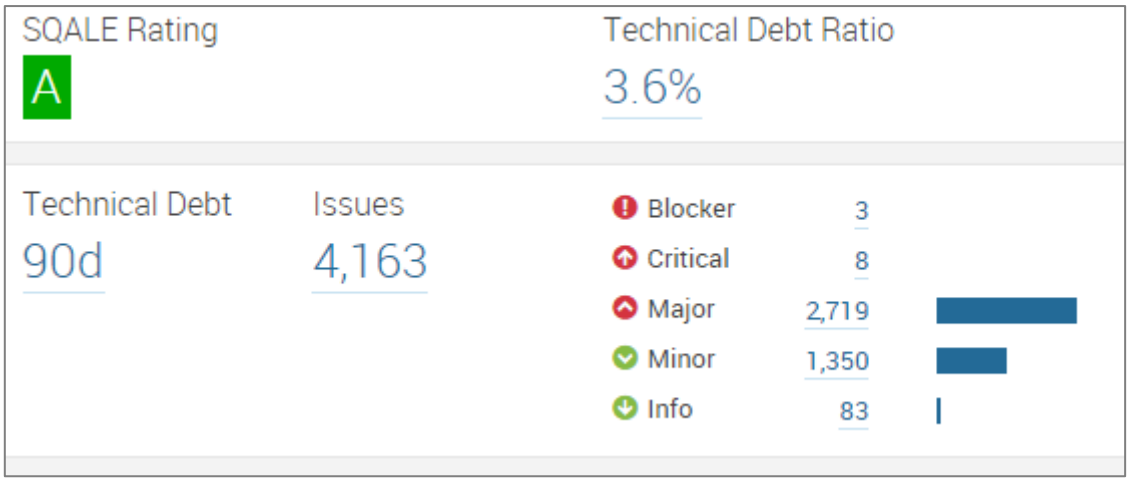


**Figura 23: Deuda técnica TextSecure**

El peso de las incoherencias en el código, como se explicó en el apartado de métricas, es de 30 sobre 100 debido al coste que estos errores suponen a la hora de analizar el código.

- **Código Repetido**. Como vimos solo el 0,9% de las líneas son repetidas. El valor de código repetido si aplicamos la siguiente tabla es de 1 sobre 10 y el peso de este elemento es de 20 sobre 100 ya que el código repetido supone un gran coste a la hora de analizar el software.

| % código repetido | Valor (0-10) |
|---|---|
| **0-10%** | 0-2 |
| **10-30%** | 2-4 |
| **30-50%** | 5-6 |
| **50%-80%** | 7-9 |
| **80%-100%** | 10 |

**Tabla 12: Clasificación código repetido TextSecure**

Por tanto si aplicamos la fórmula para calcular la densidad de defectos de la analizabilidad y dividimos los valores entre 100 para obtener una clasificación comprendida del 0 al 100 obtenemos los siguientes resultados:

10**0,8 + 40*0,2 + 30 *0,225 + 20*0,1 = 24,75 sobre 100.*

Si aplicamos la tabla definida en el apartado de las métricas la densidad de defectos de la capacidad de ser analizado es muy buena aunque está próxima a ser muy buena, el principal defecto como se puede ver es la ausencia de comentarios o las incoherencias en el código.

| Métrica | Resultado |
|---|---|
| **Compejidad ciclomática** | Muy Buena |
| **Densidad de comentarios** | Media |
| **Código duplicado** | Muy Buena |
| **Densidad de defectos de la capacidad de ser analizado** | Muy Buena |

**Tabla 13: Resumen analizabilidad TextSecure**

## 4.2 Modificabilidad

### Documentación

Documentación Interna

La documentación de las APIs públicas es importante sobre todo para aquellos que pretendan extender, reutilizar o interactuar con la aplicación.

Con esta métrica medimos el porcentaje de la documentación que está publicada para todos aquellos que deseen entender nuestro programa. Sonarqube nos muestra:

- API pública: Número de clases públicas, métodos públicos (sin contar métodos de acceso como get y set) y propiedades públicas (sin contar aquellas que son de la forma public static). Como se puede ver hay 2.465 clases, métodos y propiedades públicas.

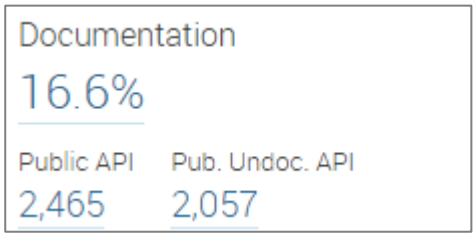


**Figura 24: Documentación TextSecure**

- API pública sin documentar: Número de APIs públicas sin una cabecera de comentarios o sin un Javadoc. Hay un total de 2057 APIs públicas sin comentar.

En total se ve que solo un 16,6% del proyecto está documentado. Esto, si lo clasificamos según la tabla descrita en la métrica de documentación, es un porcentaje muy bajo que hace que el coste de la modificabilidad crezca.

Documentación Externa

La documentación externa está orientada para ser leída incluso por quienes tal vez nunca verán el código real. Además aporta una explicación más amplia que los comentarios acerca de una aplicación.

TextSecure es una aplicación de software libre que está a disposición de todos en Github. Además en la propia página de Github encontramos bastante documentación del proyecto. Este proyecto ha sido creado por WhisperSystems una amplia comunidad de programadores y contienen página web propia con foros de discusión acerca de esta APP. En resumen podemos encontrar una gran cantidad de documentación externa pública en internet  a excepción del análisis funcional de la aplicación. Este aspecto será

reseñado en el informe de auditoría ya que el análisis funcional es un documento importante a la hora de analizar la mantenibilidad de la aplicación.

### Utilizar nombres con sentido

Para que un programa pueda ser mantenido con el menor coste posible es necesario que pueda ser entendido fácilmente incluso por las personas que no han creado el programa. Para ello es fundamental que el nombre que asignamos a las variables sea consecuente con la función de dichas variables.

Después de analizar el código de TextSecure hemos llegado a la conclusión de que en términos generales el nombre de las variables se ajusta a su función, en la siguiente imagen se ve claramente como los nombres de las variables no son aleatorios sino que se puede identificar fácilmente el contenido de ellas.

```
PartDatabase       partDatabase = DatabaseFactory.getPartDatabase(context);
SQLiteDatabase     database     = databaseHelper.getReadableDatabase();
MasterCipher       masterCipher = new MasterCipher(masterSecret);
```

**Figura 25: Ejemplo variables TextSecure**

Aunque no siempre es así hay variables para las cuales es más difícil imaginar su función solo con leer el nombre de estas, por ejemplo la variable “cc”:

```
String cc = new String(encodedCc.getTextString(), CharacterSets.MIMENAME_ISO_8859_1);
```

**Figura 26: Variable cc TextSecure**

En conclusión y tras un análisis detallado del nombre de las variables, se puede decir en la mayoría de variables, aproximadamente un 80%, su nombre se ajusta a su función

### Densidad de defectos de la capacidad de ser modificado

Estos se calculan a partir de los defectos que hemos considerado importantes para reducir los errores a la hora de cambiar nuestro software. En nuestras métricas definimos los siguientes puntos:

- **Acoplamiento entre clases**. Para medir esta característica nos fijamos en el "SQALE RATING" que se mide de la A a la E.

| Clasificación SQALE | Resultado |
|---|---|
| **A** | Muy buena |
| **B** | Buena |
| **C** | Media |
| **D** | Mala |
| **E** | Muy mala |

**Tabla 14: SQALE RATING sonarqube**

Como se ve en la imagen el ratio de TextSecure es A, esto quiere decir que tiene un valor entre 0 y 1 por lo que otorgamos 0,5. El peso de esta característica a la hora de medir el acoplamiento entre clases es de 0,2 debido a que no es tan importante como la deuda técnica que proviene de la modificabilidad.

Además tenemos que analizar que la deuda técnica para la modificabilidad que está relacionada con el acoplamiento entre clases es de 15 días sobre 39 como se ve en la imagen, por lo que otorgamos un valor de 4 sobre 10. El peso de esta característica al ser más importante que la anterior es de 0,8.


**Figura 27: Deuda técnica modificabilidad TextSecure**

Por lo que el valor otorgado para el acoplamiento entre clases es de 0,5*0,2 + 4 * 0,8 = 3,3 sobre 10

El peso del acoplamiento entre clases es de 30 sobre 100 como se definió en el apartado de métricas.

- **Incoherencias en el código.** Para medir esta característica nos fijamos en el "SQALE RATING" que se mide de la A a la E.

| Clasificación SQALE | Resultado |
|---|---|
| **A** | Muy buena |
| **B** | Buena |
| **C** | Media |
| **D** | Mala |
| **E** | Muy mala |

**Tabla 15: SQALE RATING SonarQube**

Como se ve en la imagen el ratio de TextSecure es A, esto quiere decir que tiene un valor entre 0 y 1 por lo que otorgamos 0,5.

Por otro lado apenas hay aspectos críticos o bloqueantes en el código, pero si hay un elevado número de "problemas Mayores" como se puede ver en la siguiente figura, por lo que otorgamos un valor de 4 sobre 10.

El peso de ambas características es de 0,5 ya que ambas tienen la misma importancia a la hora de analizar las incoherencias en el código. Por lo que si hacemos los cálculos obtenemos que

el valor final es: 0,5 * 0,5 + 0,5 * 4= 2,25

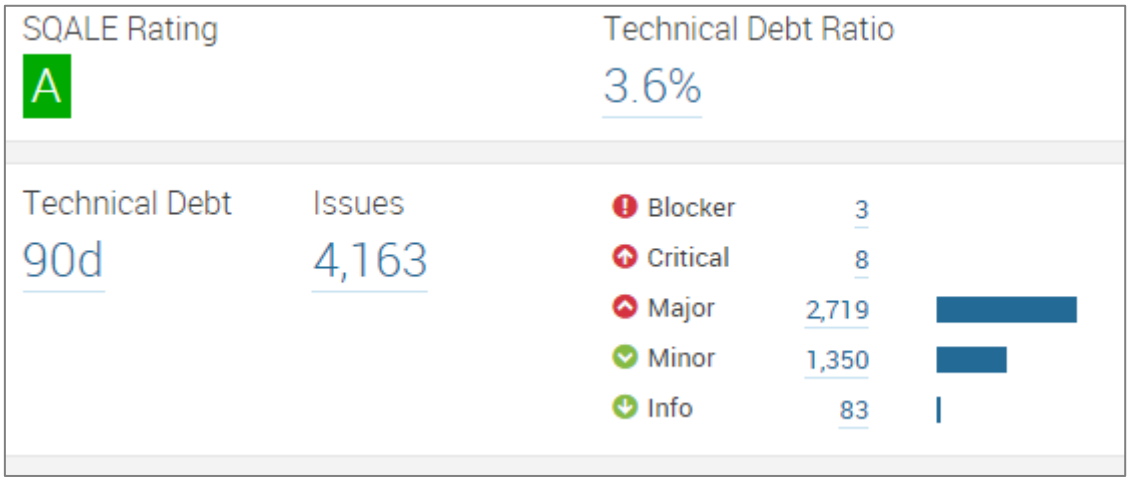


**Figura 28: Deuda técnica TextSecure**

El peso de las incoherencias en el código es de 30 debido al coste que estos errores suponen a la hora de analizar el código.

- **Código Repetido.** Como vimos solo el 0,9% de las líneas son repetidas.

El valor de código repetido si aplicamos la siguiente tabla es de 1 sobre 10 y el peso de este elemento es de 25 sobre 100 ya que el código repetido supone un gran coste a la hora de analizar el software.

| % código repetido | Valor (0-10) |
|---|---|
| **0-10%** | 0-2 |
| **10-30%** | 2-4 |
| **30-50%** | 4-6 |
| **50%-80%** | 6-8 |
| **80%-100%** | 8-10 |

**Tabla 16: Clasificación código repetido TextSecure**

- **Ausencia de comentarios**. Como hemos dicho anteriormente hay un 10.6% de líneas comentadas en este proyecto. Es decir no es un número muy elevado pero tampoco muy reducido. Si todas las líneas estuvieran comentadas habría un 50% de líneas comentadas, aunque no es necesario comentar cada línea, es decir el 40-50% de líneas comentadas sería el valor óptimo, al haber un 10,6% definimos la ausencia de comentarios con un valor de 8 sobre 10.

| % de comentarios | Valor (0-10) |
|---|---|
| **1-10%** | 9-10 |
| **10-20%** | 7-8 |
| **20-30%** | 5-6 |
| **30-40%** | 1-4 |
| **40-50%** | 0 |

**Tabla 17: Clasificación comentarios TextSecure**

El peso de la ausencia de comentarios es de 15 sobre 100 debido a que hay aspectos más importantes a la hora de evaluar los defectos de la capacidad de ser cambiado un programa.

Por tanto si aplicamos la fórmula para calcular la densidad de defectos de la capacidad de ser cambiado obtenemos

los siguientes resultados: 30*0,33 + 30*0,225 + 25 * 0,1 + 15 * 0,8 = 31,15 sobre 100.

Si aplicamos la tabla definida en el apartado de las métricas la densidad de defectos de la capacidad de ser cambiado es mala como se ha detectado en sonarqube.

| Métricas | Resultados |
|---|---|
| **Documentación** | Mala |
| **Nombres Variables** | Buena |
| **Densidad de defectos de la capacidad de ser cambiado** | Buena |

**Tabla 18: Resumen modificabilidad TextSecure**

## 4.3 Capacidad de ser probado

### Nombre de las variables

Para que un programa pueda ser mantenido con el menor coste posible es necesario que pueda ser entendido fácilmente incluso por las personas que no han creado el programa. Para ello es fundamental que el nombre que asignamos a las variables sea consecuente con la función de dichas variables.

Después de analizar el código de TextSecure hemos llegado a la conclusión de que en términos generales el nombre de las variables se ajusta a su función, en la siguiente imagen se ve claramente como los nombres de las variables no son aleatorios sino que se puede identificar fácilmente el contenido de ellas.

```
PartDatabase      partDatabase = DatabaseFactory.getPartDatabase(context);
SQLiteDatabase    database     = databaseHelper.getReadableDatabase();
MasterCipher      masterCipher = new MasterCipher(masterSecret);
```

**Figura 29: Ejemplo nombre variables TextSecure**

Aunque no siempre es así hay variables para las cuales es más difícil imaginar su función solo con leer el nombre de estas, por ejemplo la variable “cc”:

```
String cc = new String(encodedCc.getTextString(), CharacterSets.MIMENAME_ISO_8859_1);
```

**Figura 30: Ejemplo 2 nombre variables TextSecure**

En conclusión podemos decir que por norma general el nombre de las variables corresponde con su función en el programa.

### Tamaño del programa

El tamaño del programa es una característica muy importante a la hora de probar nuestro proyecto. Cuanto mayor sea un programa mayor será el coste de las pruebas, aunque hay que tener en cuenta el resto de métricas.

TextSecure es un programa bastante amplio, en la siguiente imagen se puede ver el tamaño total:

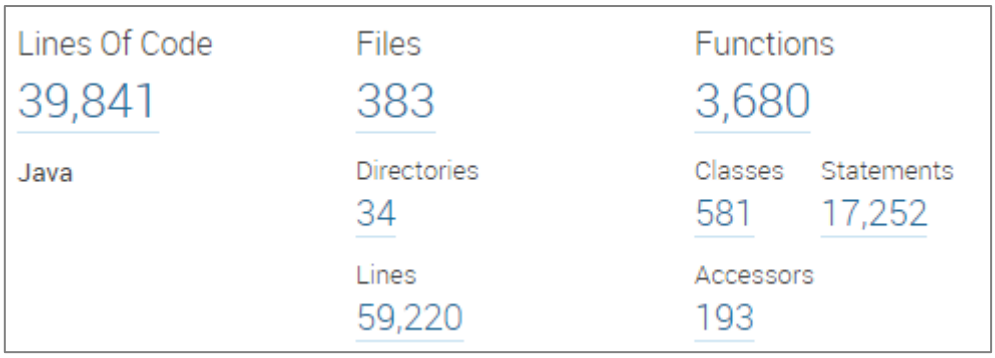


**Figura 31: Tamaño TextSecure**

Hay cerca de 60.000 líneas escritas, de ellas unas 40.000 son líneas de código (sin comentarios) estas están organizadas en 383 ficheros y 34 directorios.

Por su parte hay 3.680 funciones y 581 clases.

Estos resultados son bastante parecidos a los obtenidos con la herramienta verifysoft, pero como podemos ver hay alguna diferencia.

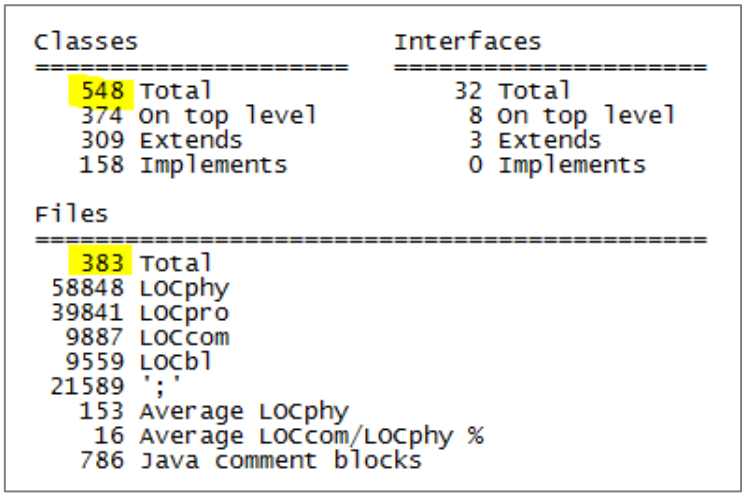


**Figura 32: Tamaño TextSecure VerifySoft**

El elevado número de funciones, clases y líneas de código hace que las pruebas sean más costosas. En la siguiente tabla se puede apreciar los resultados de la capacidad de ser probado.

| Métrica | Resultados |
|---|---|
| **Nombre de las variables** | Buena |
| **Tamaño del programa** | Medio |

**Tabla 19: Clasificación capacidad de ser probado TextSecure**

## 4.4 Reusabilidad

### Estructura del programa

Para que nuestro código pueda ser probado como es debido, es necesario que su estructura sea correcta, como definimos en las métricas del apartado 3 debemos respetar la estructura del programa. En el proyecto SonarQube no hemos detectado ninguna incoherencia ni saltos en el código, además ningún error que genere deuda técnica está relacionado con problemas en la estructura del programa.

Sin embargo, como se puede ver en SonarQube, la modificabilidad tiene una deuda técnica más alta que el resto de características, esto es debido a que el programa no está muy bien construido y hay poco acoplamiento entre clases, entre otros problemas. Esto no es un buen indicador de que la estructura del programa sea correcta. Por lo que en conclusión se debería revisar la estructura del programa, aunque no está mal diseñado para el tamaño que tiene, la deuda técnica de la modificabilidad es de solo 15 días.

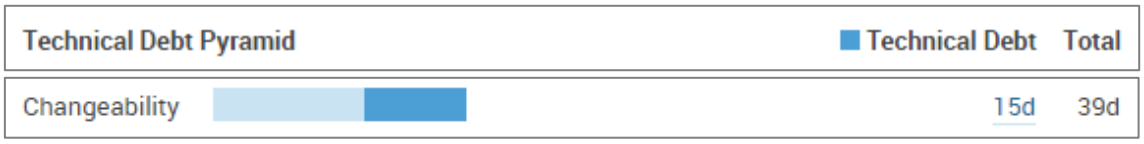


**Figura 33: Deuda técnica modificabilidad TextSecure**

### Deuda técnica

Como definimos anteriormente la deuda técnica es el coste y los intereses a pagar "por hacer mal las cosas", entendiendo hacer mal las cosas como la mala práctica a la hora de diseñar e implementar la aplicación. El sobre esfuerzo a pagar para mantener un producto software mal hecho. Por lo tanto es una característica muy importante a la hora de evaluar la estabilidad de un proyecto.

Una de las características más importantes de SonarQube es que nos calcula la deuda técnica de nuestro proyecto. Como se ve en la siguiente imagen el ratio de deuda técnica es solamente de un 3.6% que es un número muy bajo. Además el "SQALE RATING" que le da SonarQube es el mejor (A), como se ve en la imagen.

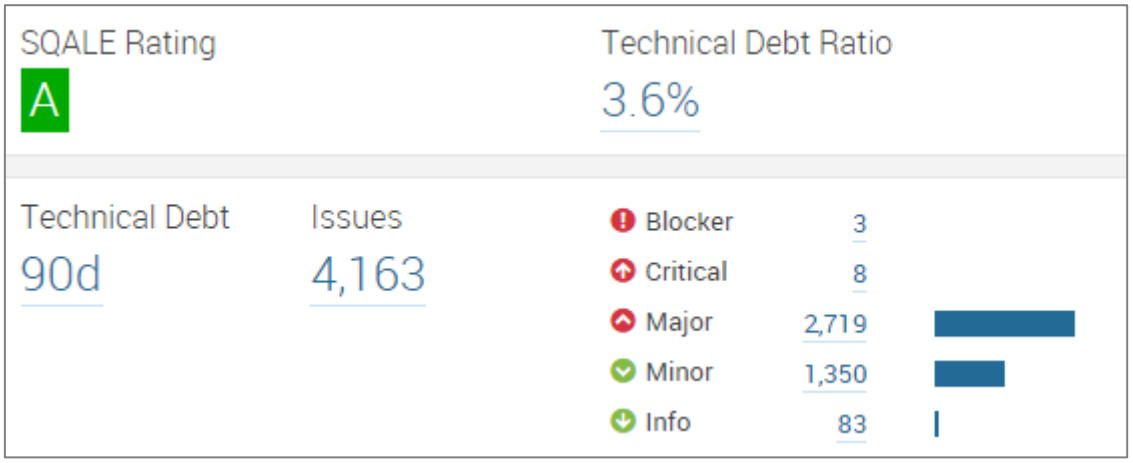


**Figura 34: Deuda técnica TextSecure**

En la siguiente imagen se puede ver el desglose por cada característica de calidad de la deuda técnica.

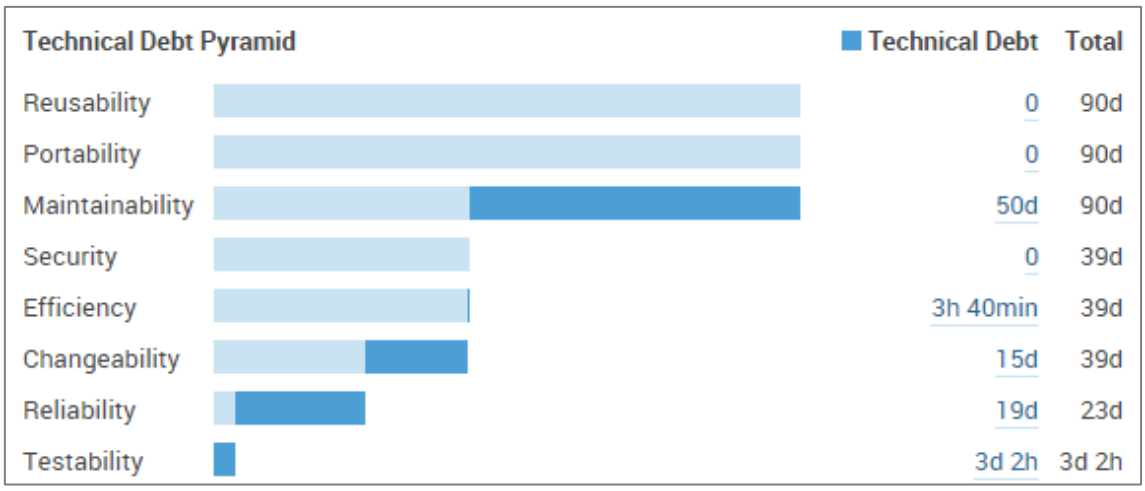


**Figura 35: Pirámide deuda técnica TextSecure**

Como se puede ver la característica que tiene un coste más alto de deuda técnica es la mantenibilidad, la característica que estamos analizando en nuestro proceso de auditoría. Un total de 50 días sobre los 90 totales. Esto es debido a que la mantenibilidad es una de las características más costosas de un proyecto. Pese a ello 50 días es un número bajo para el tamaño de nuestro proyecto.

Por su parte VerifySoft ha determinado un índice de mantenibilidad con valor de 140, si comparamos este resultado con la tabla que se encuentra en su página web oficial vemos que el índice de mantenibilidad es muy bueno:

| Measure | Measured | Alarmed | % | Limits 1st level class | Limits 2nd level class | Limits method |
|---|---|---|---|---|---|---|
| v(G) | 4445 | 49 | 1 | 1-100 | 1-100 | 1-10 |
| LOCpro | 4445 | 84 | 1 | 4-400 | 4-400 | 1-40 |
| Comment % | 4445 | 1263 | 28 | 20-60 | 20-60 | 20-60 |
| V | 4445 | 254 | 5 | 100-8000 | 100-8000 | 4-1000 |
| B | 4445 | 42 | 2 | 0-2 | 0-2 | n/a |
| MI | 4445 | 140 | 3 | 80- | 80- | 80- |
| Total | 26670 | 1832 | 6 | | | |

**Figura 36: Deuda técnica VerifySoft**

| Medida | Resultado |
|---|---|
| 85 and more | Buena mantenibilidad |
| 65-85 | Moderada mantenibilidad |
| <65 | Difícil de mantener |

**Tabla 20: Clasificación deuda técnica VerifySoft**

En conclusión se puede decir que el coste de deuda técnica es pequeño, pese a tener 90 días esto no es considerado mucho tiempo para un proyecto tan grande como el que estamos analizando.

**Densidad de defectos de Reusabilidad**

La densidad de defectos de reusabilidad mide la relación entre el número de defectos de reusabilidad que presenta el proyecto. En el apartado 3 definimos la característica más influyente a la hora de analizar esta métrica.

- **Usar excepciones en el código.** El uso de excepciones es importante a la hora de programar ya que permite al programador controlar los errores ocasionados durante la ejecución de un programa. Cuando ocurre cierto tipo de error, el sistema reacciona ejecutando un fragmento de código que resuelve la situación, por ejemplo retornando un mensaje de error o devolviendo un valor por defecto. TextSecure maneja las excepciones a lo largo de todo su código, sin embargo, SonarQube ha detectado errores críticos en el uso de las excepciones ya que estas no están bien definidas, como se puede ver en la siguiente imagen, SonarQube no entiende la variable “Throwable” y avisa de este error, ya que unas excepciones mal controladas puede suponer que se detenga nuestro programa durante su ejecución.

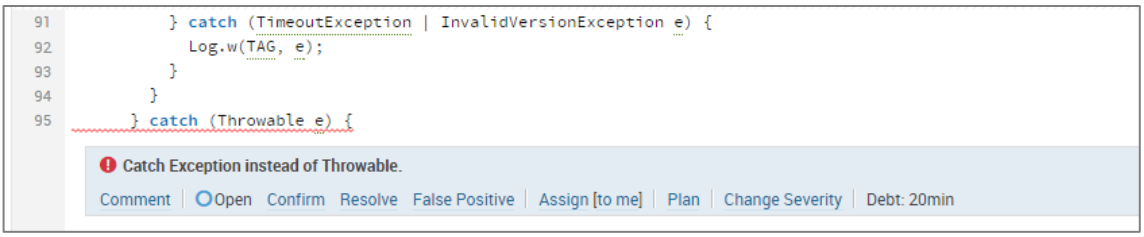


**Figura 37: Error excepción TextSecure**

Por tanto daremos un valor de 3 sobre 10 al uso de excepciones, el peso de esta característica se definió de 100 sobre 100.

Por tanto si aplicamos la fórmula para calcular la densidad de defectos de la analizabilidad obtenemos los siguientes resultados:

100 *0,3 = 30 sobre 100.

Si aplicamos la tabla definida en el apartado de las métricas la densidad de defectos de estabilidad es muy buena como se ha detectado en sonarqube.

| Métricas | Resultados |
|---|---|
| Deuda Técnica | Muy Buena |
| Estructura del programa | Buena |
| Densidad de defectos reusabilidad | Buena |

**Tabla 21: Clasificación estabilidad TextSecure**

## 4.5 Concluir la evaluación de mantenibilidad y preparar el informe de auditoría

Una vez que hemos ejecutado la evaluación de mantenibilidad debemos de interpretar los resultados obtenidos y elaborar el informe de auditoría.

Este reporte se encuentra en el Anexo 2 de esta memoria.

## 4.6 Distribución del informe de auditoría

El informe de auditoría no será distribuido al equipo auditado ya que se trata de un TFG donde no se solicitó una auditoría real por parte de los responsables del proyecto TextSecure. Sin embargo se distribuirá para la evaluación del TFG dentro del plazo estipulado al inicio del curso.

## 5 Finalización de la auditoría

Para finalizar la auditoría de TextSecure hemos elaborado varias reuniones entre los integrantes del equipo auditor para revisar los documentos obtenidos y el informe de auditoría.

Además se debería tratar con el equipo auditado los hallazgos de la auditoría y reseñar aquellos apartados que han supuesto más coste de mantenibilidad con el fin de trabajar sobre ellos.

Las conclusiones de la auditoría fueron descritas en el informe de auditoría y se resumen en que la aplicación TextSecure ha pasado la auditoría de mantenibilidad debido a su buen diseño y cumplimiento de las métricas descritas para la mantenibilidad del software.

Sin embargo es importante revisar ciertos aspectos del programa como el código repetido, el tratamiento de excepciones, y los errores en el código que hacen que aumente la deuda técnica.

Los documentos pertenecientes a la auditoría deberían conservarse en esta memoria.

## 6 Realización de seguimiento de la auditoría

Por último tras terminar el proceso de auditoría de mantenibilidad de TextSecure tenemos que realizar un seguimiento periódico de esta aplicación.

En el informe de auditoría se reflejaron los aspectos más importantes que estaban suponiendo un coste elevado en la mantenibilidad, se dieron pautas de trabajo para reducir estos costes. Por ello es importante realizar un seguimiento de los cambios que se vallan a realizar en esta aplicación con el fin de analizar si se están teniendo en cuenta las recomendaciones de mantenibilidad.

Además habrá que volver a auditar la aplicación si esta sufre cambios importantes con el fin de incorporar nuevas funcionalidades.

# 5. Conclusiones, principales aportaciones y líneas abiertas

Este trabajo propone un modelo detallado de auditoría de mantenibilidad que no ha sido descrito hasta ahora. Para la realización ha sido fundamental un análisis detallado de los antecedentes. En este análisis se han estudiado a fondo las normas ISO/IEC 9126, 14598,25000 y 19011. Esta fase fue la clave para entender los procesos de auditoría principales y comprender el concepto de la mantenibilidad que pertenece a la dimensión de la calidad del software.

A partir de este estudió se ha realizado la parte principal de este trabajo de fin de grado que es la elaboración de un modelo de auditoría de mantenibilidad para aplicaciones software. Para ello se ha invertido un gran esfuerzo en la definición de las métricas de mantenibilidad. Durante la realización del caso práctico de auditoría de mantenibilidad se ha montado un completo laboratorio de auditoría, en él se han analizado al detalle las evidencias obtenidas para cada métrica.

Durante el desarrollo de este TFG se ha visto que a día de hoy no existen herramientas para la evaluación de algunas de las métricas descritas (densidad de ciclos, pruebas, documentación externa), por lo que el desarrollo de aplicaciones que puedan evaluar estas métricas queda abierto a futuras investigaciones. Es importante señalar que se han propuesto nuevas métricas para evaluar la mantenibilidad del software, en futuras investigaciones se podría profundizar sobre estas métricas propuestas.

Como se introdujo en el apartado 1 de esta memoria el desarrollo software está en continuo crecimiento y cambio, por lo que el concepto de mantenibilidad también está ligado a este cambio, sería interesante investigar acerca de nuevas métricas relacionadas con los nuevos paradigmas de programación que están surgiendo.

Pasa lo mismo con las metodologías que siguen los equipos de desarrollo, cada vez son más los equipos que aplican las metodologías ágiles a la hora de desarrollar aplicaciones ya que los cambios en las planificaciones son continuos, se podría adaptar el proceso de auditoría de mantenibilidad descrito en este TFG a las nuevas metodologías de desarrollo que cada vez son más utilizadas.

La principal aportación de este proyecto es un proceso específico de auditoría de mantenibilidad para aplicaciones software. Para este proceso de auditoría se detalla cada apartado que lo compone como se puede ver en la siguiente figura.

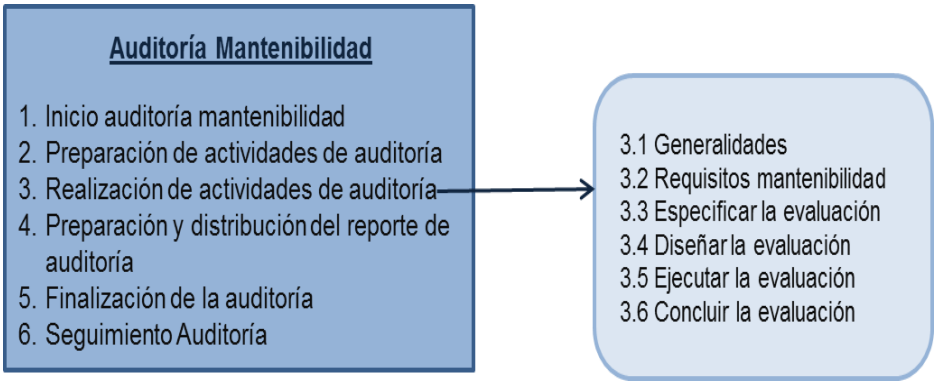


**Figura 38: Fases Auditoría Mantenibilidad**

Además se han definido una serie de métricas para evaluar la mantenibilidad de una aplicación software.

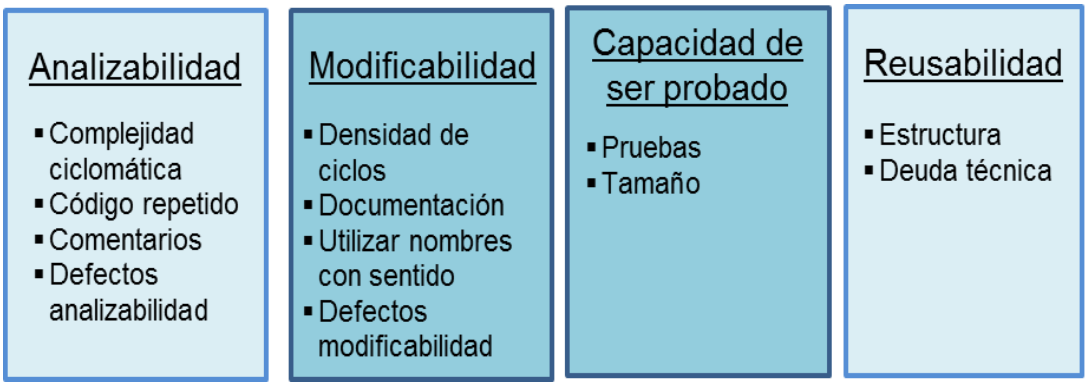


**Figura 39: Métricas Mantenibilidad**

En un proceso de auditoría de mantenibilidad es fundamental desarrollar métricas para obtener evidencias a la hora de auditar un producto software. En base a la experiencia de los tutores y el alumno del TFG se han adaptado métricas ya existentes en estándares como el 9126,25040, la plataforma SonarQube… y se ha diseñado una función de evaluación para dichas métricas.

Además se han desarrollado nuevas métricas definidas por el autor de este TFG como son:

- Análisis de la documentación publicada.
- Importancia del nombre que asignamos a las variables del proyecto.
- Concepto de "Deuda técnica" y su forma de evaluarlo unido al concepto de mantenibilidad.
- Importancia del tamaño y la estructura del programa a la hora de mantenerlo.

- Densidad de defectos para cada una de las sub-características de la mantenibilidad. Para analizar la densidad de defectos de cada sub-característica se ha calculado el peso de cada defecto en función de su importancia y se ha desarrollado su función de medición para poder evaluar el resultado.

El modelo de auditoría de mantenibilidad desarrollado por el alumno de este TFG ha sido puesto en práctica en una aplicación de software libre llamada TextSecure. Esta herramienta ha sido auditada y se ha elaborado un realista informe de auditoría.

Para la realización del caso práctico se ha montado un completo laboratorio de auditoría. Este laboratorio cuenta con las herramientas SonarQube y VerifySoft, las cuales han sido instaladas y se han aprendido a usar. Con estas herramientas se han auditado todas las métricas definidas en el modelo de auditoría y se han redactado los resultados.

Con este proyecto se ha aportado un proceso de auditoría completamente nuevo, para una de las dimensiones más importantes de la calidad, la mantenibilidad. A partir de él se ofrece al campo de desarrollo software y al de calidad una herramienta novedosa y actual para la realización de auditorías de mantenibilidad. Con esta herramienta las organizaciones podrán aumentar la calidad de sus aplicaciones y al mismo tiempo reducir el coste y el de sus desarrollos y mantenimientos.

## Conclusions

This dissertation proposes a detailed model of maintainability audit that has not been described till now. For the accomplishment a detailed analysis of the precedents has been fundamental. This phase was the key to understand the principal processes of audit and to understand the concept of the maintainability that concerns to the software dimension.

After this study, we realized the principal part of this dissertation, the elaboration of an audit model of maintainability for software applications. For it, a great effort has been inverted in the definition of the maintainability metrics.

We completed the dissertation with a practical case, a complete laboratory of audit has been mounted, with it has been analyzed in detail the evidences obtained for every metrics.

During the development of this dissertation we have seen that today don´t exist tools for the evaluation of some metrics described (density of cycles, tests, external documentation), development of applications that could evaluate this metrics remains opened future researches. We have proposed new metrics to evaluate the software maintainability, future investigations could penetrate on these metrics.

As we said in this memory software is in continuous growth and change, and the concept of maintainability also is tied to this change, it would be interesting to investigate over new metrics related to the new programming paradigms of that are arising. It's same with the methodologies that use the developments teams, there are more and more teams that apply the agile methodologies at the moment of develop applications because changes in the planning are constant, adapt the process described in this dissertation to the new development methodologies is another open line.

This dissertation has brought a whole audit process for one of the most important dimensions of quality, maintainability. With it, we offer to the software development field a new and current tool for accomplish a maintainability audit. With this tool the organizations will be able to increase the quality of their applications and at the same time reduces the cost of the developments and maintenances.

# Bibliografía

[1] Roger S. Pressman.Ingeniería del software, un enfoque práctico. Sexta Edición.

[2] Shari Lawrence Pfleeger. Ingeniería del software. Teoría y práctica.

[3] Robert C. Martín Series. Clean Code.A handbook of agile software Craftsmanship.

[4] Thomas, Z., et al., Mining Version Histories to Guide Software Changes, in Proceedings of the 26th International Conference on Software Engineering.

[5] ISO/IEC Standard 9126, International Organization for Standarization.

[6] ISO/IEC Standard 14598, International Organization for Standarization.

[7] ISO/IEC Standard 19011, International Organization for Standarization.

[8] ISO/IEC Standard , International Organization for Standarization.

[9] ISO, ISO/IEC 25000 Software and system engineering – Software product Quality Requirements and Evaluation (SQuaRE) –Guide to SQuaRE, in International Organization for Standarization. 2005: Ginebra, Suiza.

[10] ISO, ISO/IEC 25010 Software and system engineering – Software product Quality

[11] Requirements and Evaluation (SQuaRE) – Quality model and guide. International Organization for Standarization. . 2005: Ginebra, Suiza.

[12] SonarQube Hispano
https://sonarqubehispano.org/

[13] Colexio Profesional de Enxeñaría en Informática de Galicia Fernando Suárez Lorenzo Javier Garzás Parra. I Jornada sobre Calidad del Producto Software e ISO 25000.

[14] Estándares de calidad software blogspot
http://estandarescalidadsoftware.blogspot.com.es/2013/09/iso-14598_13.html

[15] Piattini Velthuis, Mario G, Polo Usaola. Mantenimiento del software. Modelos, técnicas y métodos para la gestión del cambio.

[16] Lehman, M. M. (1996) Laws of Software Evolution Revisited, in Proceedings of the 5th European Workshop on Software Process Technology EWSPT '96, Springer-Verlag.

[17] Garvin, D. (1984) What does product quality really mean?, Sloan Management

[18] Felix Oscar García Rubio, Cristina Vicente Chicote. Desarrollo de software dirigido por modelos.

## Anexo 1: Tablas Resumen TextSecure

Analizabilidad

| Métrica | Resultado |
|---|---|
| Compejidad ciclomática | Muy Buena |
| Densidad de comentarios | Media |
| Código duplicado | Muy Buena |
| Densidad de defectos de la capacidad de ser analizado | Muy Buena |

Modificabilidad

| Métricas | Resultados |
|---|---|
| Documentación | Mala |
| Nombres Variables | Buena |
| Densidad de defectos de la capacidad de ser cambiado | Buena |

Capacidad de ser probado

| Métrica | Resultados |
|---|---|
| Densidad de pruebas | Buena |
| Tamaño del programa | Medio |

Reusabilidad

| Métricas | Resultados |
|---|---|
| Deuda Técnica | Muy Buena |
| Estructura del programa | Buena |
| Densidad de defectos reusabilidad | Buena |

## Anexo 2: Laboratorio de Auditoría

SonarQube es una plataforma para evaluar código fuente. Es un software libre que usa diversas herramientas de análisis estático de código fuente como Checkstyle, PMD o FindBugs para obtener métricas que pueden ayudar a mejorar la calidad del código de nuestros programas. Además, tiene soporte para más de 20 lenguajes de programación entre los que se encuentran Java, C#, C / C++, PL / SQL, Cobol, ABAP, Python, JavaScript…

SonarQube cubre los 7 ejes principales de la calidad del software y una vez analizado un proyecto nos muestra información detallada sobre la arquitectura y el diseño, comentarios de nuestro programa, código duplicado, reglas de programación acordes con el lenguaje que estemos utilizando, bugs potenciales y su posible solución, datos referentes a la complejidad del proyecto. Estos son los 7 ejes principales de la calidad del software:

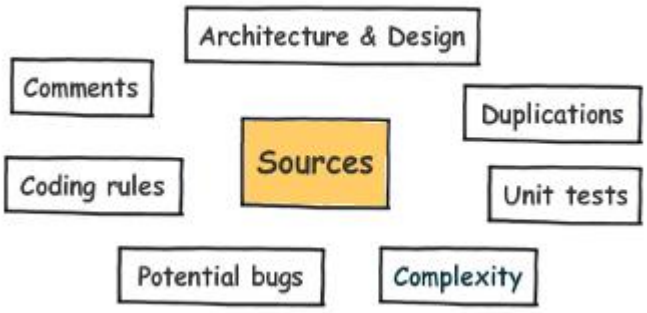


**Figura 40: Esquema ejes principales SonarQube**

SonarQube está pensado para ofrecer un seguimiento a lo largo del desarrollo y/o mantenimiento de un programa informático y apoyar a la mejora continua. Sin embargo, también puede ser utilizado para realizar análisis aislados y obtener informes acerca de nuestros proyectos.

VerifySoft es otra herramienta que nos permite evaluar la calidad de nuestros proyectos. Es necesario solicitar el permiso para obtener una versión de pruebas. Para este TFG hemos solicitado el uso de esta herramienta, en concreto el módulo de JAVA para Windows. La interfaz de verifysoft no es tan amigable como la de sonarqube, ya que para analizar un programa simplemente nos devuelve un fichero .txt. Este fichero contiene la información de las características más importantes a la hora de analizar la calidad de un software. Esta información nos aparece dividida por cada fichero que compone el proyecto y un resumen general con la información del proyecto completo.

## Anexo 3: Informe de Auditoría

### 1. Identificación del informe

Este documento corresponde al informe de auditoría de manenibilidad para la aplicación de envío seguro de mensajes de texto llamada TextSecure.

### 2. Identificación del Cliente

Este informe va destinado a la empresa open whisper systems que es la propietaria de la aplicación TextSecure.

### 3. Objetivos

· Auditar la mantenibilidad de la aplicación TextSecure

· Verificar la aplicación así como toda la documentación relativa a ella, para identificar en qué medida se ajusta a un software mantenible, para ello ha de cumplir las métricas de mantenibilidad definidas.

· Proponer líneas de mejora para la mantenibilidad de la aplicación.

La metodología a seguir será la siguiente:

a. Inspección

b. Análisis

d. Observación

e. Confirmación

### 4. Hallazgos Potenciales

Hallazgos positivos de la auditoría:

**Diseño**

Tras el análisis detallado de la aplicación TextSecure se concluye que el diseño de la aplicación es bueno, reflejo de ello es el poco porcentaje de código duplicado que hemos encontrado y el buen acoplamiento entre clases.

**Claridad del código fuente**

El lenguaje utilizado para el desarrollo de TextSecure es JAVA. El código de la aplicación ha sido correctamente escrito, con un uso apropiado del nombre de las variables y funciones. Además no existen grandes incoherencias en el código y apenas se aprecian saltos en él.

**Estructura del programa**

Se ha visto que el programa está dividido en clases y archivos según las operaciones que realiza, esto favorece la mantenibilidad del software ya que al realizar algún cambio se podrá localizar fácilmente.

**5. Informe**

Observación 1

Falta de documentos que afectan la mantenibilidad del software:

- Diagrama de secuencia
- Análisis Funcional de la aplicación.

No están publicados estos documentos y no hemos podido tener acceso a ellos, esto causa vacíos a la hora de medir la mantenibilidad ya que el análisis funcional es el documento de referencia para realizar posibles modificaciones del programa.

Riesgo

No entender bien la estructura del programa y realizar algún cambio que modifique el funcionamiento de la aplicación. Problemas a la hora del mantenimiento de la aplicación.

Recomendación

Elaborar un buen diagrama funcional para poder ser analizado cada vez que se requiera implementar un cambio en la aplicación o para cualquier labor de mantenimiento.

Observación 2

Se han encontrado 3 errores bloquentes, 8 críticos y 2.700 mayores.

Riesgo

Estos errores pueden suponer un comportamiento inesperado del programa que implique más tiempo y costes de mantenimiento.

Recomendaciones

Depurar la aplicación con el fin de corregir estos errores, con la herramienta SonarQube podemos analizar detalladamente donde se encuentra cada error en el código, además se nos define la causa del problema y sus posibles soluciones.

Observación 3

Hay un 10% de comentarios en el código, esto no es un porcentaje bajo pero hay gran cantidad de código que no está comentado

Riesgo

La ausencia de comentarios supone una mayor complejidad para entender el código fuente de la aplicación.

Recomendaciones

Escribir el máximo número posible de comentarios siempre y cuando estos sean de utilidad.

Observación 4

La deuda técnica para la mantenibilidad de la aplicación TextSecure asciende a 90 días, esto no es un número muy elevado de días para una aplicación tan grande.

Riesgo

La deuda técnica supone un impacto importante a la hora de mantener la aplicación y supone un coste económico y de tiempo para los desarrolladores.

Recomendación

Revisar los errores que suponen el aumento de la deuda técnica con el fin de corregirlos.

**6. Alcance de la auditoria**

El objetivo principal de la auditoría consiste en analizar la mantenibilidad de la aplicación TextSecure, queda fuera del alcance el análisis del resto de dimensiones de la calidad del producto software.

**7. Conclusiones:**

Durante el desarrollo de la Auditoria, se observó que las actividades de desarrollo en la aplicación de TextSecure reflejan el uso de buenas prácticas de programación

La aplicación TextSecure ha pasado la auditoría de mantenibilidad debido a su buen diseño y cumplimiento de las métricas descritas para la mantenibilidad del software.

Sin embargo es importante revisar ciertos aspectos del programa como el código repetido, el tratamiento de excepciones, y los errores en el código que hacen que aumente la deuda técnica.

**8. Recomendaciones**

· Se recomienda revisar el código fuente de la aplicación para eliminar los errores descritos durante la ejecución de auditoría.

· Las consideraciones descritas en la ejecución de auditoría se deberán tener en cuenta para posibles modificaciones futuras de la aplicación con el fin de conservar su capacidad de mantenimiento.

**9. Fecha Del Informe**

Junio 2015

**10. Identificación Y Firma Del Auditor**

Javier Valenciano López, responsable de la ejecución de la auditoría de mantenibilidad para el proyecto TextSecure.