UNIVERSIDAD TECNOLÓGICA ISRAEL

TRABAJO DE TITULACIÓN EN OPCIÓN AL GRADO DE:

INGENIERO EN SISTEMAS DE INFORMACIÓN

TEMA: CALIDAD DEL PRODUCTO EN EL DESARROLLO DE SOFTWARE EN LA UNIVERSIDAD ISRAEL APLICANDO NORMAS ISO.

AUTOR: ISAAC VICENTE CAHUEÑAS CALDERÓN.

TUTOR: MG. WILMER RAMIRO VALLE BASTIDAS.

TUTOR TÉCNICO: MG. TANNIA CECILIA MAYORGA JÁCOME.

AÑO: 2017

DEDICATORIA

El presente trabajo es dedicado a Dios por permitirme culminar una etapa de mi formación académica, a mi esposa, mi madre, hermanos y amigos que han sabido impulsar e inyectar ánimo en cada uno de los retos que se han presentado en el trayecto de mi carrera universitaria.

Isaac Vicente Cahueñas Calderón

AGRADECIMIENTO

Agradezco a Dios por permitirme culminar una etapa de mi formación académica, a mi esposa, que con su apoyo y ánimo ha sabido impulsarme en cada uno de los retos que se han presentado en el trayecto de mi carrera universitaria. A mi madre y hermanos que son de fortaleza y guía.

Isaac Vicente Cahueñas Calderón

RESUMEN

La Universidad Israel al desarrollar software no está enfocada en controlar la calidad a nivel del producto de software, para lograr este objetivo se calificará el software por medio de la aplicación llamada SonarQube el mismo que está configurado en base a la familia de normas ISO 25000 y por medio del cual nos permitirá ajustar el código fuente, asegurando así el mantenimiento, fiabilidad y seguridad que deben estar presentes en las aplicaciones desarrolladas en la Universidad Israel.

Palabras claves: ISO, Calidad, Software, Producto, SonarQube.

ABSTRACT

The Universidad Israel in developing software is not focused on quality control at the software product level in order to achieve this objective, the software will be qualified by means of the SonarQube application, which is configured according to the ISO 25000 family of standards and By means of which it will allow us to adjust the source code, thus ensuring the maintenance, reliability and security that must be present in the applications developed at the Universidad Israel.

Keywords: ISO, Quality, Software, Product, SonarQube.

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

SECCIÓN I

1. PROBLEMA DE INVESTIGACIÓN

En la Universidad Israel el desarrollo de software es de consumo interno y el mismo solo ha sido probado a nivel funcional lo que conocemos como "testing" pero no se lo ha evaluado por medio de una norma esta evaluación a nivel de producto de software a través de esta investigación se entregarán las pautas y herramientas de medición que nos permitirán mirar el producto de software desde su funcionalidad, portabilidad, usabilidad, seguridad, mantenibilidad, eficiencia.

En el ciclo de vida de un proyecto de software la calidad está inmersa en todas sus etapas esta calidad tiene por objetivo enfocarse en cada uno de los aspectos del desarrollo de software la calidad en los procesos nos ayudarán a obtener salidas y entradas uniformes, calidad en el producto de software y calidad en los recursos humanos los mismos que conforman el equipo encargado del producto final del desarrollo de software.

La calidad en los proyectos de desarrollo de software es un tema al cual cada vez se lo ha ido considerando más en las etapas de análisis, planificación, desarrollo y pruebas incluyendo más recursos que tendrán como objetivo final el entregar un producto de software que cumpla con todo lo especificado por el cliente y que sea funcionalmente efectivo para el usuario final. Para lograr esta calidad se han planteado y definido una cantidad importante de avances que nos ayudarán a controlar la calidad no solo en los entregables sino en el producto de software.

La calidad debe estar presente en todos los productos de software considerando que existen diferentes empresas dedicadas al consumo de software este puede ser de consumo interno, tratarse de factorías de software y empresas que solicitan software a

medidas según sus necesidades de negocio, esta calidad debe ser revisada y considerada tanto para empresas dedicadas a desarrollar como adquirir software

El desarrollo interno o externo debe funcionar como se solicitó y acordó que lo haría sin embargo debemos estar consciente que el software debe avanzar y adaptarse al negocio ya que el mismo cambia constantemente, este producto se espera que sea capaz de coexistir con otros desarrollos en diferentes plataformas y recibir el respectivo mantenimiento, para que el software pueda ser capaz de soportar todos los requerimientos no es suficiente centrarnos en estandarizar procesos o vigilar que los entregables cumplan estándares a nivel únicamente documental menos aun pensar que las pruebas funcionales son suficientes para pasar un desarrollo a un ambiente de producción.

- Calidad del producto no es lo mismo que "testing" o pruebas.

Que el software "funcione" es lo que normalmente intentas saber con las pruebas funcionales, pero esto no significa que por ello el producto este bien hecho. Si el producto no está hecho no podemos determinarlo hasta que se revisen sus fuentes, diseño, la lógica implementada, es decir, si no se considera lo que se llama calidad del producto software no podemos determinar que el software esté bien.

- La calidad del proceso no determina la calidad del software.

Kitchenham y Pfleeger en un artículo en IEEE software (Kitchenham & Pfleeger, 1996) "comentaban que la principal crítica a esta visión es que hay poca evidencia en que cumplir un modelo de procesos asegure la calidad del producto, la estandarización de los procesos garantiza la uniformidad en la salida de los mismos, lo que puede incluso institucionalizar la creación de malos productos".

Maibaum y Wassyng, en Computer (Maibaum & Wassyng, 2008), comentaban, "siguiendo la misma línea, que las evaluaciones de calidad deberían estar basadas en evidencias extraídas directamente de los atributos del producto, y no en evidencias circunstanciales deducidas desde el proceso. Un proceso estándar, o institucionalizado, según sea la terminología del modelo de uso, no necesariamente concluye con un producto de calidad."

2. OBJETIVO GENERAL

Controlar la calidad del producto de software en el desarrollo interno de la Universidad Israel basando en la norma ISO/IEC 25010 evaluación de la calidad del producto e ISO/IEC 25040 con el fin de mejorando sus prestaciones a nivel de aplicación y rendimiento.

3. OBJETIVOS ESPECÍFICOS

- Establecer métricas de calidad interna para el producto de desarrollo de software en la Universidad Israel.
- Establecer métricas para la calidad externa del producto de software en la Universidad Israel.
- Establecer métricas para la calidad de uso del producto de software en la Universidad Israel.

4. INTRODUCCIÓN

En la Universidad Israel se desarrolló una investigación la cual presentó una serie de herramientas capaces de asegurar la calidad en los proyectos de desarrollo de software, con ayuda de estas herramientas permite tener una integración continua, definir políticas y procedimientos que permitan automatizar procesos.

(Castrillon & Cobos, 2016)

La demanda de software ha aumentado debido a que está presente en todos los dispositivos que manejamos, en los sistemas de gestión, en el transporte, comunicaciones, energía, banca, ocio y entretenimiento. Este aumento de software ha permitido el crecimiento de empresas encargadas de su desarrollo las mismas que se conocen como "factorías de software" a su vez la falta de personal especializado para

ciertas tareas de desarrollo de software y la búsqueda en la reducción de costes da lugar a lo que se conoce como "outsourcing" del desarrollo de software. Sin embargo, cuando se externalizan actividades de desarrollo de software, también aumentan los riesgos y la falta de control sobre la calidad del software que la empresa contratada entrega, de esta manera surge la necesidad de evaluar y asegurar la calidad de software.

La evaluación de software debe basarse en evidencias directas del propio producto y no solo en evidencias del proceso de desarrollo.

Las empresas de desarrollo de software buscan la calidad del producto de software, así como la metodología seguir para poder realizar la evaluación de sus desarrollos

Desde un inicio la evaluación de la calidad de software se ha centrado en controlar la calidad de los procesos que se utilizan para su desarrollo, surgiendo así los modelos y estándares como CMMI o ISO/IEC 15504. La evidencia encontrada sobre cumplir los procesos y el estandarizar las salidas de los mismos podría institucionalizar la creación de malos productos es por eso que el interés es cada vez más alto sobre la calidad de los proyectos no solo en el proceso sino en el producto de desarrollo de software.

5. HIPÓTESIS

Utilizando las normas ISO/IEC 25010 e ISO/IEC 25040 diseñadas para controlar la calidad del producto en el desarrollo de software. La Universidad Israel al implementar estas normas evitará defectos una vez que el desarrollo pase a producción.

SECCIÓN II

1. MARCO TEÓRICO

1.1 NORMA ISO/IEC 9126

Modelo de calidad del producto de software (1991-2004)

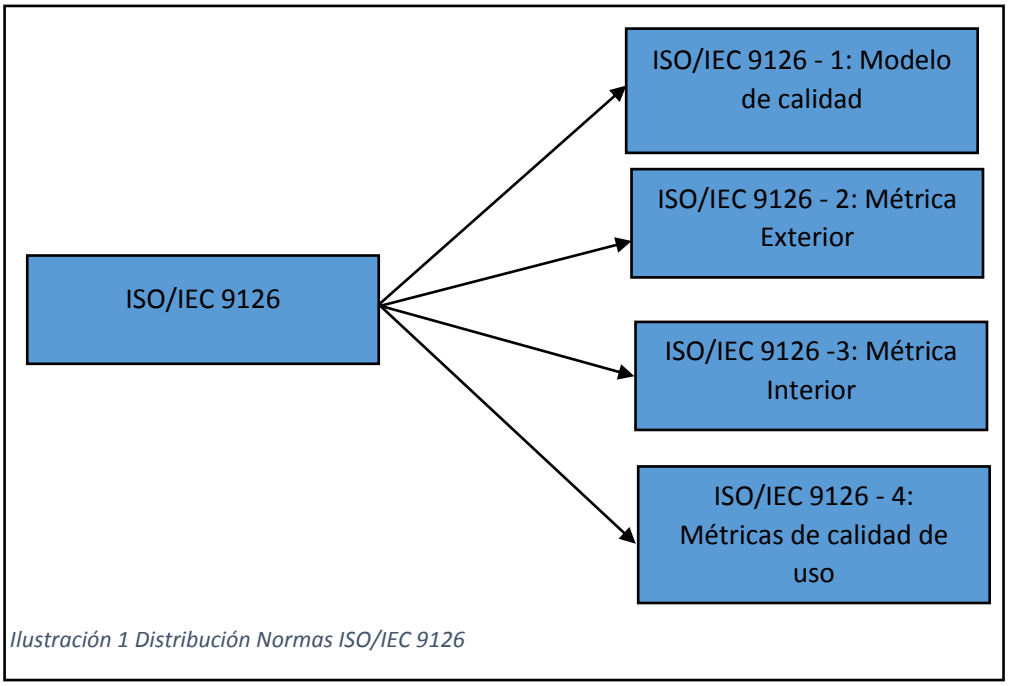


Ilustración 1 Distribución Normas ISO/IEC 9126

(Muñoz Sagavinzaga, Natavidad Alejos, Quiroz Villalobos, & Villegas Vilcherres, 2009)

Métricas de calidad interna, externa y de uso según ISO/IEC 9126.

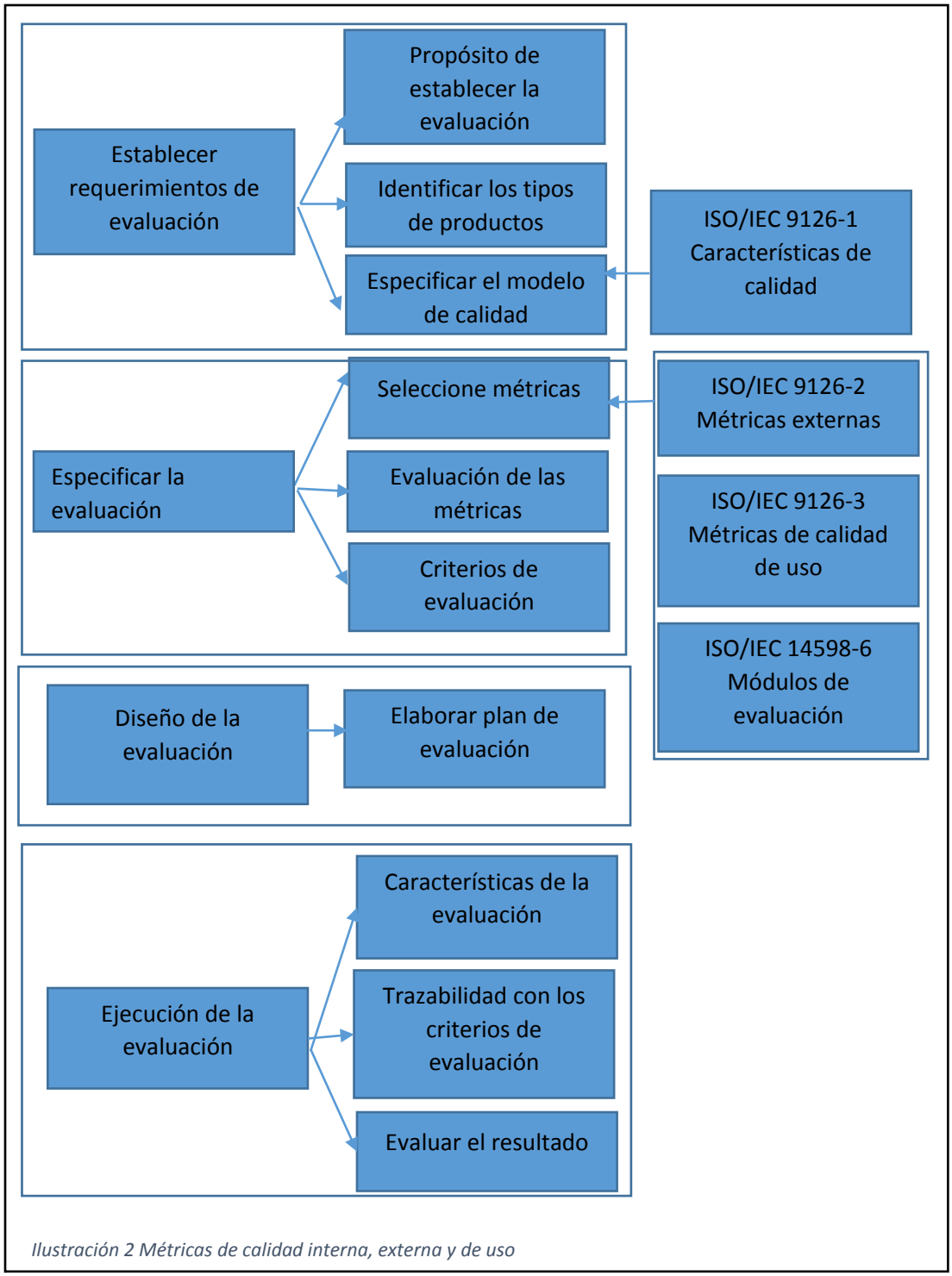


Ilustración 2 Métricas de calidad interna, externa y de uso

(Muñoz Sagavinzaga, Natavidad Alejos, Quiroz Villalobos, & Villegas Vilcherres, 2009)

1.1.1 MÉTRICAS DE FUNCIONALIDAD

- Adecuado
- Exacto
- Interoperabilidad
- Seguridad
- Conformidad de la funcionalidad

Nombre:	**Completitud de implementación funcional**
Propósito:	Qué tan completa está la implementación funcional.
Método de aplicación:	Contar las funciones faltantes detectadas en la evaluación y comparar con el número de funciones descritas en la especificación de requisitos.
Medición, fórmula:	X = 1 - A/B A = número de funciones faltantes B = número de funciones descritas en la especificación de requisitos
Interpretación:	0 <= X <= 1 Entre más cercano a 1, más completa.
Tipo de escala:	absoluta
Tipo de medida:	X = count/count A = count B = count
Fuente de medición:	Especificación de requisitos Diseño Código fuente Informe de revisión
Audiencia:	Solicitantes Desarrolladores

Tabla 1 Evaluación métricas de funcionalidad

(Mena, 2006)

1.1.2 MÉTRICAS DE FIABILIDAD

- Madurez
- Tolerancia a fallos
- Recuperable
- Conformidad de la fiabilidad

Nombre:	**Suficiencia de las pruebas**
Propósito:	Cuántas de los casos de prueba necesarios están cubiertos por el plan de pruebas.
Método de aplicación:	Contar las pruebas planeadas y comparar con el número de pruebas requeridas para obtener una cobertura adecuada.
Medición, fórmula:	X = A/B A = número de casos de prueba en el plan B = número de casos de prueba requeridos
Interpretación:	0 <= X Entre X se mayor, mejor la suficiencia.
Tipo de escala:	absoluta
Tipo de medida:	X = count/count A = count B = count
Fuente de medición:	A proviene del plan de pruebas B proviene de la especificación de requisitos
Audiencia:	Desarrolladores Mantenedores

Tabla 2 Evaluación métricas de fiabilidad

(Mena, 2006)

1.1.3 MÉTRICAS DE USABILIDAD

- Entendible
- Aprender a usar
- Operable
- Atractivo
- Conformidad de la usabilidad

Nombre:	**Funciones evidentes**
Propósito:	Qué proporción de las funciones de los sistemas son evidentes al usuario.
Método de aplicación:	Contar las funciones evidentes al usuario y comparar con el número total de funciones.
Medición, fórmula:	X = A/B A = número de funciones (o tipos de funciones) evidentes al usuario B = total de funciones (o tipos de funciones)
Interpretación:	0 <= X <= 1 Entre más cercano a 1, mejor.
Tipo de escala:	absoluta
Tipo de medida:	X = count/count A = count B = count
Fuente de medición:	Especificación de requisitos Diseño Informe de revisión
Audiencia:	Solicitante Desarrolladores

Tabla 3 Evaluación métricas de usabilidad

(Mena, 2006)

1.1.4 MÉTRICAS DE EFICIENCIA

- Comportamiento en el tiempo
- Utilización de recursos
- Conformidad de la eficiencia

Nombre:	**Tiempo de respuesta**
Propósito:	Cuál es el tiempo estimado para completar una tarea.
Método de aplicación:	Evaluar la eficiencia de las llamadas al SO y a la aplicación. Estimar el tiempo de respuesta basado en ello. Puede medirse: • Todo o partes de las especificaciones de diseño. • Probar la ruta completa de una transacción. • Probar módulos o partes completas del producto. • Producto completo durante la fase de pruebas.
Medición, fórmula:	X = tiempo (calculado o simulado)
Interpretación:	Entre más corto, mejor.
Tipo de escala:	proporción
Tipo de medida:	X = time
Fuente de medición:	Sistema operativo conocido Tiempo estimado en llamadas al sistema
Audiencia:	Desarrolladores Solicitante

Tabla 4 Evaluación métricas de eficiencia

(Mena, 2006)

1.1.5 MÉTRICAS DE MANTENIBILIDAD

- Analizable
- Cambiable
- Estabilidad
- Examinar
- Conformidad de la mantenibilidad

Nombre:	**Registro de cambios**
Propósito:	¿Se registran adecuadamente los cambios a la especificación y a los módulos con comentarios en el código?
Método de aplicación:	Registrar la proporción de información sobre cambios a los módulos
Medición, fórmula:	X = A/B A = número de cambios a funciones o módulos que tienen comentarios confirmados B = total de funciones o módulos modificados
Interpretación:	0 <= X <= 1 Entre más cercano a 1, más registrable. 0 indica un control de cambios deficiente o pocos cambios y alta estabilidad.
Tipo de escala:	absoluta
Tipo de medida:	X = count/count A = count B = count
Fuente de medición:	Sistema de control de configuraciones Bitácora de versiones Especificaciones
Audiencia:	Desarrolladores Mantenedores Solicitante

Tabla 5 Evaluación métricas de mantenibilidad

(Mena, 2006)

1.1.6 MÉTRICAS DE PORTABILIDAD

- Adaptabilidad
- Instalación
- Coexistencia
- Remplazar
- Conformidad de la portabilidad

Nombre:	**Conformidad de portabilidad**
Propósito:	Qué tan conforme es la portabilidad del producto con regulaciones, estándares y convenciones aplicables.
Método de aplicación:	Contar los artículos encontrados que requieren conformidad y comparar con el número de artículos en la especificación que requieren conformidad.
Medición, fórmula:	X = A/B A = número de artículos implementados de conformidad B = total de artículos que requieren conformidad
Interpretación:	0 <= X <= 1 Entre más cercano a 1, más completa.
Tipo de escala:	absoluta
Tipo de medida:	X = count/count A = count B = count
Fuente de medición:	Especificación de conformidad y estándares, convenciones y regulaciones relacionados. Diseño Código fuente Informe de revisión
Audiencia:	Solicitante Desarrolladores

Tabla 6 Evaluación métricas de portabilidad

(Mena, 2006)

1.2 NORMA ISO/IEC 14598

Calidad de la evaluación del producto de software, las normas ISO/IEC 14598 están distribuidas de la siguiente forma:

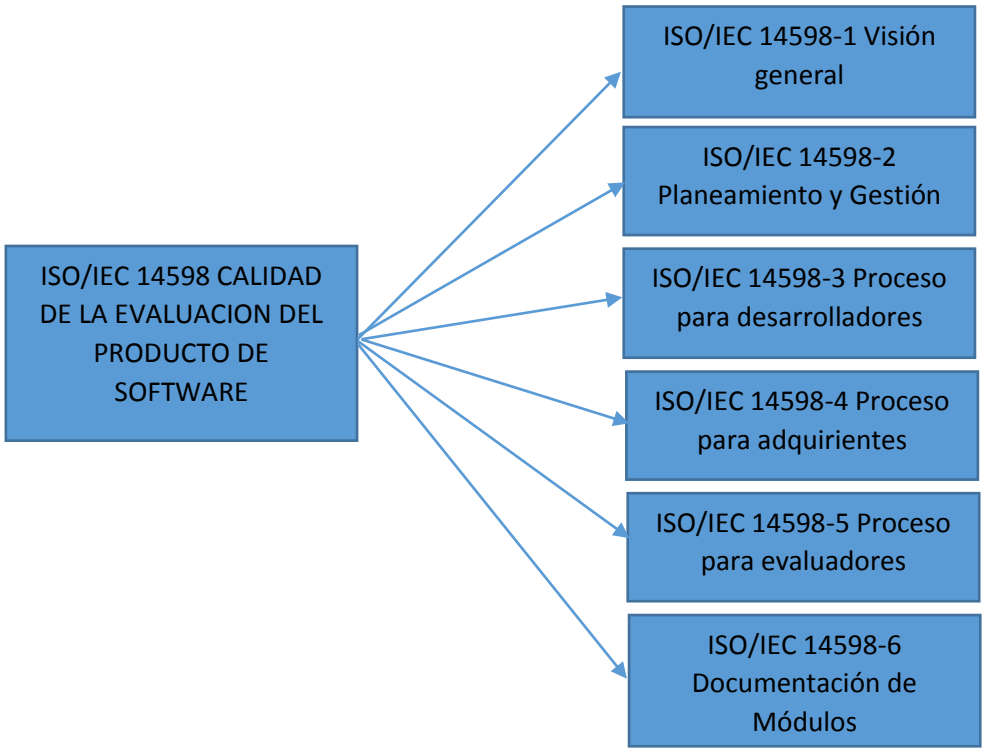


Ilustración 3 Distribución Normas ISO/IEC 14598

(Muñoz Sagavinzaga, Natavidad Alejos, Quiroz Villalobos, & Villegas Vilcherres, 2009)

Estas normas se enfocan en proveer un método para medir la calidad hacia los interesados o involucrados en el proceso de desarrollo de software como son los proveedores, compradores, evaluadores y usuarios.

Que no es más que un resumen de todo el ciclo de calidad, entrega pautas entre la evaluación del producto de software y el modelo de calidad en la visión general y se tendrá que considerar las siguientes etapas:

- Establecer los requisitos de calidad.
- Especificar la evaluación.
- Planear la evaluación.
- Ejecutar la evaluación.

1.2.1 Planeamiento y control: recomendaciones y guías que sirven como apoyo en el desarrollo de software.

1.2.2 Procesos para desarrolladores: selección y registro de indicadores que pueden ser medidos y evaluados a partir de resultados intermedios obtenidos durante las fases de desarrollo para que en base a esto se tomen decisiones acerca del proyecto.

1.2.3 Proceso para los compradores: establece un proceso sistemático para la evaluación de productos de software comercial, personalizar o modificar software existente. Usados para garantizar que un producto desarrollado o modificado cumple con los requisitos inicialmente especificados.

1.2.4 Proceso para evaluadores: Orientaciones y recomendaciones para la aplicación práctica de la evaluación de producto de software cuando las diversas partes, necesitan comprender, aceptar y confiar en los resultados de la evaluación.

1.2.5 Documentación de módulos: Describe la introducción, el alcance de la evaluación, entradas para cada uno de los procesos y entrega resultados.

(Muñoz Sagavinzaga, Natavidad Alejos, Quiroz Villalobos, & Villegas Vilcherres, 2009)

1.2.6 TIPOS DE PRODUCTOS A SER EVALUADOS SEGÚN LA ISO/IEC 14598

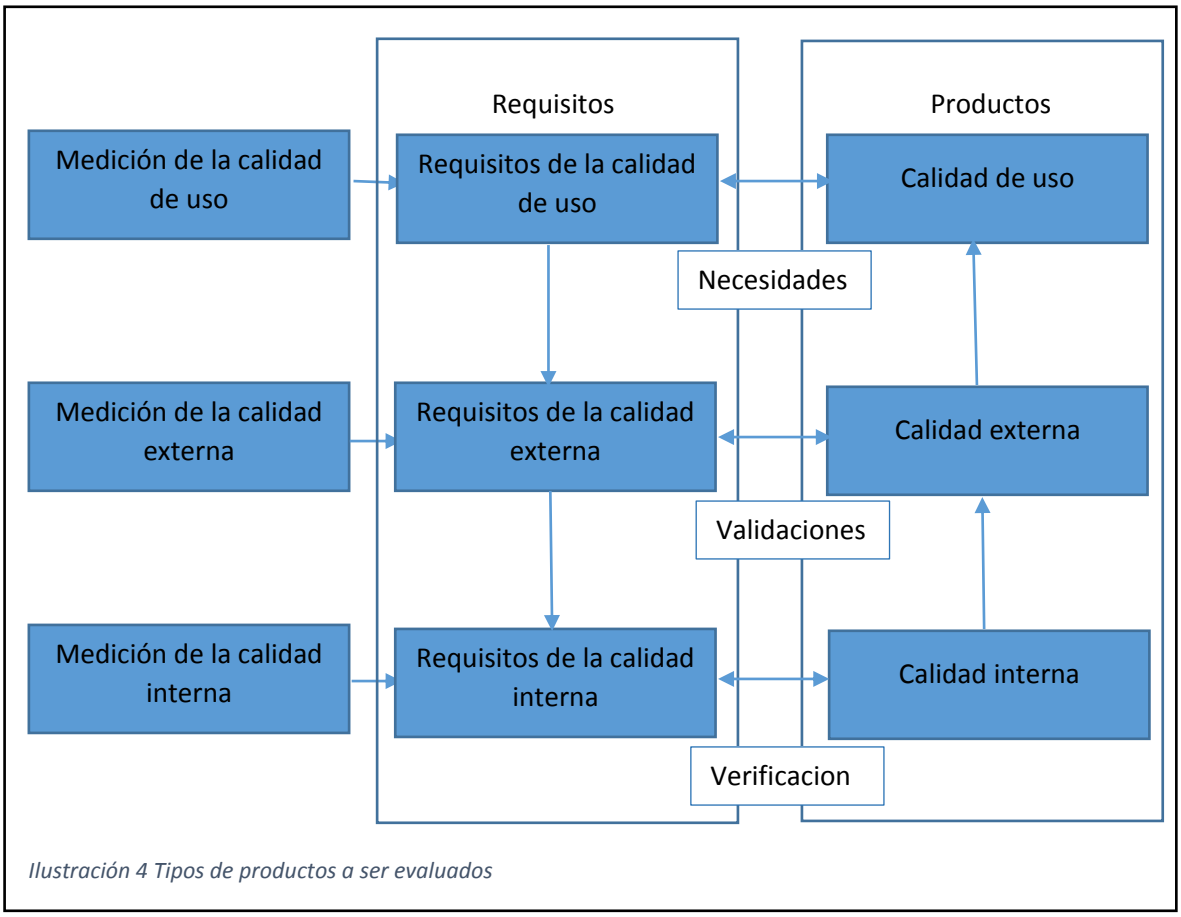


Ilustración 4 Tipos de productos a ser evaluados

(Muñoz Sagavinzaga, Natavidad Alejos, Quiroz Villalobos, & Villegas Vilcherres, 2009)

1.2.7 DISTRIBUCIÓN ENTRE LOS ESTÁNDARES DE CALIDAD ISO/IEC 9126 E ISO/IEC 145998

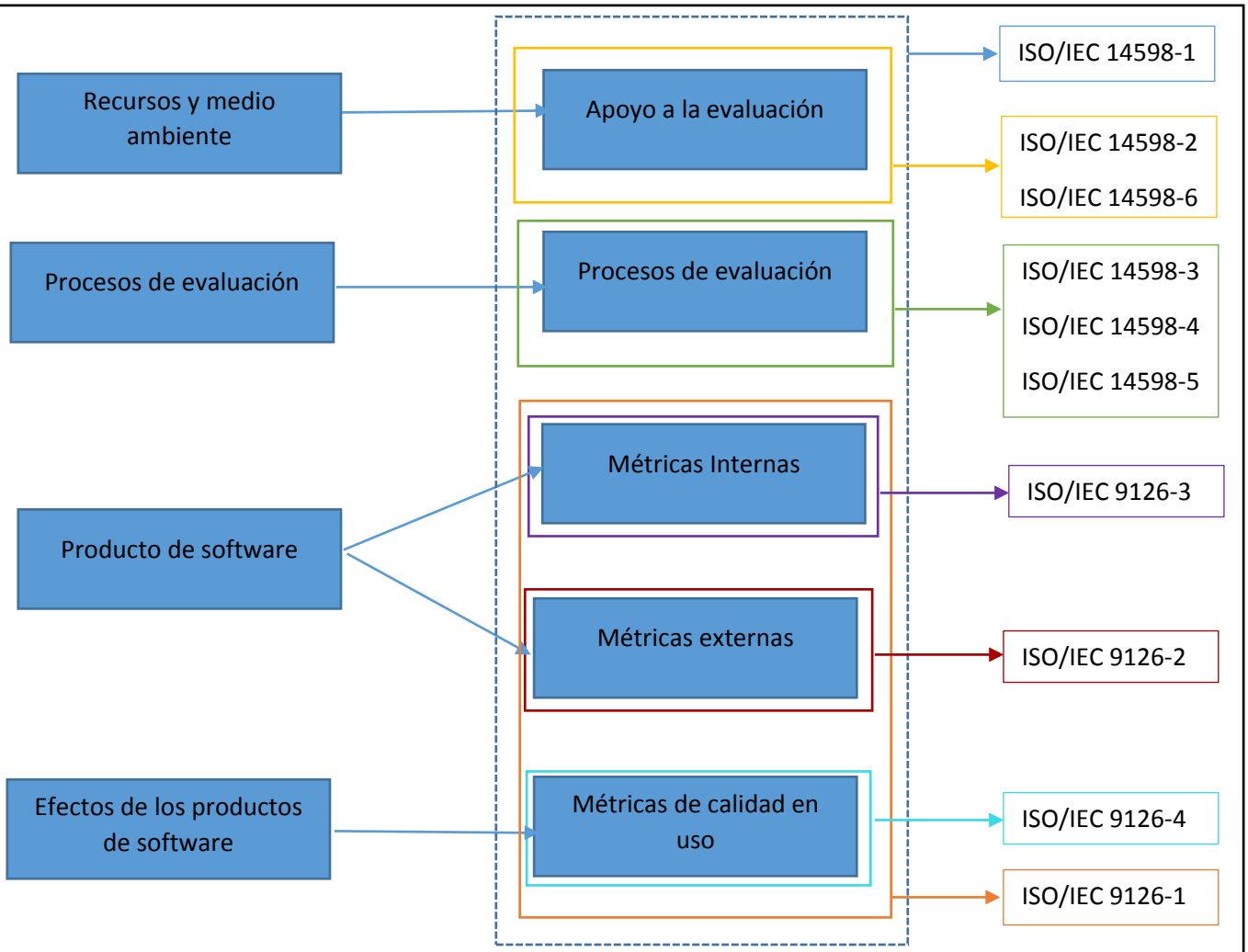


Ilustración 5 Distribución entre los estándares de calidad ISO/IEC 9126 e ISO/IEC 145998

(Muñoz Sagavinzaga, Natavidad Alejos, Quiroz Villalobos, & Villegas Vilcherres, 2009)

(ISO, 2015)

1.3 NORMAS ISO/IEC 25000

1.3.1 ISO/IEC 2500n DIVISIÓN DE GESTIÓN DE CALIDAD

La división de gestión de calidad define las diferentes normas que compone la ISO 25000

- ISO/IEC 25000 Guía SQuaRE: contiene usuarios planificados, partes que componen la guía y los modelos de referencia
- ISO/IEC 25001 Planificación y Gestión: establece los requisitos para gestionar, establecer los requerimientos de calidad y ejecutar la evaluación de software según lo planificado

(iso25000, 2017)

1.3.2 ISO/IEC 2501n DIVISIÓN DE MODELO DE CALIDAD

Describe características propias correspondientes a calidad interna y externa para el producto de software.

- ISO/IEC 25010 Modelo de calidad de sistemas y software: Modelo de calidad para el uso del producto de software, nos muestra contra que comprar el resultado del producto de software

La evaluación de calidad del producto de software determina las características que se consideraran al momento de evaluar las propiedades de un producto de software determinado.

(iso25000, 2017)

La calidad del producto se la puede interpretar como el grado en que dicho producto satisface los requisitos de los usuarios aportando de esa manera un valor

los requisitos tanto funcionales como seguridad, rendimiento y mantenibilidad los mismo que están representados en el modelo de calidad del producto

(iso25000, 2017)

a. Adecuación funcional: Representa la capacidad del producto de software para proporcionar funciones que satisfacen las necesidades declaradas e implícitas, cuando el producto se usa en las condiciones especificadas, estas características se dividen de la siguiente forma:

- Funcionalmente completo: El grado en que el producto de software cumple todas las tareas y objetivos del usuario.
- Funcionalmente correctos: El resultado del producto de software debe tener el nivel de precisión requerido.
- Pertinencia funcional: Proporcionar un conjunto apropiado de funciones para tareas y objetivos específicos del usuario.

b. Eficiencia de desempeño: Representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones.

- ISO/IEC 25012 Modelo de calidad de datos: Modelo para el almacenamiento estructurado de datos que forman parte del producto de software.

(iso25000, 2017)

1.3.3 ISO/IEC 2502n DIVISIÓN DE MEDICIÓN DE CALIDAD

Aplicación práctica de los modelos de calidad interna, externa y de uso

- ISO/IEC 25020 Modelo de referencia medición y guía: Introduce el modelo de referencia común a los elementos de medición de calidad, también proporciona una guía para escoger las medidas propuestas por la ISO.
- ISO/IEC 25021 Elementos para medir la calidad: Recomienda métricas en que basarse a lo largo del ciclo de vida del producto de software.
- ISO/IEC 25022 Medidas de calidad en uso: Especifica el cómo medir la calidad del uso del producto.
- ISO/IEC 25023 Medición de la calidad de los productos de sistema y software: define métricas de medición para la calidad de los productos de software.
- ISO/IEC 25024 Medición de la calidad en la información: Determina la calidad que deben tener los datos.

(iso25000, 2017)

1.3.4 ISO/IEC 2503n DIVISIÓN DE REQUISITOS DE CALIDAD

Las normas que forman este apartado ayudan a especificar requisitos de calidad que pueden ser utilizados en procesos de requisitos de calidad del producto de software a desarrollar como entradas del proceso de evaluación.

- ISO/IEC 25030 Requisitos de calidad: Provee un conjunto de recomendaciones para realizar las especificaciones de los requisitos de calidad del producto de software

(iso25000, 2017)

1.3.5 ISO/IEC 2504n DIVISIÓN DE EVALUACIÓN DE CALIDAD

Incluyen normas que proporcionan requisitos, recomendaciones, guías para llevar a cabo el proceso de evaluación del producto de software.

- ISO/IEC 25040 Modelo de Referencia de Evaluación y Guía: Propone un modelo de referencia general para la evaluación que considera las entradas al proceso de evaluación, las restricciones y los recursos necesarios para obtener las correspondientes salidas
- ISO/IEC 25041 Guía de Evaluación para Desarrolladores, Adquirientes y evaluadores independientes: Describe los requisitos y recomendaciones para la implementación práctica de la evaluación del producto de software desde el punto de vista de los desarrolladores, de los adquirientes y los evaluadores independientes
- ISO/IEC 25042 Módulos de Evaluación: define lo que la norma considera un módulo de evaluación, documentación estructurada que se debe utilizar a la hora de definir uno de estos módulos.
- ISO/IEC 25045 Módulo de la Evaluación para la Recuperación: Define un módulo para la evaluación de la sub característica y su capacidad de recuperarse.

(iso25000, 2017)

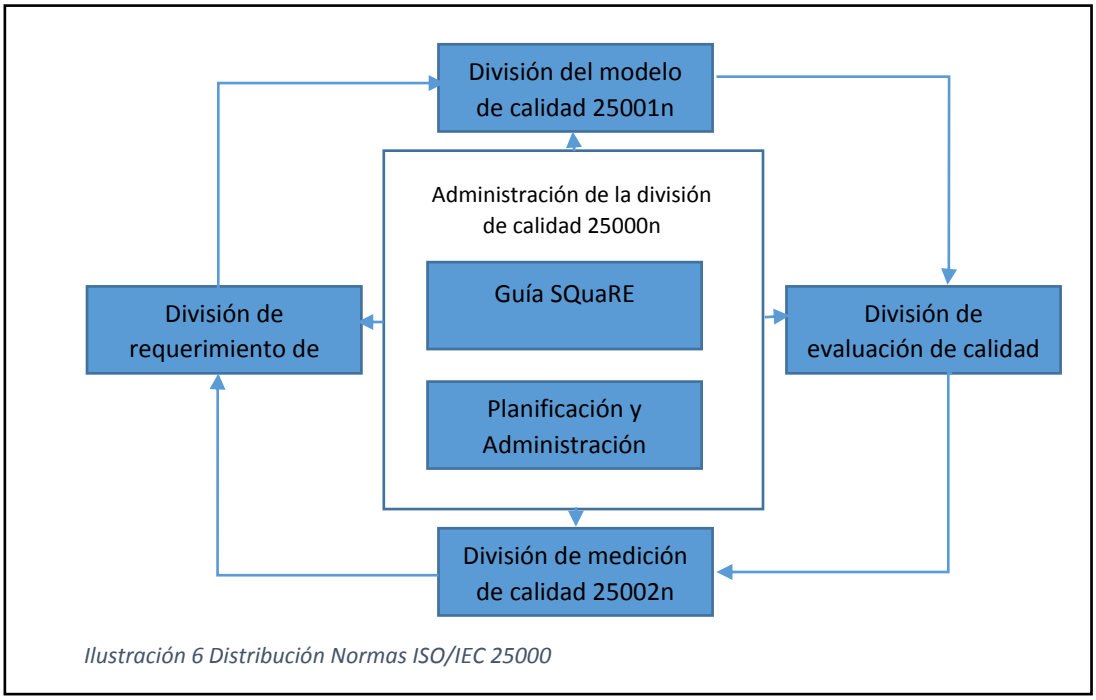


Ilustración 6 Distribución Normas ISO/IEC 25000

1.4 ISO/IEC 25010

Determina las características de la calidad de software que se puede evaluar, el modelo de calidad representa la piedra angular en torno a la cual se establece el sistema para la evaluación de la calidad del producto. En este modelo se determinan las características de calidad que se van a tener en cuenta a la hora de evaluar las propiedades de un producto software determinado. 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. Son precisamente estos requisitos (funcionalidad, rendimiento, seguridad, mantenibilidad) los que se encuentran representados en el modelo de calidad, el cual categoriza la calidad del producto en características y sub características.

El modelo de calidad del producto definido por la ISO/IEC 25010 se encuentra compuesto por las ocho características de calidad que se muestran en la siguiente figura:

(Muñoz Sagavinzaga, Natavidad Alejos, Quiroz Villalobos, & Villegas Vilcherres, 2009)

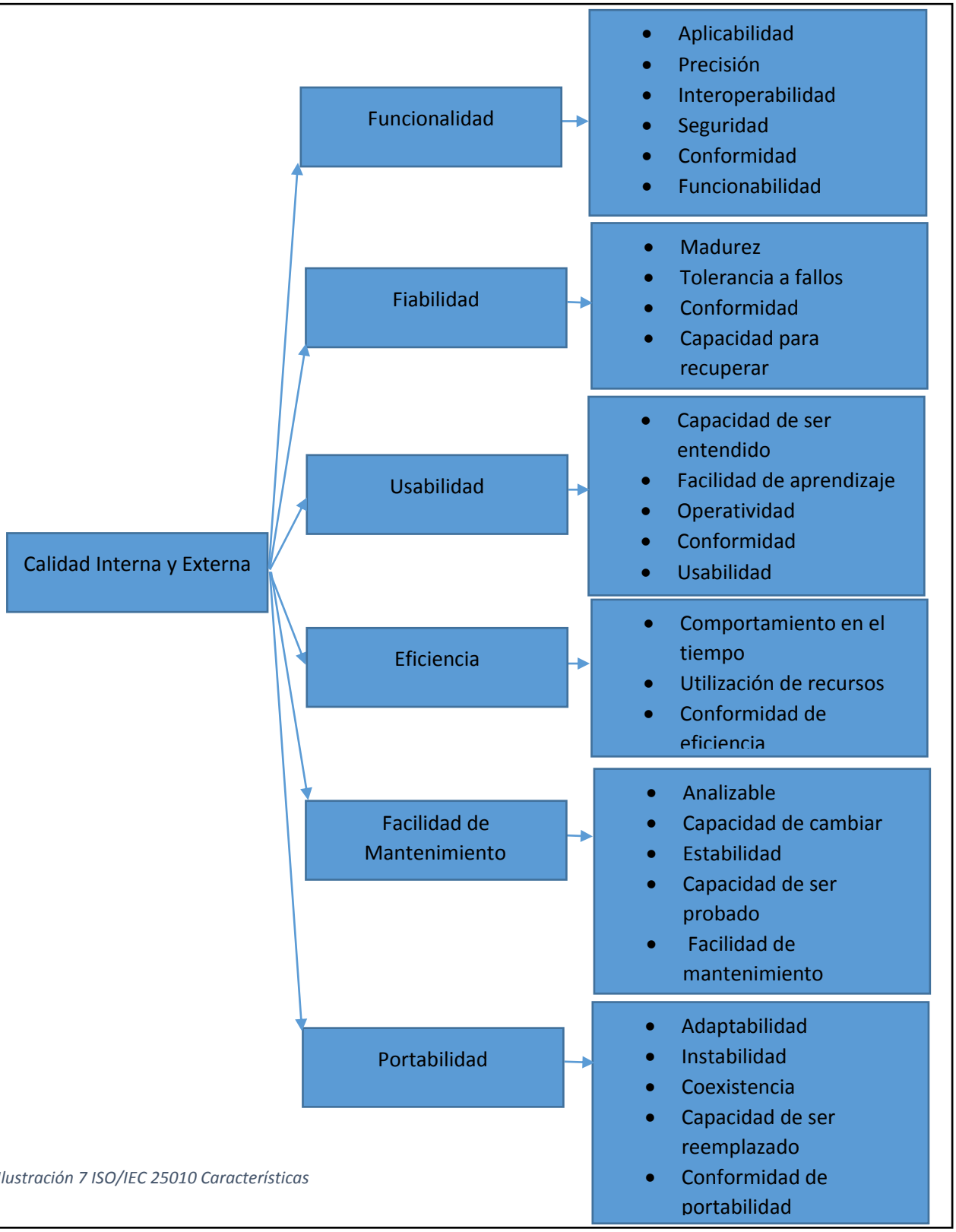


Ilustración 7 ISO/IEC 25010 Características

1.5 ISO/IEC 25040

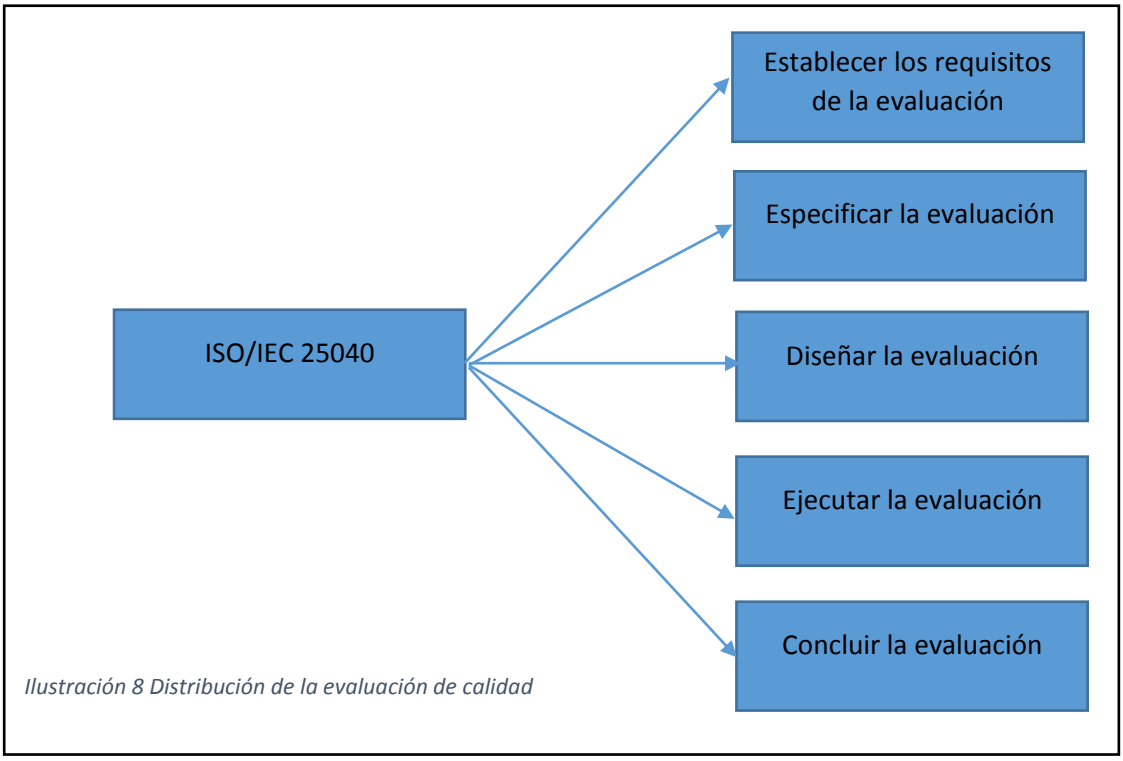


Ilustración 8 Distribución de la evaluación de calidad

1.5.1 ESTABLECER LOS REQUISITOS DE LA EVALUACIÓN

El primer paso del proceso de evaluación consiste en establecer los requisitos de la evaluación.

a. Establecer el propósito de la evaluación

En esta tarea se documenta el propósito por el que la organización quiere evaluar la calidad de su producto software (asegurar la calidad del producto, decidir si se acepta un producto, determinar la viabilidad del proyecto en desarrollo, comparar la calidad del producto con productos de la competencia).

b. Obtener los requisitos de calidad del producto

En esta tarea se identifican las partes interesadas en el producto software (desarrolladores, posibles adquirientes, usuarios, proveedores) y se especifican los requisitos de calidad del producto utilizando un determinado modelo de calidad.

c. Identificar las partes del producto que se deben evaluar

Se deben identificar y documentar las partes del producto software incluidas en la evaluación. El tipo de producto a evaluar (especificación de requisitos, diagramas de diseño, documentación de las pruebas) depende de la fase en el ciclo de vida en que se realiza la evaluación y del propósito de ésta.

d. Definir el rigor de la evaluación

Se debe definir el rigor de la evaluación en función del propósito y el uso previsto del producto software, basándose, por ejemplo, en aspectos como el riesgo para la seguridad, el riesgo económico o el riesgo ambiental. En función del rigor se podrá establecer qué técnicas se aplican y qué resultados se esperan de la evaluación.

(iso25000, 2017)

1.5.2 ESPECIFICAR LA EVALUACIÓN

En esta actividad se especifican los módulos de evaluación (compuestos por las métricas, herramientas y técnicas de medición) y los criterios de decisión que se aplicarán en la evaluación.

a. Seleccionar los módulos de evaluación

En esta tarea el evaluador selecciona las métricas de calidad, técnicas y herramientas (módulos de evaluación) que cubran todos los requisitos de la evaluación. Dichas métricas deben permitir que, en función de su valor, se puedan realizar comparaciones fiables con criterios que permitan tomar decisiones. Para ello se puede tener en cuenta la Norma ISO/IEC 25020.

b. Definir los criterios de decisión para las métricas

Se deben definir los criterios de decisión para las métricas seleccionadas. Dichos criterios son umbrales numéricos que se pueden relacionar con los requisitos de calidad y posteriormente con los criterios de evaluación para decidir la calidad del producto. Estos umbrales se pueden establecer a partir de benchmarks, límites de control estadísticos, datos históricos, requisitos del cliente.

c. Definir los criterios de decisión de la evaluación

Se deben definir criterios para las diferentes características evaluadas a partir de las sub características y métricas de calidad. Estos resultados a mayor nivel de abstracción permiten realizar la valoración de la calidad del producto software de forma general.

(iso25000, 2017)

1.5.3 DISEÑAR LA EVALUACIÓN

En esta actividad se define el plan con las actividades de evaluación que se deben realizar.

a. Planificar las actividades de la evaluación

Se deben planificar las actividades de la evaluación teniendo en cuenta la disponibilidad de los recursos, tanto humanos como materiales, que puedan ser necesarios. En la planificación se debe tener en cuenta el presupuesto, los métodos de evaluación y estándares adaptados, las herramientas de evaluación.

El plan de evaluación se revisará y actualizará proporcionando información adicional según sea necesario durante el proceso de evaluación.

(iso25000, 2017)

1.5.4 EJECUTAR LA EVALUACIÓN

En esta actividad se ejecutan las actividades de evaluación obteniendo las métricas de calidad y aplicando los criterios de evaluación.

a. Realizar las mediciones

Se deben realizar las mediciones sobre el producto software y sus componentes para obtener los valores de las métricas seleccionadas e indicadas en el plan de evaluación. Todos los resultados obtenidos deberán ser debidamente registrados.

b. Aplicar los criterios de decisión para las métricas

Se aplican los criterios de decisión para las métricas seleccionadas sobre los valores obtenidos en la medición del producto.

c. Aplicar los criterios de decisión de la evaluación

En esta última tarea se deben aplicar los criterios de decisión a nivel de características y sub características de calidad, produciendo como resultado la valoración del grado en que el producto software cumple los requisitos de calidad establecidos.

(iso25000, 2017)

1.5.5 CONCLUIR LA EVALUACIÓN

En esta actividad se concluye la evaluación de la calidad del producto software, realizando el informe de resultados que se entregará al cliente y revisando con éste los resultados obtenidos.

a. Revisar los resultados de la evaluación

Mediante esta tarea, el evaluador y el cliente de la evaluación (en caso de existir) realizan una revisión conjunta de los resultados obtenidos, con el objetivo de

realizar una mejor interpretación de la evaluación y una mejor detección de errores.

b. Crear el informe de evaluación

Una vez revisados los resultados, se elabora el informe de evaluación, con los requisitos de la evaluación, los resultados, las limitaciones y restricciones, el personal evaluador.

c. Revisar la calidad de la evaluación y obtener feedback

El evaluador revisará los resultados de la evaluación y la validez del proceso de evaluación, de los indicadores y de las métricas aplicadas. El feedback de la revisión debe servir para mejorar el proceso de evaluación de la organización y las técnicas de evaluación utilizadas.

d. Tratar los datos de la evaluación

Una vez finalizada la evaluación, el evaluador debe realizar el adecuado tratamiento con los datos y los objetos de la evaluación según lo acordado con el cliente (en caso de ser una tercera parte), devolviéndolos, archivándolos o eliminándolos según corresponda.

(ISO, 2015)

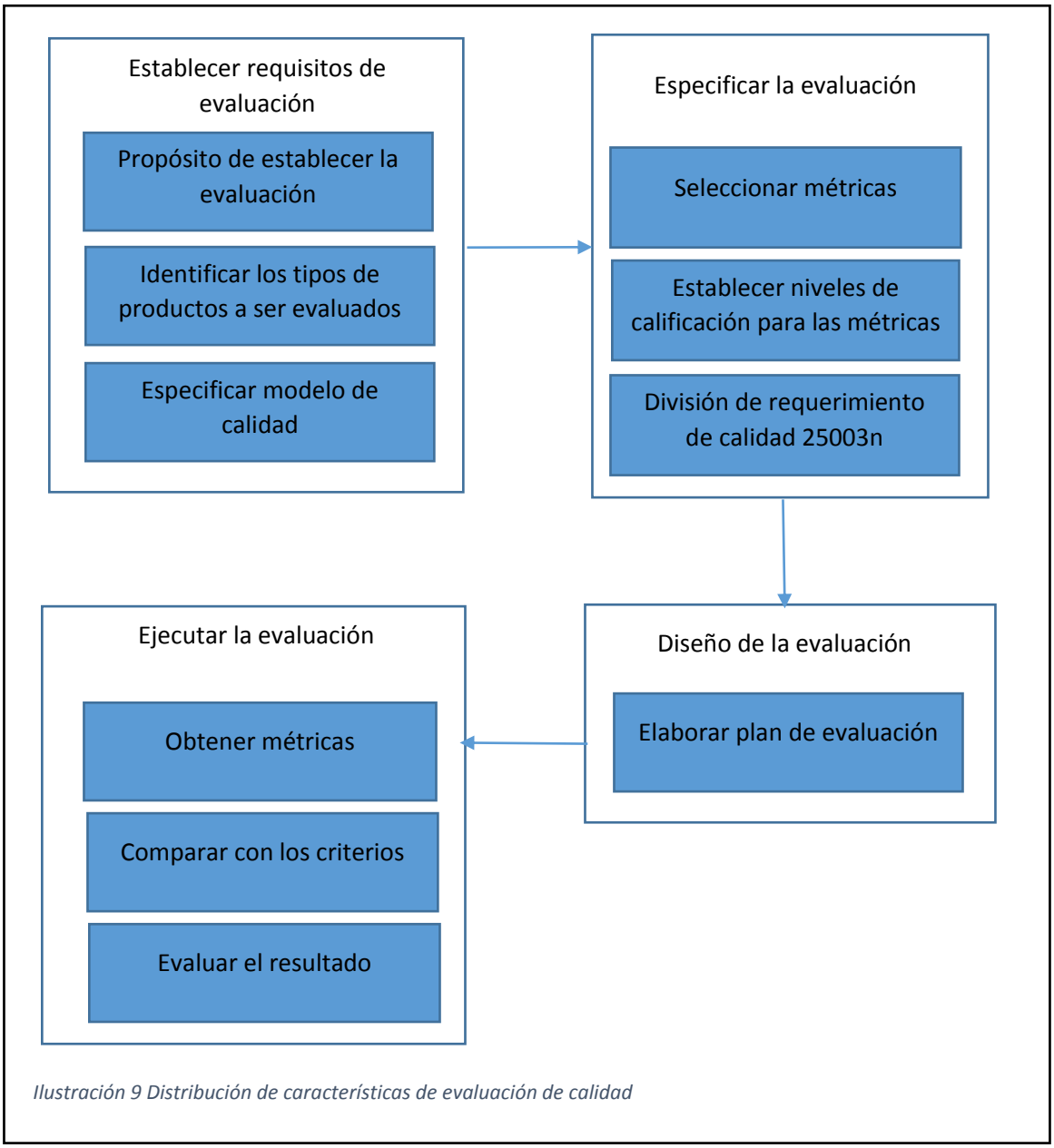


Ilustración 9 Distribución de características de evaluación de calidad

(ISO, 2015)

SECCIÓN III

1. METODOLOGÍA

La calidad de software está relacionada con el grado de satisfacción que tiene el cliente, existen diferentes tipos de clientes, sin embargo, todos buscan obtener un producto de software que sea actualizable, que permita cambiar de ambientes de trabajo y que no falle al momento de utilizarlo.

Para poder satisfacer esta necesidad la calidad del producto de software juega un papel fundamental ya que el satisfacer todas las expectativas sobre el software desarrollado es necesario implementar métricas, inspeccionar el código fuente, realizar pruebas y respetar los procesos.

Dichos procesos están presentes en toda la vida del desarrollo de software en esta investigación está enfocada en presentar pautas para el control de calidad en el producto de software. (iso25000, 2017)

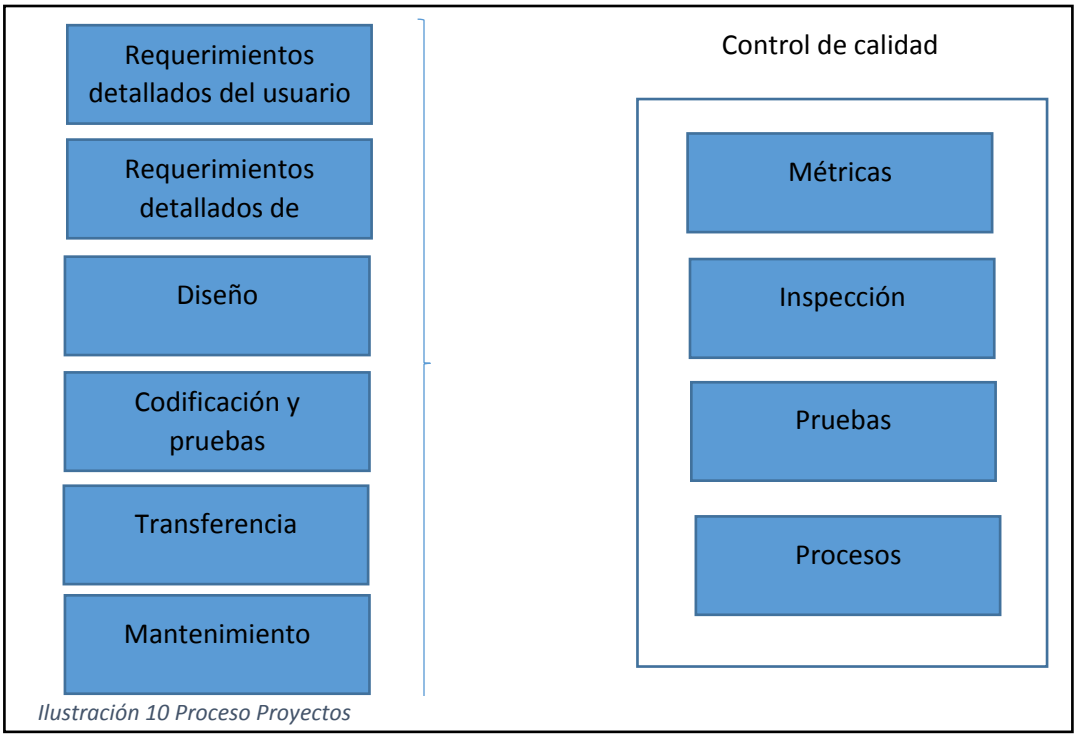


Ilustración 10 Proceso Proyectos

SQuaRE relaciona las necesidades con requerimientos de calidad y requerimientos funcionales que son:

- Identificar y obtener las necesidades de los interesados dichas necesidades deben ser clasificadas y organizadas.
- Al reconocer las necesidades de los interesados las mismas pueden ser enunciadas, no enunciadas y desconocidas.
- Seleccionar y especificar los requerimientos de calidad en uso, enunciados y seleccionados sin dejar las restricciones.
- Los requerimientos funcionales que son los dependientes y del negocio todo esto se plasma en el diseño visual.

(Websec, 2017)

El no controlar la calidad o pasar intencionalmente por alto dicho tema conlleva la definición de una deuda técnica, esta deuda no es nada más que defectos que el software puede causar a corto o largo plazo estos problemas que el software pueda causar a la operación de quién lo utilice se lo denomina intereses, es por ello que se define como deuda técnica a todo lo que se puede evitar pero no lo hacemos, existen empresas desarrolladoras de software que envían productos sin los estándares necesarios esto causa que los mantenimientos se extiendan indefinidamente.

(ISO, 2015)

La deuda técnica empieza al momento en que el desarrollo de software empieza a desarrollarse esta puede ser por utilizar mano de obra de baja calidad, desarrolladores inexpertos, fechas de entregas imposibles, actualización constante del alcance del proyecto no tener al equipo de trabajo motivado ni trabajando en un ambiente que le permita desarrollar sin problemas el software al no considerar esto un desarrollo se vuelve pasivo y empieza a consumirse a sí mismo ya sea por no haber planificado o no

haber detectado a tiempo los problemas de calidad a los cuales se estaba encaminando el desarrollo. La calidad de software tiene mucho que ver ya que un código limpio evitará que llevemos a nuestro proyecto a poseer algún tipo de deuda que no es más que las consecuencias de un software malo.

La toma de decisiones frente a los errores presentados en el desarrollo de software y el cómo manejar las incidencias nos llevará a entregar un software con o sin calidad.

Decisiones irresponsables generará deuda técnica, a pesar de esto la deuda incurrida puede ser de manera intencionada en el sentido de utilizar un equipo de trabajo sin una experiencia adecuada en desarrollar y por ende el software no será óptimo.

El saber controlar la deuda técnica intencionada a largo plazo es un trabajo estratégico, sin embargo, el entregar un proyecto en el menor tiempo, con el menor costo no siempre es sinónimo de un buen proyecto de software de buena calidad por consecuente se convierte en un proyecto casi indefinido por todos los cambios que se tienen que realizar a nivel de mantenimiento, sin dejar de lado que el cliente estará atado a facturas por soporte, incluso la deuda técnica puede darse por no hacer lo que se tenía que hacer en el momento que se tenía que hacer.

Un proyecto de software actual puede traer errores, defectos o incluso deuda técnica de otros proyectos de desarrollo:

- Proyectos en los cuales no se tomó en cuanta la calidad.
- Implementar más recursos para poder corregir un software el cual no se documentó correctamente.
- Tiempos de entrega no acorde a lo que toma desarrollar el producto.

Para medir el grado de deuda técnica en el cual está incurriendo un proyecto existe SQALE (Software Quality Assessment based on Life-cicle Expectations) se enfoca en medir la calidad del software según su deuda técnica. (Abad, 2016)

Ilustración 11 ISO 25000 Calidad del producto de software

(Cunningham, 2015)

El software al no ser algo tangible tenemos muchas aristas que pivota sobre el termino calidad la familia de normas ISO 25000 nos permiten añadir calidad al producto de desarrollo de software y la define de la siguiente forma los factores externos que afectan al software como el ambiente en el que correrá la aplicación, lo desarrollado es lo que esperaba el cliente recibir, su constitución interna. Para solventar estas incógnitas nos ayudarán el determinar y medir la calidad interna del software la mantenibilidad nos indica en qué grado puede ser el producto modificado, eficiencia el cómo optimizar todos los recursos, portabilidad en la cual un sistema de software pueda cambiar entre diferentes plataformas de hardware o software, fiabilidad hace referencia a que la aplicación cumpla la función con los requerimientos definidos

Para medir la trazabilidad entre el requerimiento y el producto de software tenemos pruebas de caja blanca. Sin embargo, no todas las aplicaciones de software tienen los mismos requerimientos de calidad y en función de esto definiremos que medir en nuestras pruebas de calidad.

Por ejemplo, al desarrollar software para atención al cliente será necesario la usabilidad, si es una aplicación web se toma en cuenta la seguridad, todos estos aspectos son imposibles de validar solo revisando entradas y salidas de procesos para esto utilizamos el testing que es una pieza fundamental en el proceso de calidad, en la calidad del producto es necesario validar los riesgos del negocio y meterse al nivel de código fuente.

En la Universidad Israel se comprobará la calidad del producto de software en uno de sus proyectos de desarrollo interno los requisitos de software se utilizarán para esta revisión son las definiciones proporcionadas por la Organización Internacional de Normalización ISO y la Comisión Electrotécnica Internacional IEC las mismas que fueron planteadas en ISO/IEC 25010 Evaluación de la calidad del producto e ISO/IEC 25040 Evaluación de la calidad del producto de software.

Los requisitos que se utilizarán estarán determinados por el alcance del proyecto a evaluar, así como a conceptos de la norma para la evaluación del producto de software, la calidad según el modelo ISO establece modelos de calidad en procesos mide la eficiencia por medio de herramientas y personas que conozcan cómo aplicar la calidad.

Para este proyecto se lo abordará de la siguiente manera:

1.- Modelos de calidad ISO/IEC 25010 está compuesta por diversas características y sub características las mismas que componen métricas que nos permitirán conocer el grado en que el producto de software cumple con lo establecido en el contrato o necesidad del cliente

2.- Procesos por medio de la ISO/IEC 25040 la misma que me indicará como evaluar, que seleccionar para evaluar, como identificamos los resultados los mismos son descrito y presentados en un informe el cual se obtendrá en función a cinco actividades:

- Métricas.
- Diseñar la evaluación.
- Ejecutar la evaluación.

- Generar informe de resultados.

3.- Herramientas que soporten y puedan medir el producto y el proceso las mismas que está compuesto por entradas, salidas y presentación de resultados. Estas herramientas nos permitirán reducir tiempos en las evaluaciones y en el análisis ya sea estático o dinámico.

Hay que tener en cuenta que las herramientas miden no evalúan la medición nos permitirá tener indicadores con los cuales determinar lo siguiente:

- El porcentaje de código copiado y pegado.
- Demasiados case en un switch conocido como código espagueti.

En el ecosistema de la evaluación del producto las herramientas representan de manera gráfica indicadores que presenta información detallada la misma puede obtenerse históricamente y de esa manera comparar con resultados anteriores.

Existen empresas que se dedican a acreditar el software que se las conoce como laboratorios de software y empresas que están encargadas de acreditarlas FINDING, SonarQube, Jenkins son un ejemplo de herramienta para el control de software que está alineada con las normas ISO 25000.

Estos laboratorios con ayuda de herramientas nos permitirán permitir varias condiciones y características el costo por cada evaluación que se mide en función al número de líneas que tiene el código y la técnica que requiere el evaluador para su producto de software.

Una vez que la empresa evaluada ha corregido los errores enviados en el informe se procede a evaluar nuevamente el software. Al estar el software sin errores o con la calidad mínima que solicito la empresa o laboratorio emite un certificado para la calidad del producto.

Un ejemplo AENOR revisa como se verifica la calidad y si el laboratorio evaluó o no el software enviado.

La necesidad primordial en esta investigación se enfocará al producto de software del proyecto de Titulación de la Universidad Israel la misma que será capaz de cubrir todos los aspectos definidos previamente por el documento de requerimiento y posteriormente un informe final en el cual detalla los diferentes ámbitos evaluados y así garantizar que dicho producto consta con la calidad deseada para ser funcional evitando así la deuda técnica que pueda tener el producto de software.

Por tanto, al buscar la calidad del producto se be considerar que no todas las pruebas se las puede hacer de forma automática ya que existen escenarios que no se los podrá automatizar ya que al estar aislado al equipo de desarrollo puede ser un problema.

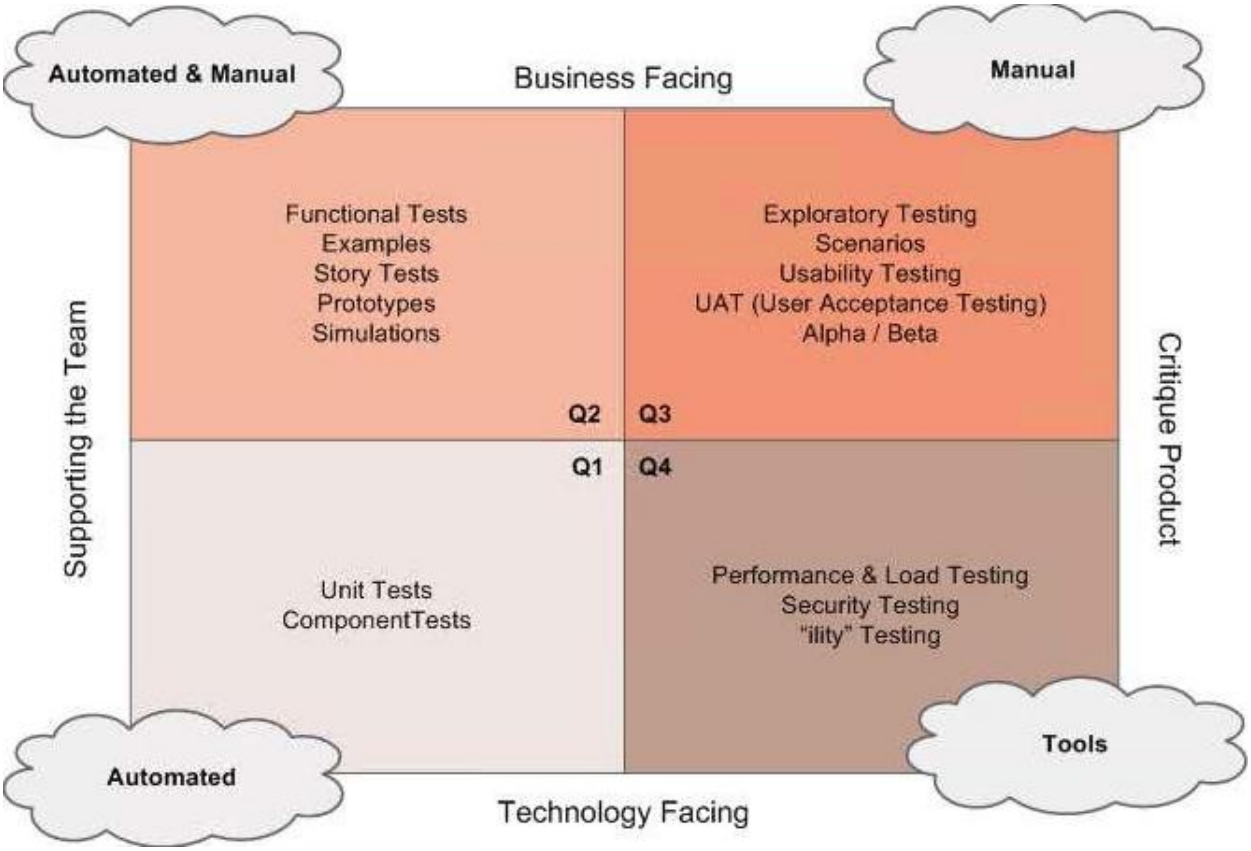


Ilustración 12 Distribución de pruebas según el cuadrante

(Crispin, 2008)

- Cuadrante 1. Pruebas unitarias y de componentes, que normalmente se recomienda automatizar.
- Cuadrante 2. Pruebas que pueden realizarse de manera automática o manual, y que suelen ser las pruebas funcionales, simulaciones, prototipos.
- Cuadrante 3. Pruebas manuales, que suelen ser las de usabilidad, aceptación.
- Cuadrante 4. Herramientas que se hacen con herramientas, como son las de rendimiento y carga.

2. PROPUESTA

"SonarQube es una plataforma desarrollada en Java que nos permite realizar análisis de código con diferentes herramientas de forma automatizada. Es software libre y 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 un programa.

Es una herramienta esencial para nuestra fase de testing y auditoria de código dentro de nuestro ciclo de desarrollo de nuestra aplicación. Existen muchos problemas que se pueden encontrar en un código. Las reglas difieren y pueden clasificarse en 5 grupos según su severidad: Bloqueador, crítico, grave, menor e informativo. Así que, si hay un bug o un bug potencial, va a ser clasificado como problema bloqueador o crítico, y algunos problemas como "los números mágicos no se deben utilizar" van a ser clasificados con severidad menor o informativa. Y deberá volver a la fase de desarrollo para corregir esas partes de código."

(Websec, 2017)

2.1 EVALUACIÓN DE CALIDAD DEL PRODUCTO DE SOFTWARE SEGÚN ISO/IEC 25010/25040

La calidad del producto software se mide en función en que el producto de software cumple con lo solicitado por la Universidad Israel y agrega un valor a las funciones cotidianas por medio del desarrollo interno de aplicaciones.

La evaluación se la realizará por medio de parámetros establecidos en la ISO 25010 e ISO 25040.

2.1.1 ADECUACIÓN FUNCIONAL

Representa la capacidad del producto software para proporcionar funciones que satisfacen las necesidades declaradas e implícitas.

a. Requisitos: Documento inicial con los requisitos y el detalle de cómo y para que fue diseñado el requerimiento de desarrollo de software, además el plan de pruebas funcionales realizadas por el equipo de desarrollo en el que se detalla cómo evaluaron el software.

b. Completitud funcional: Grado en el cual el conjunto de funcionalidades cubre todas las tareas y los objetivos del usuario especificados.

c. Corrección funcional: Capacidad del producto o sistema para proveer resultados correctos con el nivel de precisión requerido.

d. Pertinencia funcional: Capacidad del producto software para proporcionar un conjunto apropiado de funciones para tareas y objetivos de usuario especificados.

c. Reglas configuradas: N/A.

e. Método de evaluación: Tomas los requisitos con los cuales se procedió al desarrollo de la aplicación y analizar en forma de checklist indicando si cumple o no con la evaluación. En la lista de ítem se nombran todos los casos de prueba analizados individualmente por medio de las funcionalidades en las que se determinó su grado midiendo del 1 al 5.

ITEM	Cumple: SI / NO	Observaciones
Módulo de registro para iniciar sesión en la aplicación		
Existe la función para recuperar la contraseña o usuario en caso de olvidarlo		

Tabla 7 Métodos de evaluación

2.1.2 EFICIENCIA DE DESEMPEÑO

Esta característica representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones.

a. Requisitos: Aplicación compilada en el ambiente creado para realizar las respectivas pruebas, con una instancia diferente de base de datos utilizada para realizar el desarrollo, de preferencia un ambiente similar a producción para que los resultados sean los correctos.

b. Comportamiento temporal: Los tiempos de respuesta y procesamiento, el throughput (volumen de trabajo) de un sistema cuando lleva a cabo sus funciones bajo condiciones determinadas en relación con un banco de pruebas (benchmark) establecido.

c. Utilización de recursos: Las cantidades y tipos de recursos utilizados cuando el software lleva a cabo su función bajo condiciones determinadas.

d. Capacidad: Grado en que los límites máximos de un parámetro de un producto o sistema software cumplen con los requisitos.

e. Reglas configuradas: N/A

f. Método de evaluación: Una vez conocido el software instalado, se procederá a analizar si existe algún conflicto, si los recursos del sistema son suficientes para que la aplicación se ejecute correctamente.

2.1.3 COMPATIBILIDAD

Capacidad de dos o más sistemas o componentes para intercambiar información y/o llevar a cabo sus funciones requeridas cuando comparten el mismo entorno hardware o software.

a. Requisitos: Conocer las características de software y hardware en la cual será instalado el sistema desarrollado se va a instalar.

b. Coexistencia: Capacidad del producto para coexistir con otro software independiente, en un entorno común, compartiendo recursos comunes sin detrimento.

c. Interoperabilidad: Capacidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada.

d. Reglas configuradas: N/A

2.1.4 USABILIDAD

Capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones.

a. Requisitos: Un usuario que no haya usado antes la aplicación para utilizarlo por primera vez.

b. Capacidad para reconocer su adecuación: Capacidad del producto que permite al usuario entender si el software es adecuado para sus necesidades.

c. Capacidad de aprendizaje: Capacidad del producto que permite al usuario aprender su aplicación.

d. Capacidad para ser usado: Capacidad del producto que permite al usuario operarlo y controlarlo con facilidad.

e. Protección contra errores de usuario: Capacidad del sistema para proteger a los usuarios de hacer errores.

f. Estética de la interfaz de usuario: Capacidad de la interfaz para agradar y satisfacer la interacción con el usuario.

g. Accesibilidad: Capacidad del producto que permite que sea utilizado por usuarios con determinadas características y discapacidades.

h. Reglas configuradas: N/A.

i. Método de evaluación: Conclusiones por parte del que ejecuta las pruebas indicando "sí".

2.1.5 FIABILIDAD

Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados.

a. Requisitos: Sistema instalado en un ambiente de pruebas.

b. Madurez: Capacidad del sistema para satisfacer las necesidades de fiabilidad en condiciones normales.

c. Disponibilidad: Capacidad del sistema o componente de estar operativo y accesible para su uso cuando se requiere.

d. Tolerancia a fallos: Capacidad del sistema o componente para operar según lo previsto en presencia de fallos hardware o software.

e. Capacidad de recuperación: Capacidad del producto software para recuperar los datos directamente afectados y reestablecer el estado deseado del sistema en caso de interrupción o fallo.

f. Reglas configuradas: N/A

g. Método de evaluación: Utilizarlo en el ambiente de test, estas pruebas se las realizarán durante un tiempo determinado, y en este tiempo analizar cómo se comporta el software y presentar un informe final sobre este caso así conocer la fiabilidad del software.

h. INFORME FINAL

Prueba Funcional: Fiabilidad del software Especificar el módulo o la solución total sobre el cual se realizará las respectivas pruebas, conociendo el alcance del proyecto **Listar Dependencias:** Orden lógico del desarrollo y la utilización siguiendo el manual funcional, casos de uso, historias de usuario, documento inicial.
Elaborado por: Nombre de la persona o equipo de trabajo
Objetivo:

Tabla 8 Informe Final

2.1.6 SEGURIDAD

Capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no puedan leerlos o modificarlos.

a. Requisitos: Código fuente de la aplicación.

b. Confidencialidad: Capacidad de protección contra el acceso de datos e información no autorizados, ya sea accidental o deliberadamente.

c. Integridad: Capacidad del sistema o componente para prevenir accesos o modificaciones no autorizados a datos o programas de ordenador.

d. No repudio: Capacidad de demostrar las acciones o eventos que han tenido lugar, de manera que dichas acciones o eventos no puedan ser repudiados posteriormente.

f. Responsabilidad: Capacidad de rastrear de forma inequívoca las acciones de una entidad.

g. Autenticidad: Capacidad de demostrar la identidad de un sujeto o un recurso.

h. Reglas

- **El registro en la consola no debe utilizarse.**

Se produce al insertar código de depuración en el ambiente de producción, en particular en el código que ejecuta cliente, se corre el riesgo de exponer inadvertidamente información confidencial.

```
private void DoSomething ()

{

 // ...

 Console.WriteLine ("Si ingresa aquí el código esta bien");// Noncompliant

 // ...
```

Tabla 9 Ejemplo de registro en consola

- **Las direcciones IP no deben ser codificadas**

Codificar una dirección IP en el código fuente es una mala idea por varias razones:

- Se requiere una re-compilación si la dirección cambia.
- Obliga a usar la misma dirección en todos los entornos (desarrollo, test, aceptan test, producción).
- Pone la responsabilidad de establecer el valor a utilizar en producción sobre los hombros del desarrollador.

El siguiente código no compila.

```
var ip = "127.0.0.1";

var address = IPAddress.Parse(ip);
```

Tabla 10 Ejemplo IP no compila

Solución.

```
var ip = ConfigurationManager.AppSettings["myapplication.ip"];

var address = IPAddress.Parse(ip);
```

Tabla 11 Ejemplo IP que compila

- **Comprobación de desbordamiento no debe deshabilitarse para Enumerable.Sum.**

Enumerable.Sum () siempre ejecuta la adición en un contexto marcado, por lo que se lanzará una OverflowException si el valor excede a MaxValue incluso si se especificó un contexto no comprobado. Usar un contexto sin verificar de todos modos representa un malentendido de cómo funciona Sum.

Esta regla plantea un problema cuando se especifica un contexto no comprobado para una suma en tipos enteros.

Código que no compila.

```
void Add(List<int> list)
{
 int d = unchecked(list.Sum());  // Noncompliant
 unchecked
 {
  int e = list.Sum();  // Noncompliant
 }
}
```

Tabla 12 Ejemplo SUM no compila

Solución.

```
void Add(List<int> list)
{
 int d = list.Sum();
 try
 {
  int e = list.Sum();
 }
 catch (System.OverflowException e)
 {
  // exception handling...
 }
}
```

Tabla 13 Ejemplo SUM que compila

Excepciones.

Cuando la llamada a Sum () está dentro de un bloque try-catch, no se informan problemas.

```
void Add(List<int> list)
{
  unchecked
  {
    try
    {
      int e = list.Sum();
    }
    catch (System.OverflowException e)
    {
      // exception handling...
    }
  }
}
```

Tabla 14 Ejemplo SUM con excepción

2.1.6.8 Método de evaluación: Ejecutar la aplicación SonarQube y verificar el número de vulnerabilidades que presenta el código fuente, para posteriormente corregir.

2.1.7 MANTENIBILIDAD

Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas.

a. **Requisitos:** Código fuente para ser analizado, con todos sus módulos.

b. **Modularidad:** Capacidad de un sistema o programa de ordenador (compuesto de componentes discretos) que permite que un cambio en un componente

tenga un impacto mínimo en los demás.

c. **Reusabilidad:** Capacidad de un activo que permite que sea utilizado en más de un sistema software o en la construcción de otros activos.

d. **Capacidad de ser analizado:** Facilidad con la que se puede evaluar el impacto de un determinado cambio sobre el resto del software, diagnosticar las deficiencias o causas de fallos en el software, o identificar las partes a modificar.

e. **Capacidad para ser modificado:** Capacidad del producto que permite que sea modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño.

f. **Capacidad para ser probado:** 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.

g. **Reglas configuradas:** N/A

h. Modo de evaluación: Verificar el código por medio de juicio de expertos analizar la forma en que se desarrolló, en ese análisis confirmar si es posible añadir una nueva funcionalidad, si el código está parametrizado o a su vez la forma en que se integró la capa de datos con la de aplicación.

2.1.8 PORTABILIDAD

Capacidad del producto o componente de ser transferido de forma efectiva y eficiente de un entorno hardware, software, operacional o de utilización a otro.

a. Requisitos: Aplicación compilada y manual de instalación.

b. Adaptabilidad: Capacidad del producto que le permite ser adaptado de forma efectiva y eficiente a diferentes entornos determinados de hardware, software, operacionales o de uso.

c. Capacidad para ser instalado: Facilidad con la que el producto se puede instalar y/o desinstalar de forma exitosa en un determinado entorno.

d. Capacidad para ser reemplazado: Capacidad del producto para ser utilizado en lugar de otro producto software determinado con el mismo propósito y en el mismo entorno.

e. Reglas configuradas: N/A

f. Modo de evaluación: Guiarse por medio del manual y proceder a instalar si la aplicación se instala y funciona correctamente, el caso de pruebas habría sido ejecutado exitosamente.

3. CONDICIONES GENERALES PARA LA EJECUCIÓN DE PRUEBAS

A continuación, se enuncian las funcionalidades que se desean probar en escala de valoración de 1 a 5. Plantilla de ejemplo para evaluaciones

Objetivo	
Hardware Requerido	
Software Requerido	
No. de Usuarios	
Procedimiento de Prueba	1. Definir usuario de pruebas 2. Lista de tareas funcionales del proyecto en las cuales se valorará según su escala 3. Una vez terminada la prueba, deberá remitir el resultado al equipo de desarrollo.

Tabla 15 Condiciones generales para la ejecución de pruebas

3.1 EVALUACIÓN DE LAS MÉTRICAS DE CALIDAD

CUMPLIMIENTO DE LOS FACTORES DE CALIDAD	C: Cumple (Escala 1 a 5) NC: No Cumple (Escala 1 a 5) NR: No requerido (Escala 1 a 5) VE: Valor Estimado (Escala 1 a 5)

Métrica a Evaluar	C	NC	NR	VE	Total	Observaciones
1. Adecuación funcional						
2. Eficiencia de desempeño						
3. Compatibilidad						
4. Usabilidad						
5. Fiabilidad						
6. Seguridad						
7. Mantenibilidad						
8. Portabilidad						

Prueba Ejecutada por:	Firma	Fecha

Tabla 16 Evaluación Métricas de calidad

4. RESULTADOS

Para el análisis de la solución SonarQube se ha seleccionado el proyecto denominado Sistema Académico desarrollado en la Universidad Israel.

Sobre este código fuente se ha aplicado las reglas mencionadas en la propuesta de este trabajo y las reglas de las mismas para analizar su adecuación funcional, eficiencia, compatibilidad, usabilidad, fiabilidad, seguridad, mantenibilidad y portabilidad.

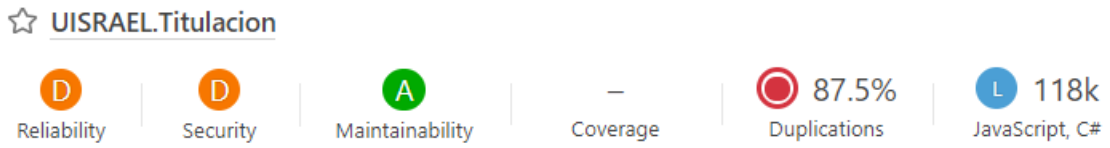


Ilustración 13 Calificación obtenida del proyecto

4.1 ANÁLISIS DE ERRORES Y VULNERABILIDADES

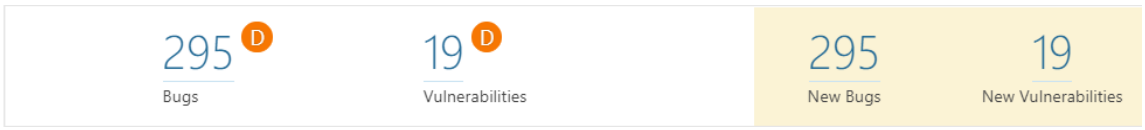


Ilustración 14 Errores y vulnerabilidades

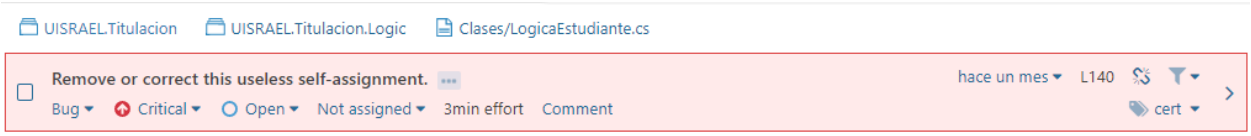


Ilustración 15 Error 1 proyecto UISRAEL.Titulación

No hay ninguna razón para volver a asignar una variable a sí mismo. O bien esta sentencia es redundante y debe ser eliminada, o la reasignación es un error y algún otro valor o variable estaba destinado a la asignación en su lugar.

Noncompliant Code Example

```
public void SetName(string name)
{
  name = name;
}
```

Compliant Solution

```
public void SetName(string name)
{
  this.name = name;
}
```

Ilustración 16 Referencia error 1

Código fuente del proyecto UISRAEL.Titulación en el cual se extrae la sección que indica que se ha incumplido la regla de eficiencia ya que se está redundando el código.

```
public static DocumentoIngresoEstudianteModel AgregarDocumentoIngresoEstudiante(
 DocumentoIngresoEstudianteModel model)
    {
      try
      {
        using (var ts = new TransactionScope())
        {
          var db = new UniversidadIsraelEntities();
          DateTime fechaActual = DateTime.Now;
          var documentoIngreso = new EstudianteDocumentoIngreso()
          {
            IdEstudiante = model.IdEstudiante,
            Entregado = model.Entregado,
            IdDocumentoIngreso = model.IdDocumentoIngreso,
            Fecha = fechaActual,
            FechaEntregado = model.FechaEntregado,
            FechaActualización = fechaActual,
            IsActivo = true
          };
          db.EstudianteDocumentoIngresos.Add(documentoIngreso);
          db.SaveChanges(model.UsuarioAccion);
          model.IdDocumentoIngresoTipo = model.IdDocumentoIngresoTipo;
```

4.2 CÓDIGO PROBLEMÁTICO O CON DEUDA TÉCNICA

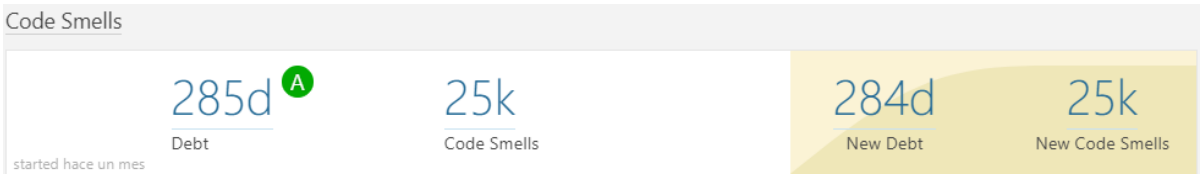


Ilustración 17 Calificación en código que puede causar deuda técnica

Ilustración 18 Error 2 proyecto UISRAEL.Titulación

Si se declara una variable local pero no se utiliza, es código muerto y se debe quitar. Hacerlo mejorará la capacidad de mantenimiento porque los desarrolladores no se preguntarán para qué se utiliza la variable.

Ilustración 19 Referencia error 2

Código fuente del proyecto UISRAEL.Titulación en el cual se extrae la sección que indica que se ha incumplido la regla de código muerto al declarar variables y no utilizarlas.

```
#region Operaciones CRUD

/// <summary>
```

```
        /// Obtengo los usuarios pro sistema
        /// </summary>
        /// <param name="idSistema"></param>
        /// <returns></returns>
        public static IQueryable<UsuarioGridModel> ObternerUsuariosSistema(int
idSistema)
        {
            try
            {
                var db = new UniversidadIsraelEntities();

                var datosSistema = db.Sistemas.Where(w => w.IdSistema == idSistema).Select(s
=> s).FirstOrDefault();
```

4.3 CÓDIGO DUPLICADO

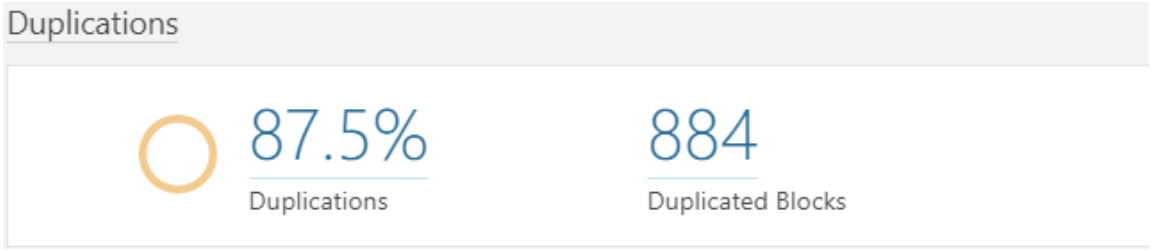


Ilustración 20 Calificación en duplicidad de código

	Duplicated Lines (%)	Duplicated Lines
Scripts/kendo/2015.2.902/kendo.web.min.intellisense.js	100.0%	13,151
Scripts/kendo/2015.2.902/kendo.web.min.intellisense.js	100.0%	13,151
Scripts/kendo/2015.2.902/kendo.web.min.intellisense.js	100.0%	13,151
Scripts/kendo/2015.2.902/kendo.web-vsdoc.js	100.0%	12,696
Scripts/kendo/2015.2.902/kendo.web-vsdoc.js	100.0%	12,696
Scripts/kendo/2015.2.902/kendo.web-vsdoc.js	100.0%	12,696
Scripts/kendo/2015.2.902/kendo.mobile.min.intellisense.js	100.0%	7,515
Scripts/kendo/2015.2.902/kendo.mobile.min.intellisense.js	100.0%	7,515
Scripts/kendo/2015.2.902/kendo.mobile.min.intellisense.js	100.0%	7,515
Scripts/kendo/2015.2.902/kendo.mobile-vsdoc.js	100.0%	7,554
Scripts/kendo/2015.2.902/kendo.mobile-vsdoc.js	100.0%	7,554
Scripts/kendo/2015.2.902/kendo.mobile-vsdoc.js	100.0%	7,554

Ilustración 21 Listado de código duplicado

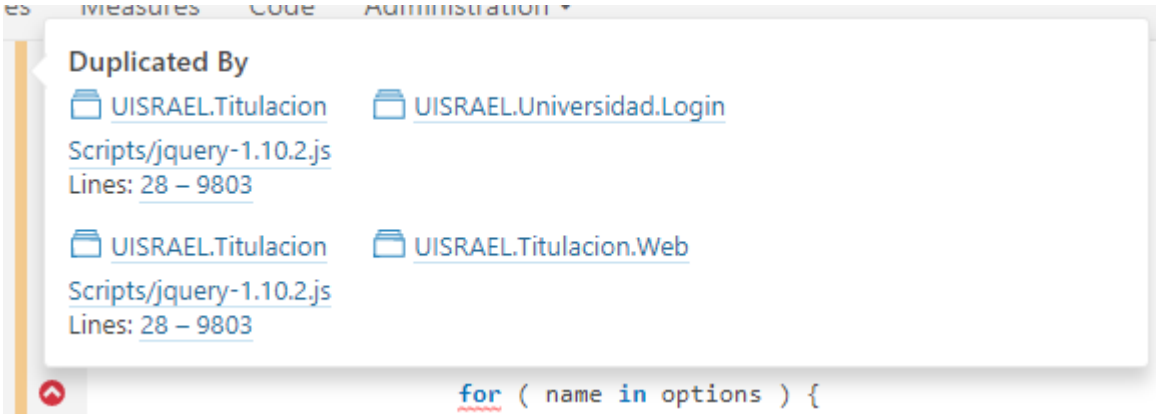


Ilustración 22 Referencia del bloque de código duplicado

SECCION IV

1. CONCLUSIONES

- El seguir un modelo de procesos no asegura la calidad interna del producto de software, es por eso que al aplicar el software SonarQube al desarrollo de software en la Universidad Israel se identificó posibles errores que afectaría a los proyectos de la Universidad.

- La calidad externa de software es importante porque la misma nos asegura que la interacción con el usuario sea natural, fácil de entender, predictiva. Con estas características el trabajo de soporte disminuirá y por ello el grado de satisfacción de quién lo utiliza.

- La mantenibilidad del producto de software asegurará la calidad del uso en el desarrollo de software porque esto facilitará su actualización y mejoramiento de la aplicación.

2. RECOMENDACIONES

- La calidad tiene que estar presente desde siempre, desde que empezamos a pensar el software y en todas sus fases.

- Promovemos calidad del software cuando conseguimos definir y aplicar una estrategia de versionado del software.

- La calidad de un producto la determina el proceso usado para desarrollarlo.

3. BIBLIOGRAFÍA

Abad, J. (6 de Julio de 2016). *SlideShare*. Obtenido de SlideShare: https://es.slideshare.net/jorgeabad1/hablemos-de-deuda-tcnica

Castrillon, I., & Cobos, F. (2016). Implementación de Herramientas para el Aseguramiento de la Calidad en el Desarrollo de Software. *Implementación de Herramientas para el Aseguramiento de la Calidad en el Desarrollo de Software*. Quito, Pichincha, Ecuador.

Crispin, L. (2008). *Agile Testing: A Practical Guide for Testers and Agile Teams.* Boston: Pearson Education.

Cunningham, W. (12 de Febrero de 2015). *Testeandosoftware*. Obtenido de Testeandosoftware: https://testeandosoftware.com/deuda-tecnica-que-es/

Fernández, C., Rodriguez, M., & Piattini, M. (9 de Octubre de 2014). *Calidad del producto de software.* Obtenido de Calidad del producto de software: http://www.en.aenor.es/documentos/certificacion/folletos/calidad_producto_software_ISO25000.pdf

Garzás, J., & Piattini, M. (2006). Object-oriented design knowledge: Principles, heuristics, best practices. *A catalog of object oriented design rules*, 307-347.

Garzás, J., Irrazábal, E., & R, S. E. (2011). Guía práctica de supervivencia en una auditoría CMMI. *Boletín De La ETSII*, 1-33.

Glass, R. L. (2003). Facts and fallacies of software engineering. *Addison Wesley*, 50-62.

ISO. (21 de Octubre de 2015). *DocSlide.* Obtenido de DocSlide: http://documents.tips/documents/la-familia-de-normas-iso-iec-25000.html

iso25000. (01 de 2017). *iso25000*. Obtenido de iso25000: http://iso25000.com/index.php/normas-iso-25000

Larbi, S. (18 de Junio de 2008). *Codeodor*. Obtenido de Codeodor: http://www.codeodor.com/index.cfm/2008/6/18/Common-Excuses-Used-To-Comment-Code-and-What-To-Do-About-Them/2293

Mena, G. (Marzo de 2006). *Mena*. Obtenido de Mena: http://mena.com.mx/gonzalo/maestria/calidad/presenta/iso_9126-3/

Muñoz Sagavinzaga, L., Natavidad Alejos, L. F., Quiroz Villalobos, L., & Villegas Vilcherres, P. (3 de Agosto de 2009). *SlideShare.* Obtenido de SlideShare: http://es.slideshare.net/albert317/calidad-del-producto-software

Rolando, M. (6 de Enero de 2012). *Mindmeister.* Obtenido de Mindmeister: https://www.mindmeister.com/es/93168397/norma-iso-25000-o-square-mr

Short, G. (25 de Junio de 2010). *Technical Debt: SlideShare*. Obtenido de SlideShare: http://www.slideshare.net/garyshort/technical-debt-2985889

Sonarqube: Technical Debt. (11 de Enero de 2016). Obtenido de Sonar Qube: https://docs.sonarqube.org/display/SONARQUBE52/Technical+Debt#TechnicalDebt-Wheretostartpayingthetechnicaldebt?

Websec, J. (4 de Mayo de 2017). *quantika14*. Obtenido de quantika14: http://blog.quantika14.com/blog/2017/05/04/que-es-sonarqube/

4. ANEXOS

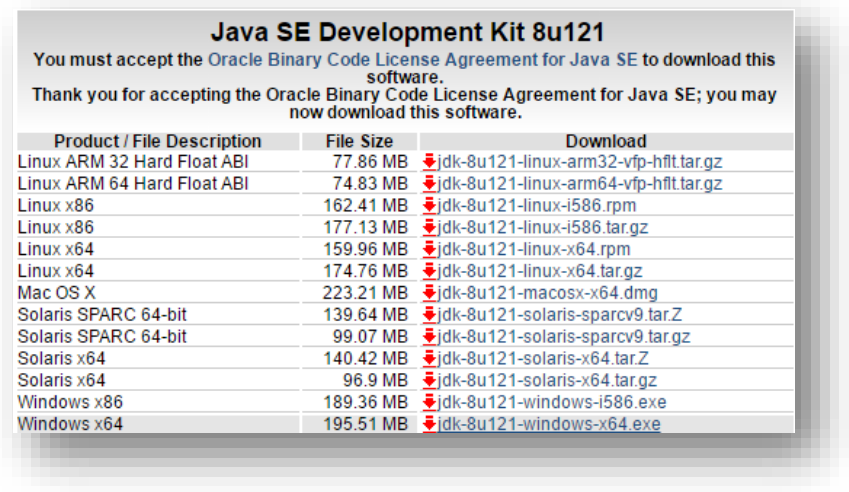
4.1 INSTALACIÓN PRE REQUISITOS

4.1.1 DESCARGAR JAVA

Descargar Java SE para este manual se utilizará 64 bits y para ello ingresamos al siguiente link:

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

En la siguiente pantalla escogemos la versión que se adapta a nuestro sistema operativo.



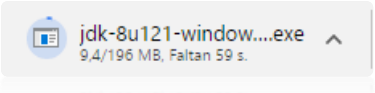
Java SE Development Kit 8u121

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

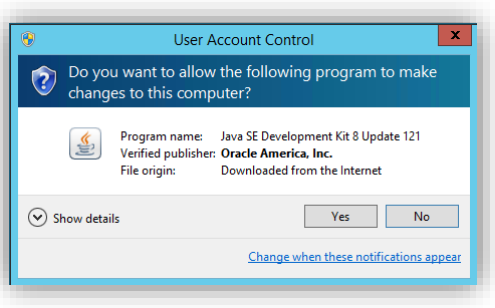
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.86 MB	jdk-8u121-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.83 MB	jdk-8u121-linux-arm64-vfp-hflt.tar.gz
Linux x86	162.41 MB	jdk-8u121-linux-i586.rpm
Linux x86	177.13 MB	jdk-8u121-linux-i586.tar.gz
Linux x64	159.96 MB	jdk-8u121-linux-x64.rpm
Linux x64	174.76 MB	jdk-8u121-linux-x64.tar.gz
Mac OS X	223.21 MB	jdk-8u121-macosx-x64.dmg
Solaris SPARC 64-bit	139.64 MB	jdk-8u121-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.07 MB	jdk-8u121-solaris-sparcv9.tar.gz
Solaris x64	140.42 MB	jdk-8u121-solaris-x64.tar.Z
Solaris x64	96.9 MB	jdk-8u121-solaris-x64.tar.gz
Windows x86	189.36 MB	jdk-8u121-windows-i586.exe
Windows x64	195.51 MB	jdk-8u121-windows-x64.exe

Clic sobre la versión que se ajusta a nuestro entorno, e inmediatamente iniciará la descarga.

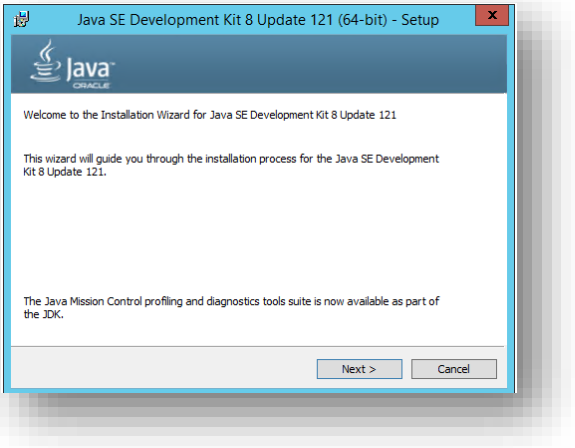


Una vez finalizado el proceso procedemos a ejecutar el archivo, para ello presionamos doble clic sobre el ejecutable y presentara una alerta para permitir al sistema operativo ejecutar la instalación

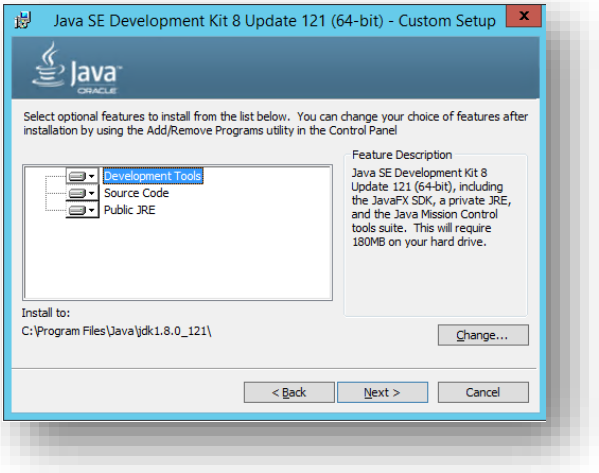
4.1.2 INSTALACIÓN JAVA



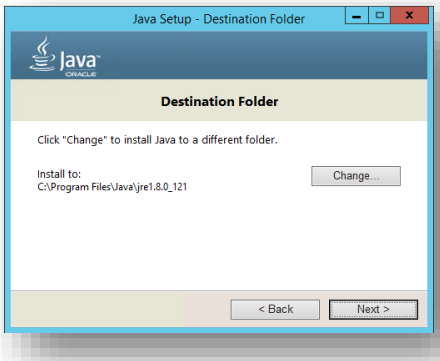
Concedemos permisos para la instalación.



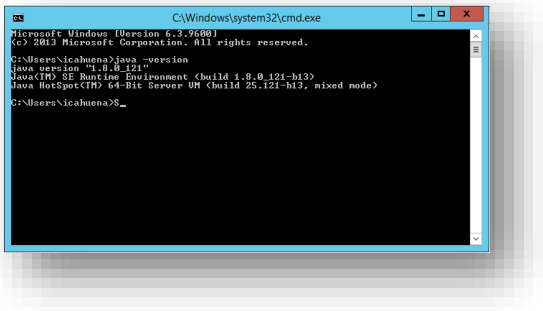
Clic en siguiente.



Determinamos la ruta en la que deseamos instalar java y procedemos a dar clic en siguiente.



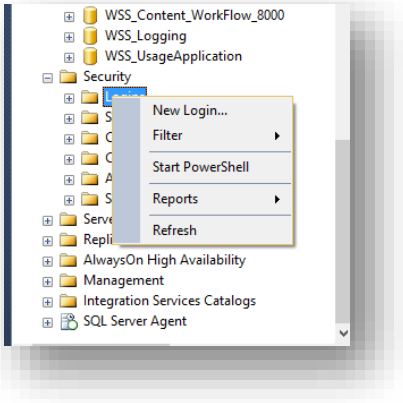
Una vez terminado de instalar procedemos a comprar digitando en la consola de comandos CMD java-version, ejecutado este comando nos indicará la versión que poseemos y nos confirmará que la instalación fue exitosa.



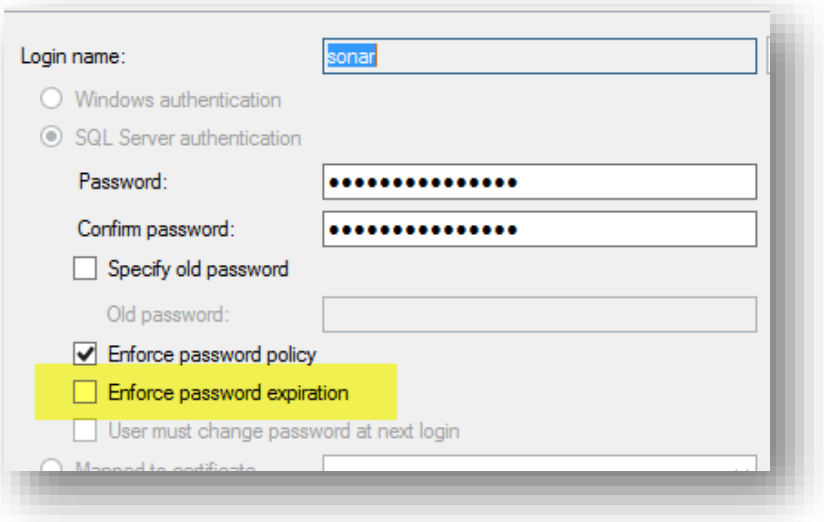
4.2 CONFIGURACIÓN BASE DE DATOS SQL

4.2.1 CREACIÓN DE USUARIO

Nos conectamos a la base de datos con un usuario administrador, ingresamos a la sección de Security, clic derecho en login y presionamos en añadir nuevo usuario.

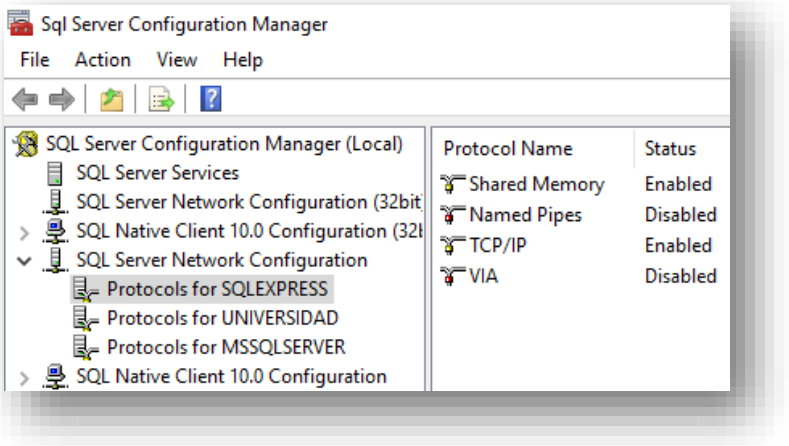


El usuario a crear debe poseer el nombre sonar y una contraseña para este manual se utilizará sonar, quitar el check en la opción para evitar que la contraseña expire.

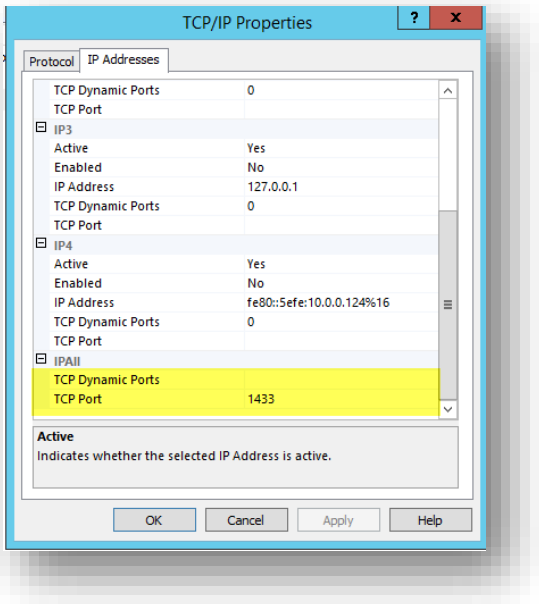


4.2.2 CONFIGURACIÓN DEL PUERTO

Ingresar a Sql Server Configuration Manager.

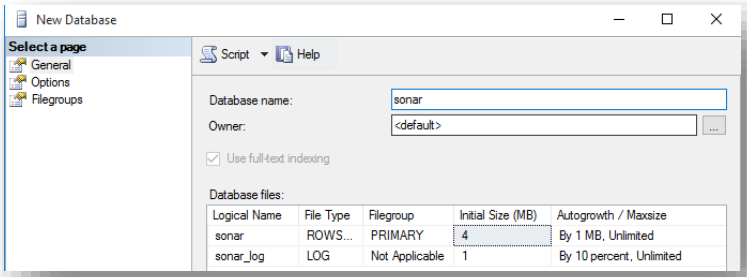


En la sección SQL Server Network Configuration activar TCP/IP e ingresar a propiedades, seleccionar IP Addresses en la división IPAll establecer el TCP Port en 1433.

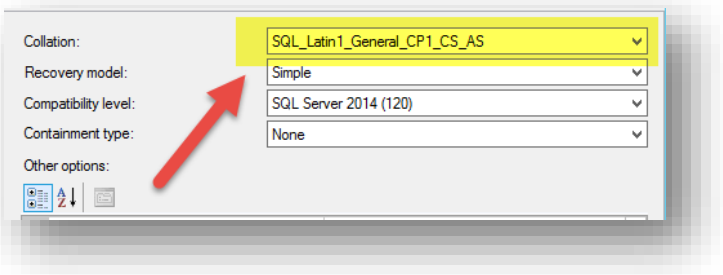


4.2.3 CREAR LA BASE DE DATOS

Ingresar al SQL Management y creamos una base de datos llamada **sonar.**



En opciones avanzadas establecemos el tipo de colección **Case Sensitive and Accent Sensitive, like SQL_Latin1_General_CP1_CS_AS**

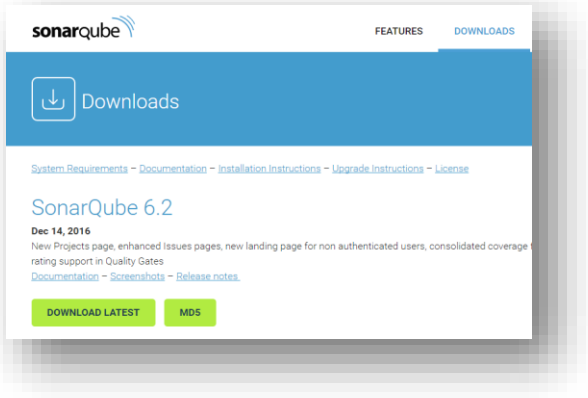


Para comprobar la conexión utilizar el sql management y el usuario sonar utilizando el puerto 1433.

4.3 INSTALACIÓN Y CONFIGURACIÓN DE SONARQUBE

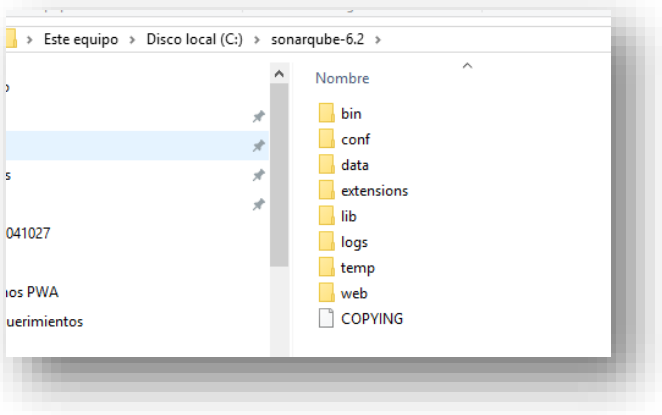
4.3.1 DESCARGAR SONARQUBE

Ingresar a la siguiente dirección https://www.sonarqube.org/



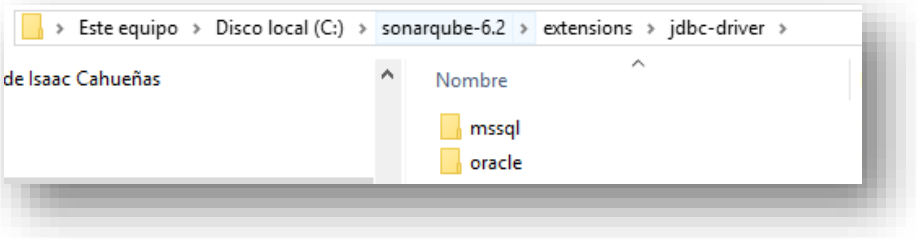
Clic en descargar y luego procedemos a bajar la última versión el archivo se almacenará en la ruta por defecto programada en el navegador, en este caso "Descargas".

Descomprimir el contenido en el disco C:

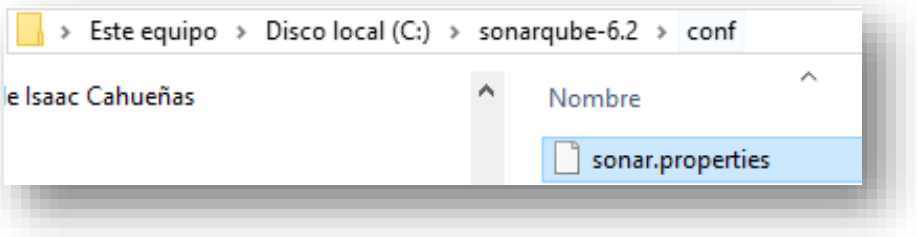


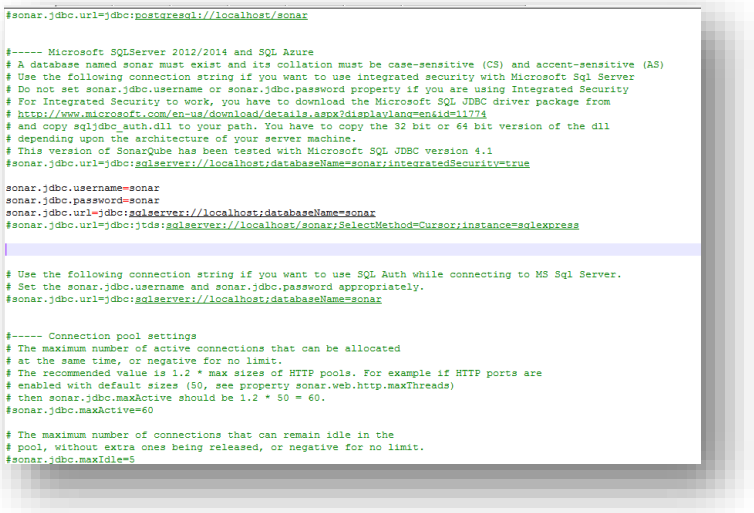
4.3.2 INSTALACIÓN SONARQUBE

En el path C:\sonarqube-6.2\extensions\jdbc-driver procedemos a crear una carpeta llamada **mssql** que es donde almacenará los datos de creación de tablas.



En el path C:\sonarqube-6.2\conf proceder a editar el archivo.



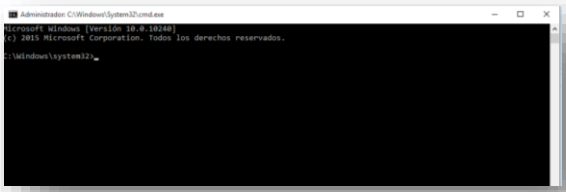


```
#sonar.jdbc.url=jdbc:postgresql://localhost/sonar

#----- Microsoft SQLServer 2012/2014 and SQL Azure
# A database named sonar must exist and its collation must be case-sensitive (CS) and accent-sensitive (AS)
# Use the following connection string if you want to use integrated security with Microsoft Sql Server
# Do not set sonar.jdbc.username or sonar.jdbc.password property if you are using Integrated Security
# For Integrated Security to work, you have to download the Microsoft SQL JDBC driver package from
# http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=11774
# and copy sqljdbc_auth.dll to your path. You have to copy the 32 bit or 64 bit version of the dll
# depending upon the architecture of your server machine.
# This version of SonarQube has been tested with Microsoft SQL JDBC version 4.1
#sonar.jdbc.url=jdbc:sqlserver://localhost;databaseName=sonar;integratedSecurity=true

sonar.jdbc.username=sonar
sonar.jdbc.password=sonar
sonar.jdbc.url=jdbc:sqlserver://localhost;databaseName=sonar
#sonar.jdbc.url=jdbc:jtds:sqlserver://localhost/sonar;SelectMethod=Cursor;instance=sqlexpress


# Use the following connection string if you want to use SQL Auth while connecting to MS Sql Server.
# Set the sonar.jdbc.username and sonar.jdbc.password appropriately.
#sonar.jdbc.url=jdbc:sqlserver://localhost;databaseName=sonar


#----- Connection pool settings
# The maximum number of active connections that can be allocated
# at the same time, or negative for no limit.
# The recommended value is 1.2 * max sizes of HTTP pools. For example if HTTP ports are
# enabled with default sizes (50, see property sonar.web.http.maxThreads)
# then sonar.jdbc.maxActive should be 1.2 * 50 = 60.
#sonar.jdbc.maxActive=60

# The maximum number of connections that can remain idle in the
# pool, without extra ones being released, or negative for no limit.
#sonar.jdbc.maxIdle=5
```

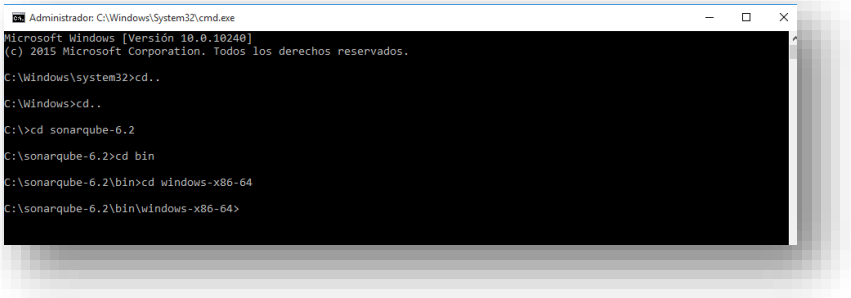
Editamos el archivo quitando los comentario en la sección de Microsoft SQL Server y proporcionamos los datos para la conexión una vez realizado esto proceder a guardar los cambios.

4.3.3 LEVANTAR EL SERVICIO DE SONARQUBE

Ejecutamos la línea de comando en modo administrador.

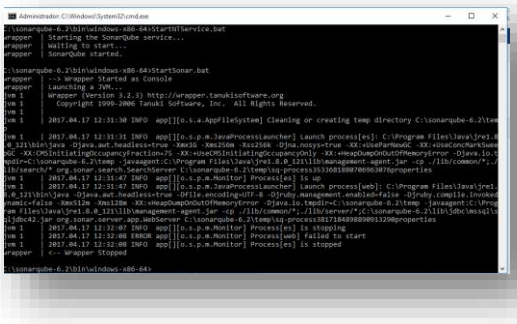


Accedemos a la siguiente ubicación C:\sonarqube-6.2\bin\windows-x86-64>

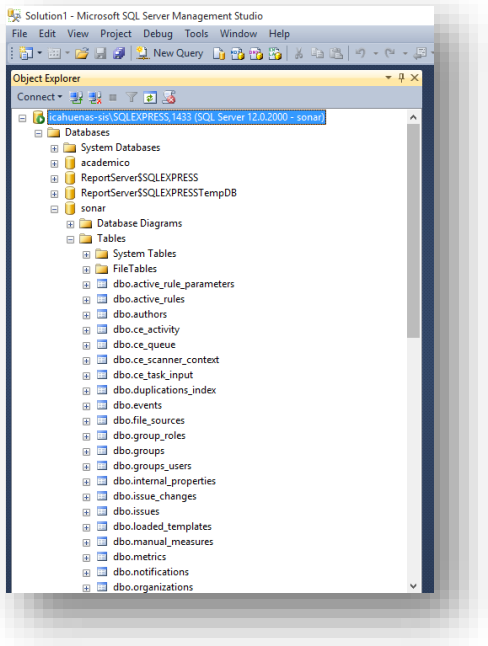


Proceder a levantar el servicio **StartNTService.bat**

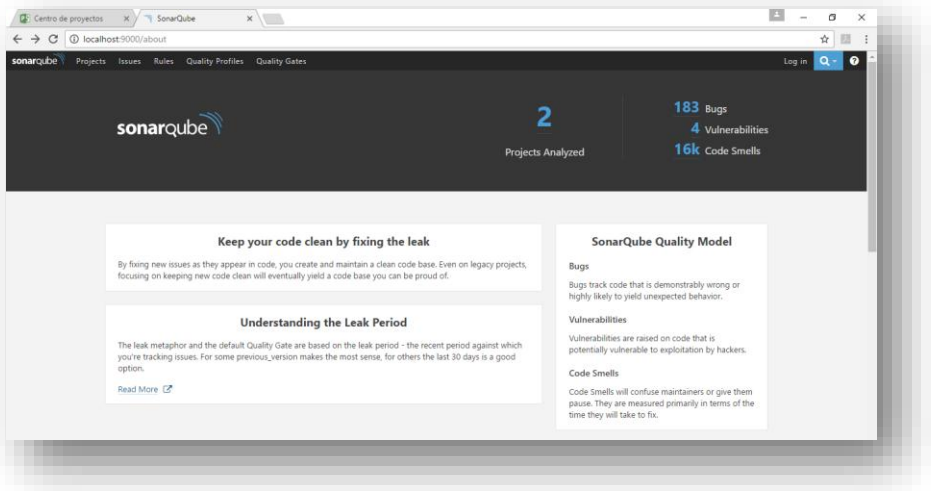
C:\sonarqube-6.2\bin\windows-x86-64>StartNTService.bat



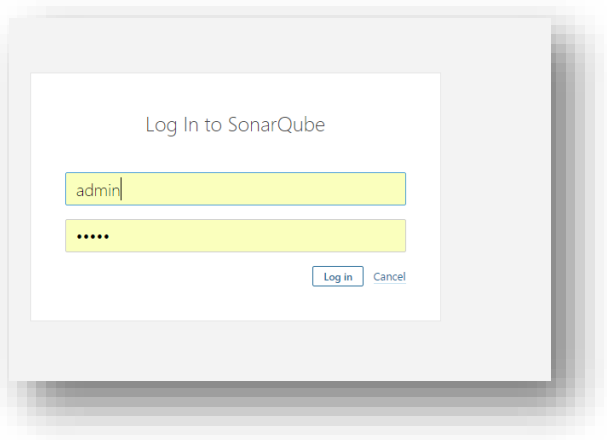
Automáticamente creará las tablas necesarias para la ejecución de las pruebas para comprobar esto ingresar a la instancia de base de datos y confirmar que las tablas se crearon correctamente.

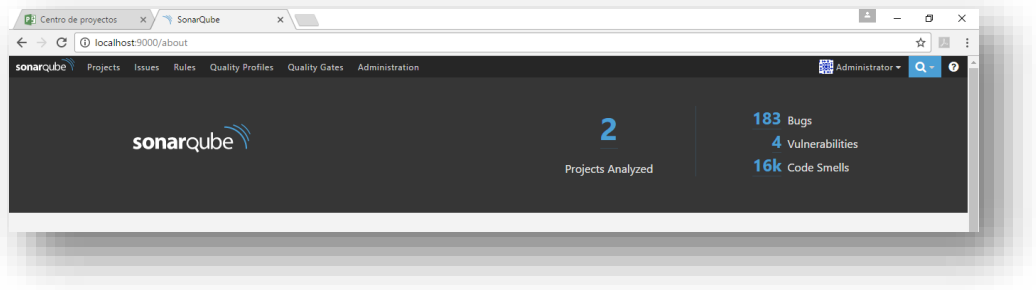


Ya verificado que las tablas están ahí ingresar a la aplicación por medio del navegador http://localhost:9000 y tendremos la interfaz sobre la cual es posible administrar los proyectos.



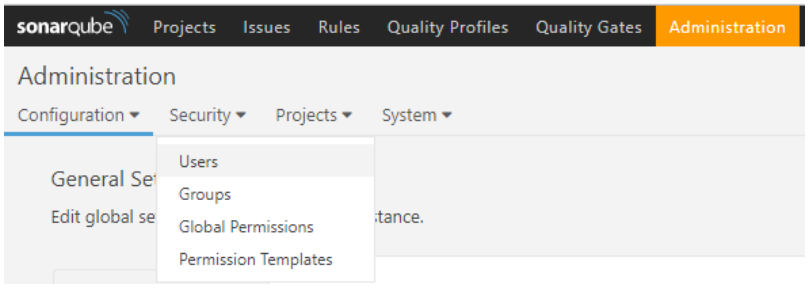
Para iniciar sesión utilizar el usuario **admin** y contraseña **admin**





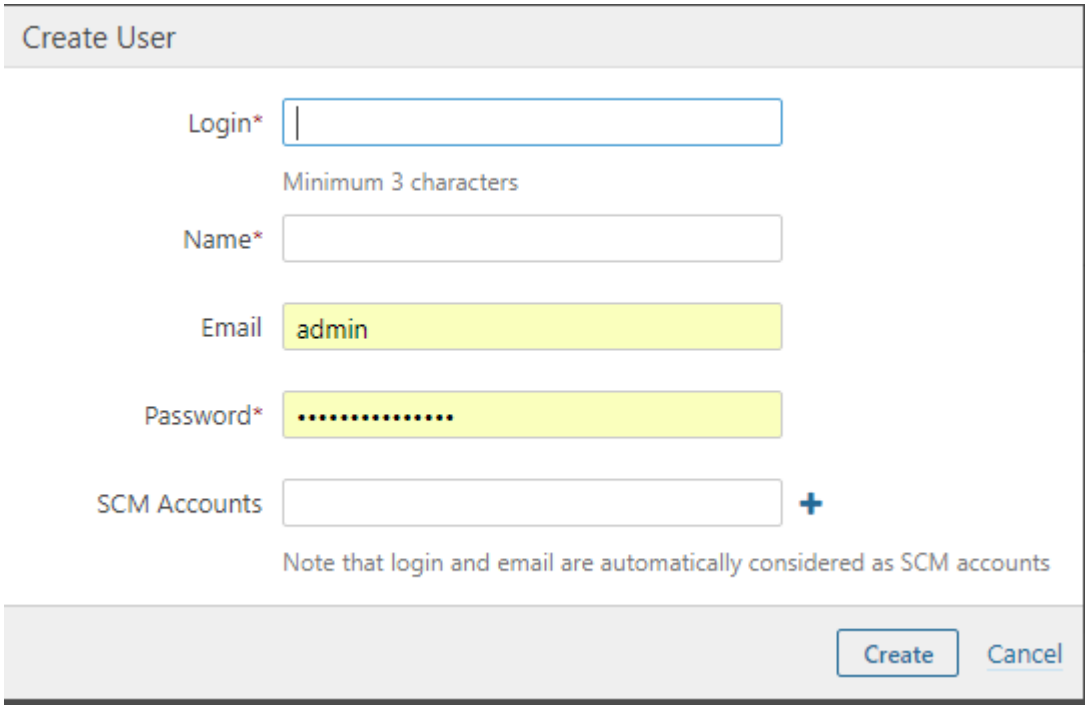
4.3.4 CREACIÓN DE USUARIOS

Iniciar sesión con el usuario administrador, en la sección "Administration" pasamos a "Security" y luego en "Users"



Clic en "Create User".

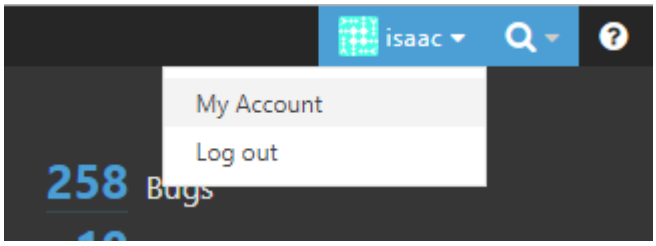
Establecemos los datos que corresponden al usuario.



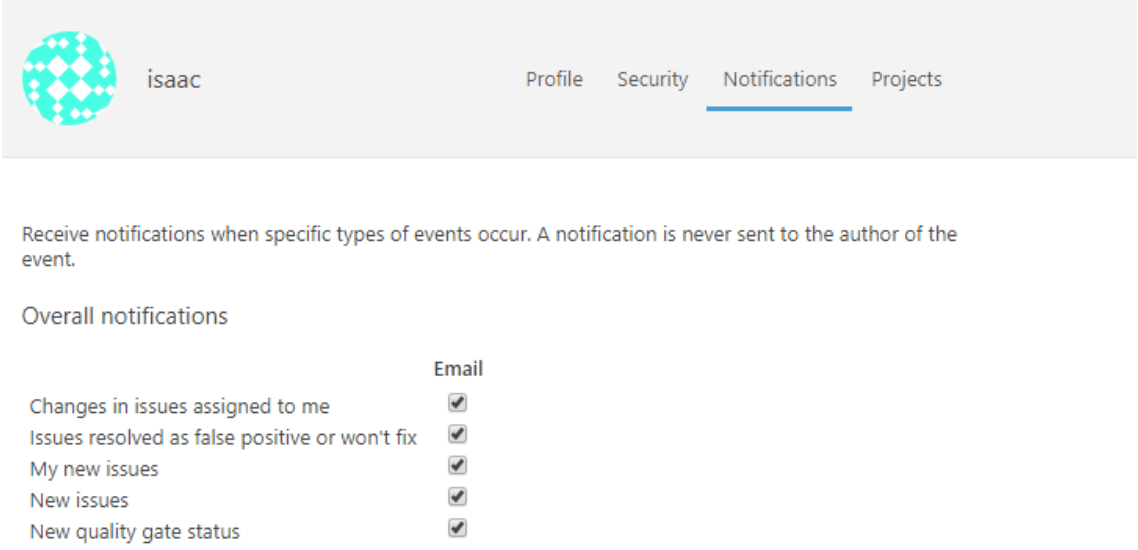
Una vez ingresada toda la información, presionar "Create".

4.3.5 ACTIVAR ENVIO DE NOTIFICACIONES

Iniciar sesión con el usuario que deseamos activar el envío de notificaciones, en la esquina superior derecha en el nombre de usuario presionamos en "My Account".

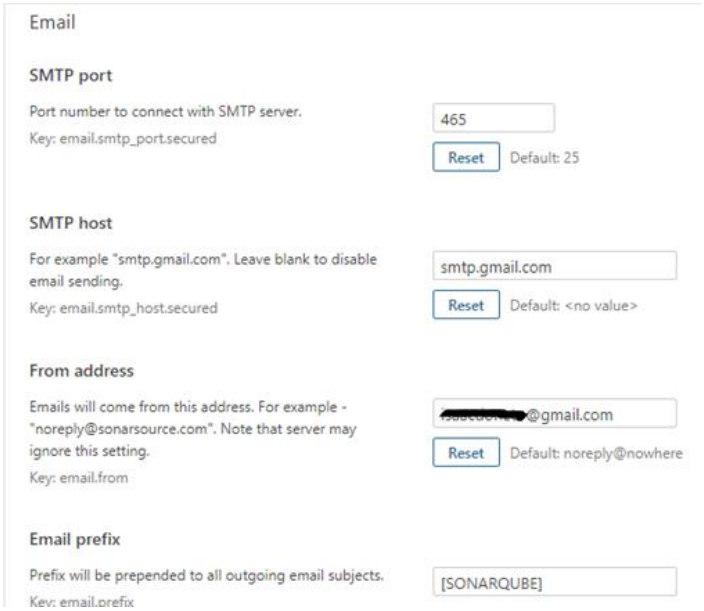


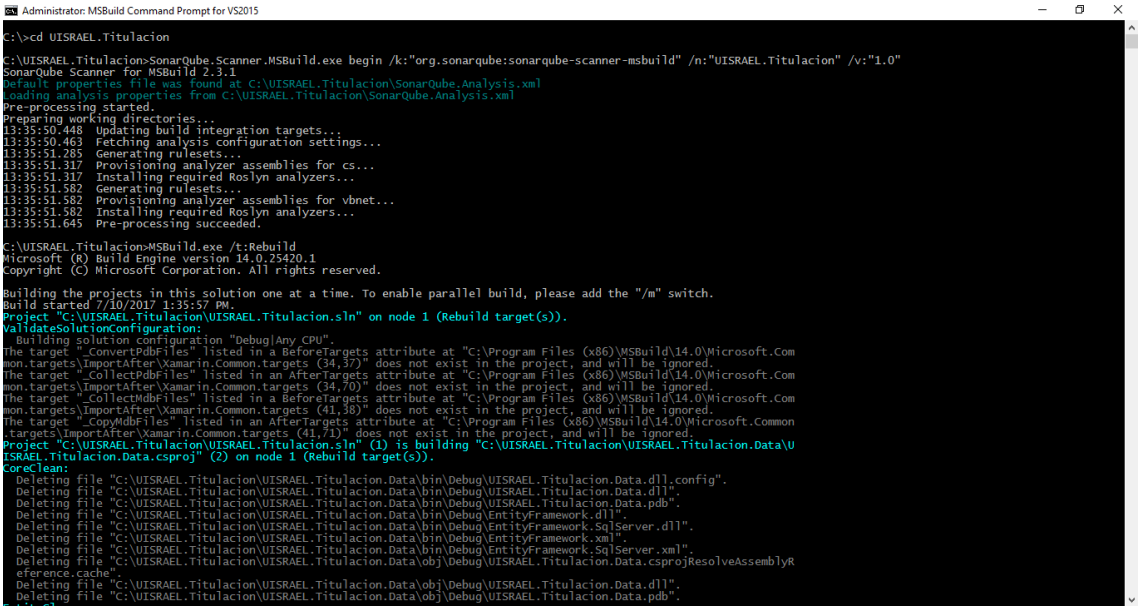
En la sección "Notifications" marcamos el tipo de notificación que se desea recibir



4.3.6 CONFIGURAR SERVIDOR DE CORREO ELECTRÓNICO

En la sección "General" que está en "Administration" en la sección "Email" se ingresa los parámetros del servidor de correo de salida





4.4 EJECUCIÓN DE PRUEBAS CON SONARQUBE

En la carpeta **sonar-scanner-msbuild-2.3.1.554** copiar el código fuente.

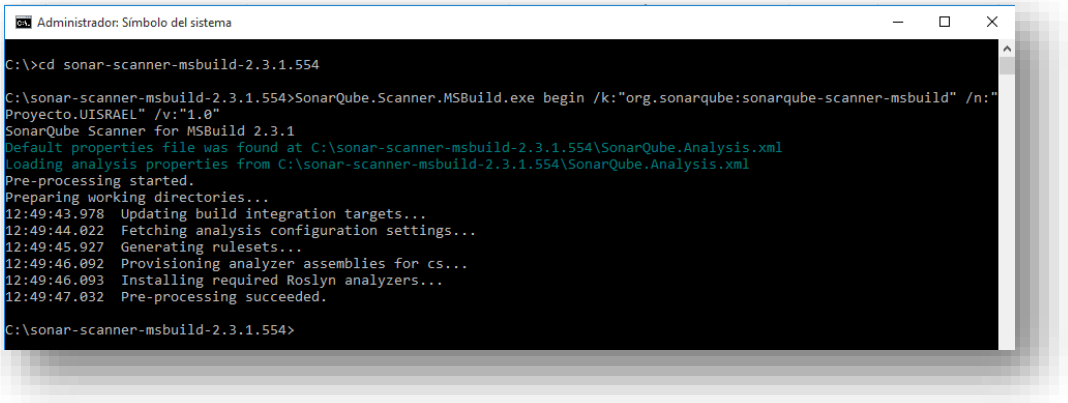
Ejecutamos en el siguiente orden:

SonarQube.Scanner.MSBuild.exe begin /k:"org.sonarqube:sonarqube-scanner-msbuild" /n:"UISRAEL.Titulacion" /v:"1.0"

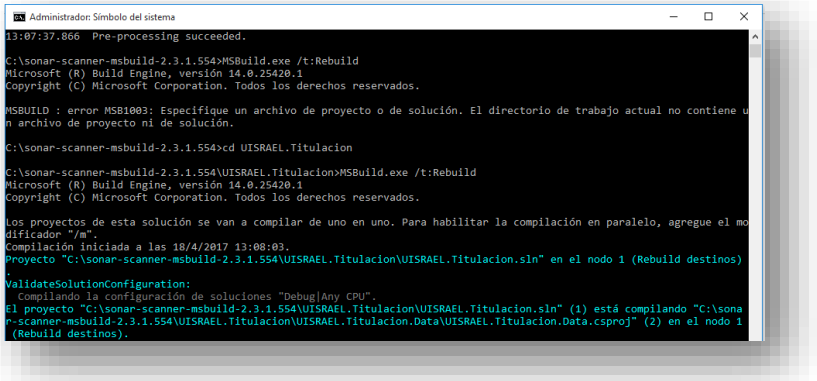
k: Se refiere al key o clave principal que daremos al proyecto

n: Nombre del proyecto

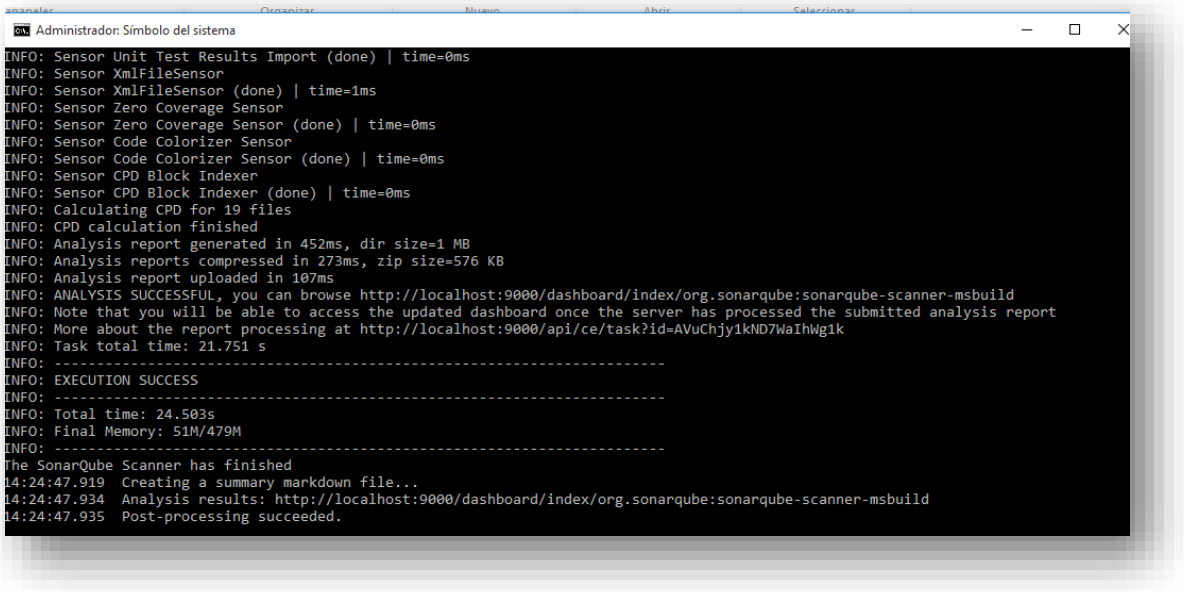
v: Versión sobre la que deseamos ejecutar

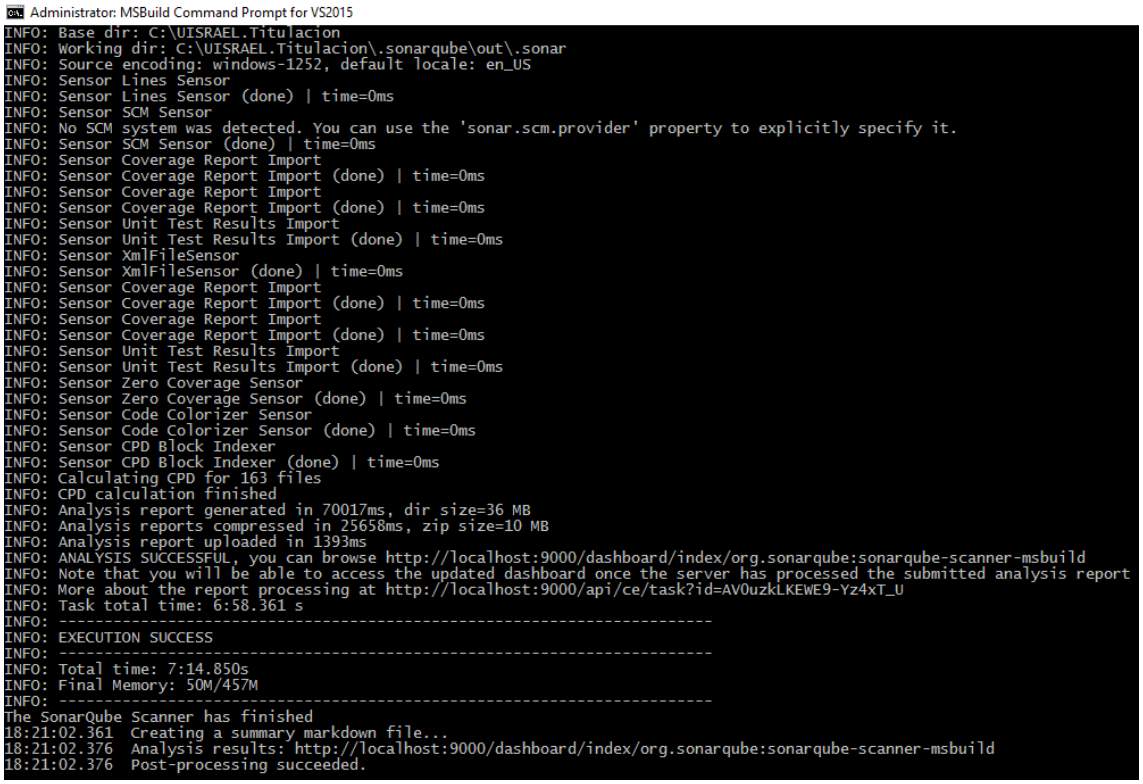


MSBuild.exe /t:Rebuild

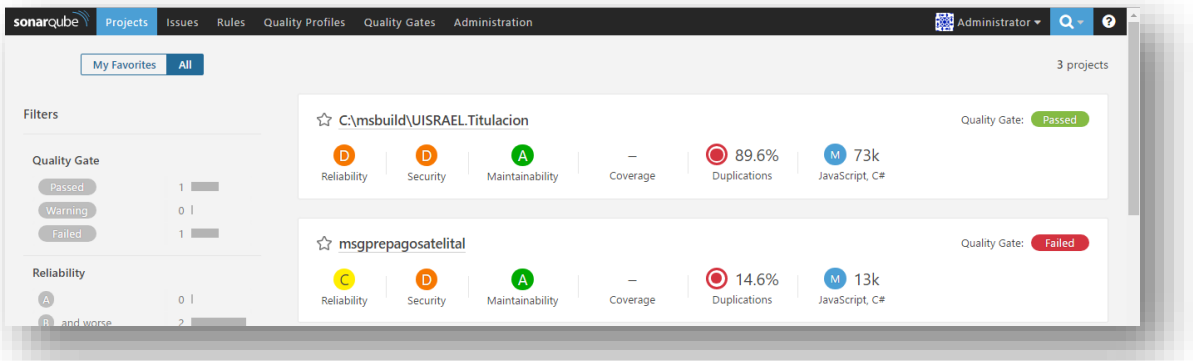


SonarQube.Scanner.MSBuild.exe end



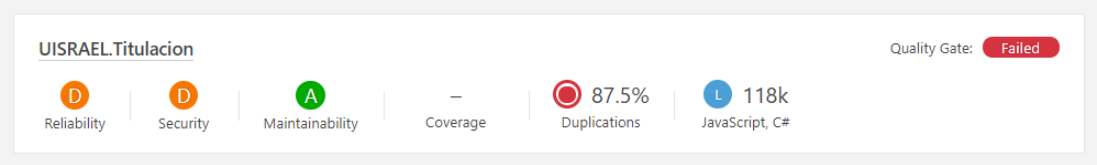


Al final tendremos un reporte en nuestro localhost:9000 o en el sitio en el que se haya publicado el servicio



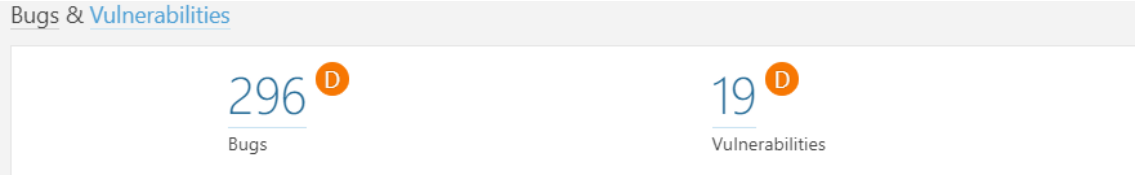
4.5 DESCRIPCIÓN DEL DASHBOARD

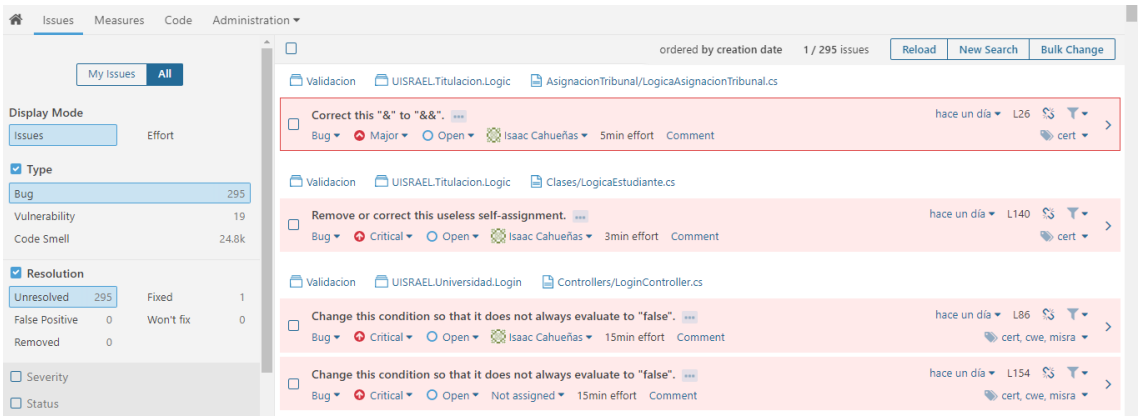
En la descripción general tenemos detalles de la compilación del proyecto**.**



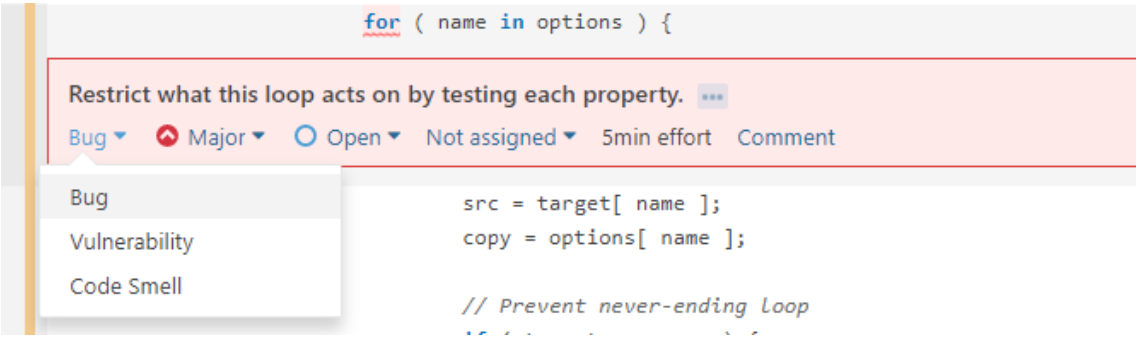
Quality Gate: Es la calificación

Bugs: Alertas referentes al código que no está de acuerdo con las reglas configuradas en el software de análisis de código

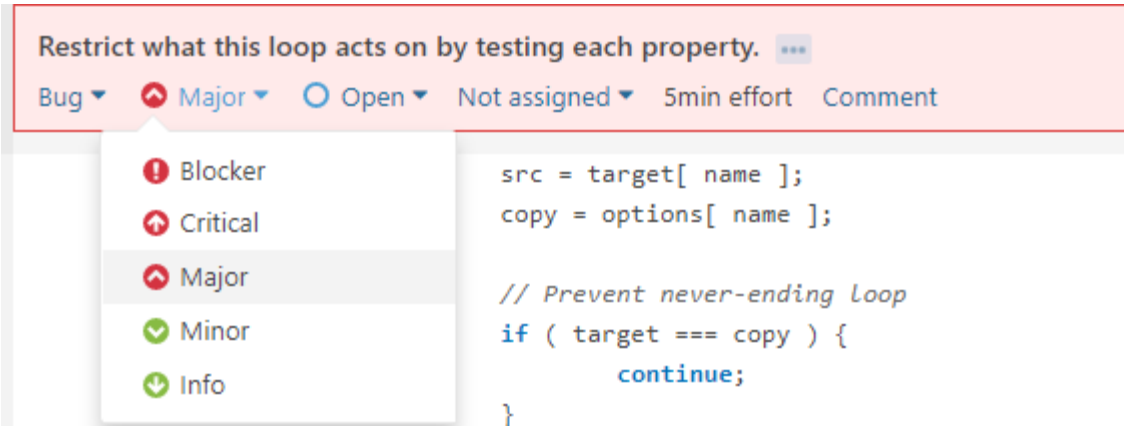




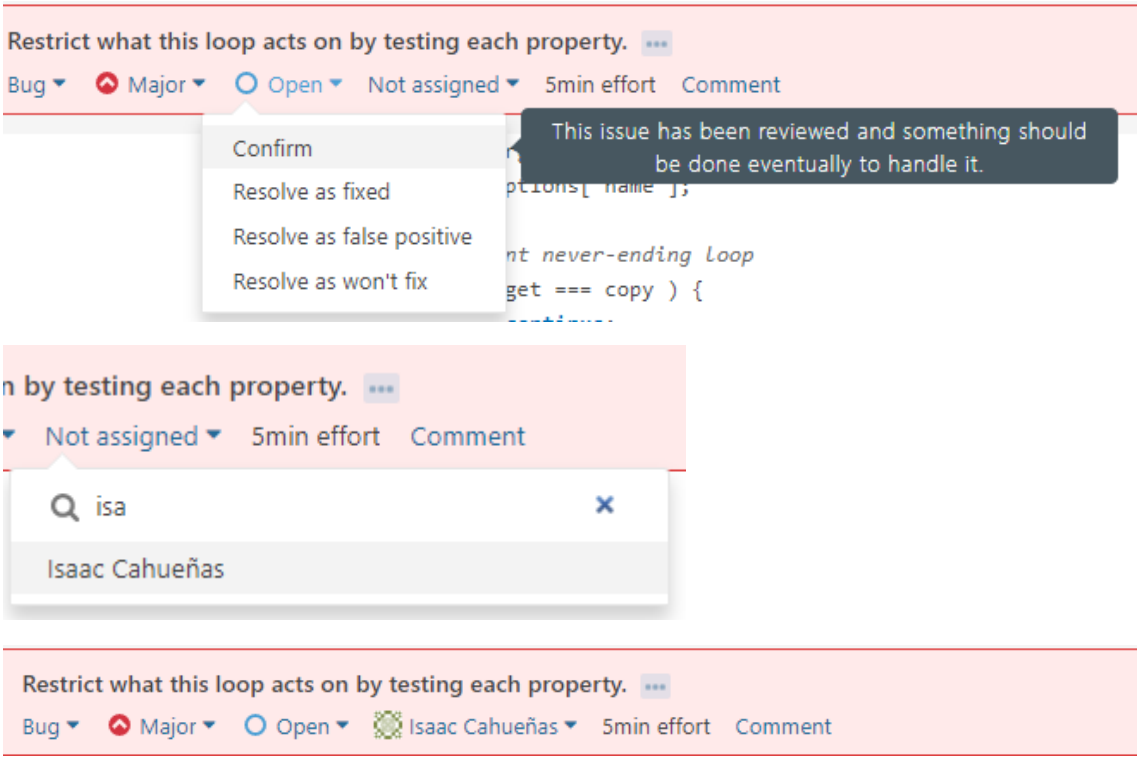
Categorización del bug: Al presionar doble clic sobre el bug podemos ingresar a la sección de código en la que se encuentra al revisar el contexto podremos proceder a categorizar



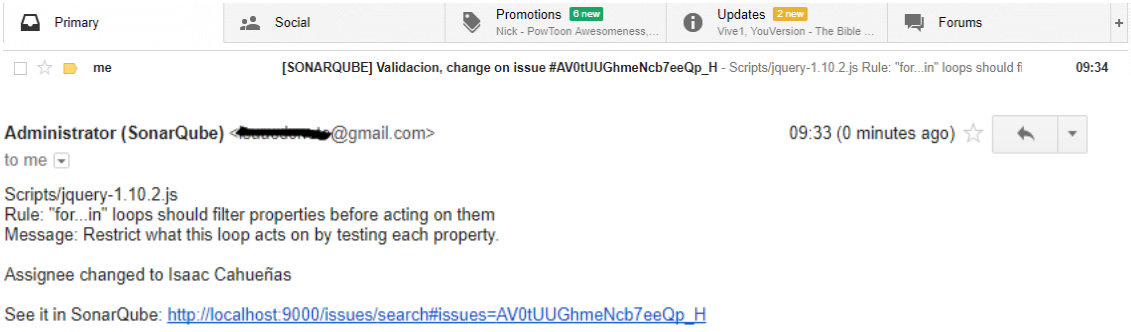
Prioridad del bug: Depende de la importancia que dará el administrador del proyecto



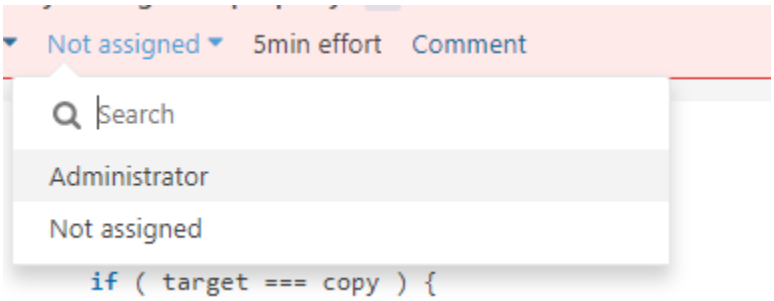
Estado del bug: Nos permite conocer el estado actual del bug



Al asignar o cambiar de estado el bug automáticamente se enviará una notificación para alertar de los cambios

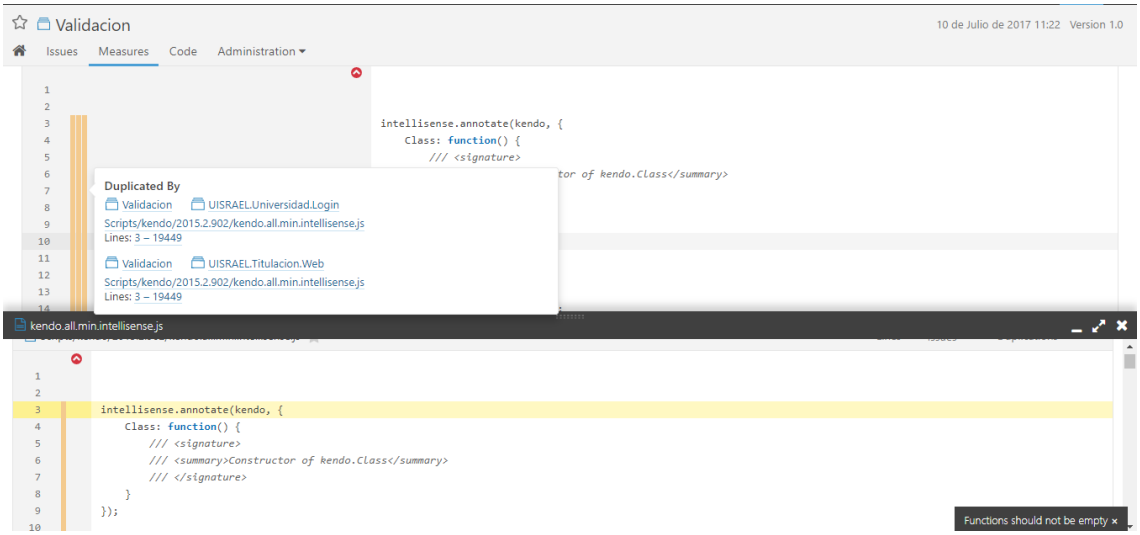


Asignación del bug: Permite indicar quién estará a cargo de cada uno de los bugs, para poder asignar previamente tiene que estar creado como usuario



Duplications: Detecta el porcentaje de código duplicado existente.

Localiza y permite comparar el código



Code Smells: Nos da a conocer el código que no se está utilizando por ejemplo variables declaradas pero no utilizadas