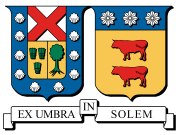
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO, CHILE



MODELO DE CALIDAD PARA EVALUAR CONTINUIDAD DE DESARROLLO

Tesis presentada como requerimiento parcial
para optar al título profesional de
INGENIERO CIVIL EN INFORMÁTICA
por

Pablo Alejandro Acuña Rozas

Comisión Evaluadora:
Dr. Hernán Astudillo
Dr. Raúl Monge

DICIEMBRE 2013

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO, CHILE

TÍTULO DE LA TESIS:
MODELO DE CALIDAD PARA EVALUAR CONTINUIDAD DE DESARROLLO

AUTOR:
PABLO ALEJANDRO ACUÑA ROZAS

Tesis presentada como requerimiento parcial para optar al título profesional de **Ingeniero Civil en Informática** de la Universidad Técnica Federico Santa María.

Profesor Guía
Dr. Hernán Astudillo

Profesor Correferente
Dr. Raúl Monge

Diciembre 2013.
Valparaíso, Chile.

Índice general

Index of Figures v

1. Contexto **1**

1.1. Calidad de Software 1

1.2. Mantenibilidad 2

2. Problema **3**

3. Propuesta **4**

4. Implantación **5**

4.1. Requerimientos de la evaluación 6

4.1.1. Propósito de la evaluación 6

4.1.2. Requerimientos de calidad del producto de software 6

4.1.3. Partes del producto sometidas a evaluación 6

4.1.4. Rigor de la evaluación 7

4.2. Especificación de la evaluación para SAEFramework Servidor 7

4.2.1. Selección de métricas para mantenibilidad 7

4.2.2. Selección de métricas para portabilidad 10

4.3. Especificación de la evaluación para SAE Framework Cliente 11

4.3.1. Selección de métricas para mantenibilidad 11

4.4. Diseño de la evaluación 12

4.4.1. Para SAEFramework Servidor 12

4.4.2. Para SAEFramework Cliente 12

4.4.3. Plan de actividades de evaluación 14

4.5. Ejecución de la evaluación 14
4.6. Conclusiones de la evaluación 14

Bibliography **15**

Índice de figuras

Capítulo 1

Contexto

1.1. Calidad de Software

Calidad de Software es un tema tremendamente importante en el proceso de desarrollo. Hoy en día sabemos que un producto de software que no cuenta con los estándares de calidad adecuados, no goza de propiedades como mantenibilidad, seguridad, usabilidad, etc. Si bien la calidad en general es un concepto subjetivo, se han desarrollado un gran número de intentos por definir una base común con la cual se pueda evaluar un producto de software o que sirva para definir un desarrollo apropiado con el fín de terminar con un producto confiable. Estos intentos generalmente apuntan a definir un modelo de calidad que agrupe las principales características que un producto debe tener.

La ISO, IEC y IEEE definen calidad a través de 6 distintas alternativas [2]:

1. El grado en el cual un sistema, componente o proceso cumple con los requerimientos especificados.

2. La capidad de un producto, servicio, sistema o proceso para cumplir con las necesidades, expectativas o requerimientos del usuario o cliente.

3. El conjunto de características de una entidad que le confieren su habilidad de satisfacer los requerimientos declarados y además los implícitos.

4. Conformidad en las expectativas del usuario, conformidad en los requerimientos del usuario, satisfacción del cliente, confiabilidad y el nivel presente de defectos.

5. El grado en el cual un conjunto inherente de características cumple con los requerimientos.

6. El grado en el cual un sistema, componente o proceso cumple con las necesidades o expectativas de un ciente o usuario.

Como se puede observar en las definiciones, calidad no tiene una definición universal e incluso se podría argumentar que es un tema de carácter filosófico [1].

1.2. Mantenibilidad

La mantenibilidad también es un tema recurrente en ingeniería de software. Al igual que calidad, se compone de elementos subjetivos y que están sujetos al contexto del problema. Para medir la mantenibilidad de un producto de software, se suelen utilizar métricas cuantitativas y cualitativas. Estas métricas usualmente indican que tan mantenible es el producto. Esto puede implicar diversos hechos, por ejemplo, que tan fácil es modificar el código fuente del software, que tan seguro es la modificación del código fuente sin dañar el funcionamiento correcto del sistema, que tan fácil es entender el código fuente del software con el fin de modificarlo o extenderlo, etc. Esta característica de calidad afecta generalmente a los desarrolladores, puesto que el usuario final no es afectado directamente por la buena calidad del código fuente, mientras el software cumpla con los requerimientos de usuario. Sin embargo es frecuente escuchar sobre proyectos que han debido ser paralizados producto de falta de calidad en su código, lo cual no permite seguir manteniéndolo.

[1] `http://en.wikipedia.org/wiki/Quality_(philosophy)`

Capítulo 2

Problema

Uno de los principales problemas que reside en un producto de software es la mantenibilidad. Esta característica se presenta en la mayoría (si es que no en todos) los modelos de calidad originados en la literatura. La mantenibilidad afecta principalemente a los desarrolladores y mantenedores del código del producto de software. Un punto crítico se produce en el momento en que el mantenimiento y extensión del producto es tomado por terceros. Si este producto no es mantenible, puede generar grandes problemas para aquellos terceros que deban continuar con el desarrollo.

Capítulo 3

Propuesta

En este trabajo se propone un modelo de calidad basado en el modelo ISO/IEC 25010 enfocado en la mantenibilidad del producto de software. Esta característica está directamente relacionada con la continuidad de desarrollo. El modelo ISO/IEC 25010 contiene la mantenibilidad como una de sus principales características y además entrega un conjunto de sub-características que sirven para guiar la búsqueda de las métricas apropiadas. Estas métricas no están contenidas en el modelo y deben ser escogidas por los evaluadores, encontrándose muchas de ellas en la literatura. En este trabajo se presentan las sub-características escogidas así como las métricas elegidas para realizar las mediciones sobre un producto de software real, con el fín de evaluar su mantenibilidad y así, entregar información acerca como afectaría su calidad en la continuidad de desarrollo por parte de terceros.

Capítulo 4

Implantación

Para evaluar un producto de software utilizando el modelo ISO/IEC 25010, se debe escoger un conjunto de características de las principales que lo componen. Estas características deben ser escogidas de acuerdo al motivo de la evaluación y deben ser acordes al contexto del producto.

Como se mencionó previamente, la continuidad de desarrollo de un producto de software está directamente ligada con la mantenibilidad de este producto, puesto que la calidad de su código fuente va a incidir en el trabajo posterior de los desarrolladores, más aún si estos no tienen conocimiento previo del sistema (terceros).

Para realizar esta elección, la ISO/IEC provée guías con una serie de consejos para definir correctamente un plan de calidad, partiendo por la elección de características y finalizando con la análisis de la evaluación propiamente tal. Estas guías se pueden encontrar el división ISO/IEC 25040 [1], la cua corresponde a la división de evaluación de calidad. En esta división provée requerimientos, recomendaciones y guías para la evaluación de un producto de software ya sean realizadas por evaluadores independientes, adquirientes o desarrolladores. También se presenta un apoyo para documentar una medición como un módulo de evaluación.

Los principales hitos dentro de este proceso se pueden resumir en:

1. Establecer los requerimientos de la evaluación
2. Especificar la evaluación
3. Diseñar la evaluación

4. Ejecutar la evaluación

5. Concluir la evaluación

La ejecución de estas tareas se describe a continuación.

4.1. Requerimientos de la evaluación

4.1.1. Propósito de la evaluación

Se desea evaluar el producto de software SAE de la empresa MOSAQ con el fín de elaborar un plan de calidad que permita cerrar brechas en torno a temas de mantenibilidad de su producto. A través de esta evaluación se acreditará que la empresa utilizó un conjunto de buenas prácticas para construir el software y que éste puede ser mantenido y modificado por terceros en el caso que fuese necesario.

4.1.2. Requerimientos de calidad del producto de software

En conjunto con MOSAQ, se analizaron las características y subcaracterísticas que ofrece el modelo ISO 25010 con el fin de encontrar las que mejor se adecúen a los requerimientos de calidad. El principal requerimiento es mantenibilidad y además se optó por evaluar algunos aspectos de la portabilidad del producto. Para estas dos características, se escogieron subcaracterísticas del modelo ISO 25010 y ciertas métricas para implementar la evaluación.

4.1.3. Partes del producto sometidas a evaluación

Se sometieron a evaluación dos módulos del producto de software. Estos son SAE Framework Servidor y SAE Framework Cliente. La evaluación difiere en algunos aspectos para cada módulo puesto que tienen diferencias en la arquitectura de la implementación.

4.1.4. Rigor de la evaluación

Se decidió que la mayor cantidad de esfuerzo y rigor debe estar enfocado en estudiar y analizar la mantenibilidad del producto.

Se estudiaron a fondo métricas relacionadas con la mantenibilidad del producto y se utilizaron criterios que permiten asegurar prácticas profesionales en la construcción del producto de software.

También se realizaron descripciones cualitativas acerca de la portabilidad del producto. Esta característica junto con la manteniblidad, es importante para los clientes de MOSAQ ya que ambas influyen en los procesos que se deben llevar a cabo en el caso de que se tuviese que trabajar con terceros en un futuro.

4.2. Especificación de la evaluación para SAEFramework Servidor

A continuación se presentan las métricas definidas para realizar la evaluación. Estas métricas han sido seleccionadas tomando en cuenta las principales recomendaciones que la ISO entrega en la serie 25000, las cuales sirven para implementar un modelo y plan de evaluación de calidad utilizando el modelo presentado en la división 25010.

4.2.1. Selección de métricas para mantenibilidad

Las subcaracterísticas elegidas para mantenibilidad son **modularidad**, **reusabilidad**, **modificabilidad** y **capacidad de pruebas**.

A continuación se presentan las métricas para evaluar estas subcaracterísticas.

Cohesión Relacional (Modularidad)

Es el número promedio de relaciones internas por tipo. Se mide utilizando:

$$H = \frac{R + 1}{N}$$

Donde R es el número de relaciones internas entre tipos y el paquete, N el número de tipos en el paquete.

Las clases dentro de un *assembly*[1] deben estar fuertemente relacionadas, de esta manera la cohesión tendrá tener un valor alto. Por otro lado, valores demasiado altos podrían indicar sobre-acoplamiento. Un buen rango es $1{,}5 \leq H \leq 4{,}0$.

LCOM (Falta de cohesión en métodos) (Modularidad)

El principio de responsabilidad única consiste en que una clase no debe tener más de una razón para cambiar. Una clase con esta característica es cohesiva.

$$LCOM = 1 - \frac{\sum_{f \in F} |M_f|}{|M| \times |F|}$$

Donde M son los métodos estáticos e instancias en la clase, F campos instanciados en la clase y M_f los métodos que acceden el campo f_i.

En una clase que es completamente cohesionada, cada método debe acceder a cada campo instanciado:

$$\sum_{f} |M_f| = |M| \times |F|$$

de manera que el $LCOM = 0$.

Un valor alto de $LCOM$ generalmente quiere decir que una clase tiene una baja cohesión. Tipos en los cuales $LCOM \geq 0{,}8$ y $|F| \geq 10$ y $|M| \geq 10$ podrían ser problemáticos. Sin embargo, es muy difícil evitar estos casos con poca cohesión.

LCOM HS (Falta de cohesión de métodos Henderson-Sellers) (Modularidad)

Esta métrica es similar a la anterior, pero toma su valor en un rango $[0 - 2]$. Un valor LCOM HS mayor a 1 debería ser considerado peligroso.

$$LCOMHS = M - \frac{\sum_{f \in F} |M_f|}{F} \times (M - 1)$$

Tipos en los cuales $LCOMHS \geq 1{,}0$ y $|F| \geq 10$ y $|M| \geq 10$ deberían ser evitados. Esta restricción es más fuerte (por lo tanto más fácil de satisfacer) que la descrita para $LCOM$.

[1] biblioteca de código compilado

Acoplamiento eferente (Modularidad)

Número de tipos en el paquete correspondiente, que dependen de tipos que están fuera del paquete.

Un valor muy alto de esta métrica podría implicar problemas de diseño. Tipos que tengan este valor muy alto están entrelazados con muchas otras implementaciones. Mientras más alto sea el valor, mayor es el número de responsabilidades que el tipo tiene.

Acoplamiento aferente (Modularidad)

Número de tipos fuera del paquete, que dependen de tipos que están en el paquete en evaluación.

Un valor alto de esta métrica no es necesariamente peligroso, sin embargo es interesante saber que partes del código son altamente utilizadas.

Esta métrica es útil especialmente cuando es igual a 0, lo cual podría indicar un elemento de código sin uso. Estos casos deben ser manejados con cuidado para puntos de entrada, constructores de clases o finalizadores ya que estos métodos siempre tendrán un valor 0 para acoplamiento aferente y no corresponden a código sin uso.

Instabilidad (Modificabilidad)

Es la razón entre el acoplamiento eferente y el acoplamiento total. Esta métrica indica la resiliencia al cambio del paquete.

$$I = C_e / (C_e + C_a)$$

Donde C_e es el acoplamiento eferente y C_a el acoplamiento aferente.

Un valor de $I = 0$ indica un paquete completamente estable, fácil de modificar. Un valor de $I = 1$ indica un paquete completamente inestable.

Complejidad Ciclomática (Modificabilidad)

Número de decisiones que pueden ser tomadas en un procedimiento. Procedimientos con un valor mayor a 15 son difíciles de entender, mientras que con un valor mayor a 30 son extremadamente complejos y deberían ser divididos en métodos más pequeños (a menos que sea código auto-generado).

índice de mantenibilidad (Modificabilidad)

Corresponde a un índice entre 0 y 100 que representa la facilitad relativa de mantener el código. Un valor más alto indica una mejor mantenibilidad. Un valor entre 20 y 100 indica que el código tiene una buena mantenibilidad. Un valor entre 10 y 19 indica que el código es moderadamente mantenible y un código entre 0 y 9 indica una baja mantenibilidad.

Código duplicado (Reusabilidad)

Se utiliza alguna heurística para detectar código potencialmente duplicado. El hecho de encontrar un porcentaje alto de duplicación, podría indicar que no se está haciendo un reuso adecuado en el software.

4.2.2. Selección de métricas para portabilidad

Prácticas de instalación (Instalabilidad)

Descripciones cualitativas acerca de cómo se implementa una instalación estándar para el producto. Se estudian estos procesos y se recomiendan mejoras para alinear las prácticas a estándares profesionales.

4.3. Especificación de la evaluación para SAE Framework Cliente

4.3.1. Selección de métricas para mantenibilidad

Complejidad Ciclomática (Modificabilidad)

Número de decisiones que pueden ser tomadas en un procedimiento. Procedimientos con un valor mayor a 15 son difíciles de entender, mientras que con un valor mayor a 30 son extremadamente complejos y deberían ser divididos en métodos más pequeños (a menos que sea código auto-generado).

Profundidad (Modificabilidad, Analizabilidad)

Estudia el nivel de anidamiento que puede existir entre funciones o expresiones dentro del código. Se debe definir un límite crítico de profundidad y verificar que no se esté sobrepasando.

Número de parámetros (Analizabilidad)

Estudia el número de parámetros en una función. Al reducir este valor, se puede mejorar la analizabilidad y modularidad del código de manera sustancial. Al igual que la métrica anterior, se debe definir un valor límite para estudiar el código y verificar que no se esté sobrepasando.

Código duplicado (Reusabilidad)

Se utiliza alguna heurística para detectar código potencialmente duplicado. El hecho de encontrar un porcentaje alto de duplicación, podría indicar que no se está haciendo un reuso adecuado en el software.

4.4. Diseño de la evaluación

Para llevar a cabo las mediciones, MOSAQ hizo entrega de las fuentes principales de su software, los cuales bajo un acuerdo de confidencialidad, fueron analizados por parte del equipo Toeska.

4.4.1. Para SAEFramework Servidor

Para este módulo se utilizaron las siguientes herramientas para realizar las mediciones.

- NDepend v5.0.0.8085[2]
- Visual Studio 2013 Code Metrics[3]
- Visual Studio 2012 Code Clone Analysis[4]

4.4.2. Para SAEFramework Cliente

- SonarQube[5]
- JsHint v2.1.11[6]
- WebStorm v7 Inspection tools[7]

  La configuración elegida para JsHint es la siguiente:

```
{
    "globals": {
        "console": false,
        "jQuery": false,
        "_": false
```

[2]http://www.ndepend.com
[3]http://msdn.microsoft.com/en-us/library/bb385914.aspx
[4]http://msdn.microsoft.com/en-us/library/hh205279.aspx
[5]http://www.sonarqube.org/
[6]http://www.jshint.com/
[7]http://www.jetbrains.com/webstorm/

```
    },
    "maxparams": 5,
    "maxdepth": 5,
    "maxstatements": 25,
    "maxcomplexity": 10,
    "es5": true,
    "browser": true,
    "boss": false,
    "curly": false,
    "debug": false,
    "devel": false,
    "eqeqeq": true,
    "evil": true,
    "forin": false,
    "immed": true,
    "laxbreak": false,
    "newcap": true,
    "noarg": true,
    "noempty": false,
    "nonew": false,
    "nomen": false,
    "onevar": true,
    "plusplus": false,
    "regexp": false,
    "undef": true,
    "sub": true,
    "strict": false,
    "white": true,
    "unused": true
}
```

En esta configuración podemos ver los parametros *maxparams*, *maxdepth* y *maxcomplexity* los cuales nos permiten definir los límites para el número de parámetros, profundidad y complejidad ciclomática respectivamente. Estos parámetros fueron seleccionados de manera informada de acuerdo a buenas prácticas investigadas previamente y acordes a un software con las características de SAE Framework.

4.4.3. Plan de actividades de evaluación

A través de reuniones con MOSAQ, se acordaron los siguientes puntos dentro del plan de trabajo en lo que respecta a planificación y evaluación de calidad.

- El equipo Toeska realizará las mediciones pertinentes sobre el código fuente y sobre algunas prácticas de desarrollo del equipo de MOSAQ. Con estas mediciones se evaluarán los requerimientos de calidad establecidos en el modelo que se definió.
- El equipo Toeska entregará los resultados de la primera evaluación, los puntos donde se encontraron brechas que deben ser consideradas, además guías para realizar los ajustes necesarios para que el producto obtenga mejores resultados durante la próxima evaluación.
- Finalmente se realizará una evaluación final utilizando los mismo criterios utilizados para la primera evaluación, en la cual se espera haber cerrado las brechas encontradas anteriormente y así certificar que el producto cumple con las características escogidas de la norma ISO 25010.

4.5. Ejecución de la evaluación

4.6. Conclusiones de la evaluación

Bibliografía

[1] Systems and software engineering – systems and software quality requirements and evaluation (square) – evaluation process. *ISO/IEC 25040:2011*.

[2] Iso/iec/ieee systems and software engineering - vocabulary. *IEEE Unapproved Draft Std P24765-2009, Sept 2009*, pages–, 2009.