Impacto de las métricas CK en la refactorización

José del Sagrado[1], Isabel María del Águila[1],
Alfonso Bosch[1], and Francisco Chicano[2]

[1] Departmento de Informática,
Universidad de Almería,
04120 Almería, España
jsagrado@ual.es, imaguila@ual.es, abosch@ual.es,
[2] Departamento de Lenguajes y CC. de la Computación
Universidad de Málaga,
29071 Málaga, España
chicano@lcc.uma.es

Resumen Las métricas CK a nivel de diseño orientado a objetos, son las que alcanzan un mayor consenso sobre la identificación de la necesidad de una refactorización. Para estimar el impacto de estas métricas de calidad en la refactorización en este trabajo nos basamos en la reducción de la entropía. Para ellos se parte de los datos validados de refactorizaciones y de métricas de código de varios proyectos open source. Las valoraciones obtenidas se combinan para ordenar las métricas y proponemos un método para medir su influencia incluso en aquellas situaciones en las que no todas las métricas puedan ser valoradas o cuando esta valoración no alcance unas tasas suficientemente representativas. Los resultados obtenidos están en la misma línea de trabajos previos de otros autores, siendo de mayor influencia las medidas de complejidad y acoplamiento y de menor aquellas métricas relativas a la herencia y la cohesión.

Keywords: refactorización, métricas de código, evolución del software

1. Introducción

La refactorización busca mejorar la calidad de la estructura de un producto software que tiende a degradarse conforme evoluciona el sistema [12]. En concreto, la refactorización de código es la técnica de modificación del código fuente sin cambiar su comportamiento [7]. Esta técnica va ligada al paradigma de desarrollo dirigido por la prueba y en la industria del software los equipos alternan iteraciones dedicadas a la construcción de nuevas funcionalidades basadas en casos de prueba, con iteraciones dedicadas a la refactorización del código para mejorar su consistencia interna, legibilidad y extensibilidad futura.

A pesar de sus ventajas, refactorizar no es una tarea sencilla puesto que, en primer lugar, hay que identificar la necesidad de una refactorización (habitualmente ligada a los "bad smells" o erosiones de código) y, después, es necesario aplicarla de forma apropiada. Si bien es cierto que, a criterio de Fowler, las

métricas no son rival frente a la intuición de los desarrolladores a la hora de la refactorización [7], está claro que los buenos desarrolladores necesitan medir las características de los productos software que construyen y que el hecho de medir distintos aspectos de la calidad del software permite a los equipos de desarrollo tomar decisiones correctas, estimar costes, evaluar riesgos y fundamentar sus intuiciones [6].

En este trabajo vamos a realizar un estudio del impacto que las métricas CK [5], las más ampliamente unificadas a nivel de diseño orientado a objetos, tienen sobre la identificación de la necesidad de una refactorización. Para la estimación del impacto de la refactorización y las métricas de calidad se pueden utilizar distintas técnicas. Algunos ejemplos, aunque no los únicos, son la regresión logística [3], análisis de hipótesis [2] o experimentos controlados [4]. En este estudio empleamos la reducción en la *entropía* para medir este impacto sobre un conjunto de datos consistente en refactorizaciones y métricas de código de 37 versiones de 7 proyectos Java open source que han sido validadas manualmente para eliminar los falsos positivos [10].

El trabajo se estructura en seis secciones. En la sección 2 se esbozan distintas líneas y trabajos que han abordado el estudio de la relación entre factores de calidad y refactorización. Las medidas y conjuntos de datos utilizados se describen en la sección 3. La toma de decisiones en este ámbito se aborda desde el punto de vista de la reducción en la entropía y los distintos criterios de valoración empleados se definen en la sección 4. La sección 5 explora la influencia de las métricas sobre la decisión de refactorizar. Por último, la sección 6 recoge las conclusiones y trabajos futuros.

2. Relación entre factores de calidad y refactorización

Entre los desarrolladores de software ha sido ampliamente aceptada la necesidad de utilizar mecanismos de medida cuantitativos para abordar cualquier proceso de mejora o predicción. Las métricas son una potente herramienta para gestionar los proyectos de desarrollo de software. Desafortunadamente no existe un conjunto de métricas universalmente válidas para cualquier tarea de ingeniería del software puesto que cada una se especializa en un aspecto concreto. De entre ellas cabe destacar las *'suites'* de métricas de diseño orientado a objetos por llevar más de 25 años implantadas y porque actualmente son fáciles de obtener mediante herramientas de análisis de código como SonarQube[3] o SourceMeter[4]. Este hecho unido a la disponibilidad de numerosos repositorios de código abierto las convierten en una buena oportunidad de investigación cuantitativa en el campo de la ingeniería del software.

Algunos trabajos de investigación utilizan las métricas de código para estimar la calidad en términos de la propensión a fallos y defectos [17,13]. En el ámbito de la evolución del software, la mantenibilidad y la refactorización también se han realizado numerosos trabajos que plantean estudios empíricos para justificar y

[3] https://www.sonarqube.org/
[4] https://www.sourcemeter.com/

analizar la utilidad de las métricas en la aplicación práctica de la refactorización. En los primeros trabajos se buscaba acercar las investigaciones de métricas de software y programación orientada a objetos mediante el análisis de los cambios entre versiones a nivel de las clases [11]. Sin embargo, las medidas ligadas al cambio necesitan ser complementadas puesto que, en proyectos reales, en torno al 70 % de las clases no presentan cambios y el resto pueden ser tanto mejoras en la estructuración (refactorización), como la inclusion de nuevas funcionalidades [14]. Se deben manejar enfoques más complejos donde se evalúe el impacto de la refactorización y de los factores de calidad externos (como la reutilización, flexibilidad, extensibilidad y efectividad) que se calculan a partir de las métricas (factores de calidad internos) [15]. Habitualmente los factores de calidad son contradictorios y cualquier acción de mejora no mejorará todas a la vez.

La decisión de refactorizar se centra en los desarrolladores y, además de los valores puramente cuantitativos, su punto de vista también debería tenerse en cuenta. En diversos trabajos se plantea como objetivo dar soporte a la decisión de qué y cuándo refactorizar con técnicas objetivas, bien sea dando una representación visual a la distancia de la medida de cohesión entre componentes del software [16] o bien aplicando el concepto de Pareto optimalidad a la optimización de la secuencia de operaciones a aplicar en una clase, siendo los objetivos contradictorios el acoplamiento y la desviación estándar del número de métodos de la clase [8]; eso si, todo para un reducido grupo de refactorizaciones.

Por otra parte, aunque existen herramientas que detectan si se han producido refactorizaciones, estas suelen generar numerosos falsos positivos llegando a precisiones del 79 % lo que supone una sobreestimación [3,10]. A pesar del esfuerzo empleado en la construcción de herramientas automáticas o semiautomáticas para la identificación de anomalías que requieran refactorización, son varios los estudios sobre el análisis de los cambios en las métricas definidas por las características del código que concluyen que no existe una justificación clara para relacionar las erosiones de código, las métricas y la refactorización efectiva. Para Hegedus et al. no existe una correlación estadísticamente significativa entre las métricas y las opiniones de los expertos sobre mantenibilidad [9]. El enfoque basado en métricas es efectivo para identificar el cambio, pero se vuelve ineficaz como base para determinar el tipo de cambio, por refactorización o por nueva funcionalidad [14]. La utilización de regresión para relacionar la refactorización con las métricas de código y la presencia de erosiones pone de relieve que, con muy pocas excepciones, las métricas de calidad no muestran una relación clara con la refactorización [3].

Todo esto confirma la oportunidad de plantear el estudio del impacto de las métricas CK en la decisión de refactorización desde otro punto de vista. Así, siguiendo las pautas de diseño de experimentos [1], en este trabajo recurrimos al contenido de información cómo indicio de la importancia de las métricas en la refactorización.

3. Datos del estudio

Los datos de partida son los publicados en el trabajo de Kádár et al. [10], donde se realiza la validación manual de las refactorizaciones identificadas automáticamente entre dos versiones consecutivas de siete proyectos open source en Java disponibles en Github (ver tabla 1). En este trabajo, además de la eliminación de falsos positivos, el conjunto de datos publicado incorpora las métricas CK a nivel de clase que han sido calculadas de manera automática. A continuación se describen brevemente todas las medidas utilizadas y en la tabla 1 se incluyen los valores máximos alcanzados en cada proyecto. Una descripción extensa del significado de cada una de ellas lo podemos encontrar en [5,3].

- **LCOM** mide la **falta de cohesión** en una clase. Se calcula utilizando un grafo no dirigido donde los nodos son los métodos locales, sin considerar constructores, destructores ni métodos get y set. Dos nodos están conectados si y sólo si usan un atributo común (local o heredado), se usa un método abstracto o un método invoca al otro. El resultado es el número de componentes conectadas en el grafo.
- **WMC** mide la **complejidad** de una clase en base a la complejidad de sus métodos. Se calcula como la complejidad ciclomática de McCabe para todos los métodos locales y los bloques iniciales. Expresa el número de caminos de control independientes.
- El **acoplamiento** entre clases se mide con **CBO** como el número de clases a las que una clase dada está ligada directamente excluyendo la herencia y con **RFC** que es el conjunto repuesta de una clase, es decir el cardinal de todos los métodos que puede ser ejecutados en respuesta a un mensaje a un objeto de la clase.
- Las métricas de **herencia** son **DIT**, que mide la profundidad del árbol de herencia y **NOC** que mide el número de subclases subordinadas a una clase dada.

Tabla 1. Características de los proyectos. Máximos valores de las métricas

Proyecto	#clases	#ref	LCOM	WMC	CBO	RFC	DIT	NOC
antlr4	437	23	78	216	53	117	6	33
junit	657	9	25	28	54	5	7	2
MapDB	439	4	22	1525	40	156	4	11
mct	2162	15	60	413	56	246	5	65
mcMMO	301	4	138	261	48	200	2	13
oryx	536	15	10	121	38	104	5	64
titan	1486	13	60	323	124	299	9	36

4. Criterios de Valoración

Las distintas métricas que se calculan sobre el código pueden valorarse respecto a la decisión de la refactorización de una clase, basándose en la reducción en la *entropía* de esta decisión que se consigue al conocer el valor de una métrica concreta. Este concepto puede utilizarse para definir una secuencia de métricas que tenga por objetivo acotar la decisión sobre si refactorizar o no una clase.

De manera intuitiva, la *ganancia de información* mide la información que comparten la decisión de refactorizar y una métrica dada. Es decir, mide cuánto conocer la métrica reduce la incertidumbre sobre el decisión de refactorizar. Así, preferiremos métricas que alcancen una mayor ganancia de información. Formalmente, si D es un conjunto de observaciones de la forma $(m_1, m_2, \ldots, m_n, r)$, donde m_i es el valor de la métrica M_i y r la decisión de refactorizar o no, la *ganancia de información* para la métrica M_i se define como

$$GI(D, M_i) = H(D) - \sum_{m \in valores(M_i)} \frac{|D_m|}{|D|} \cdot H(D_m), \tag{1}$$

donde H es la entropía de la distribución de probabilidad $(p_1, p_2, \cdots, p_n)$ asociada a la decisión de refactorizar en el conjunto de observaciones D considerado

$$H(D) = -\sum_{i=1}^{n} p_i \cdot \log_2(p_i). \tag{2}$$

La forma en la que está definida la ganancia de información hace que las variables que puedan tomar un alto número de valores alcancen mayores puntuaciones. Por eso, alternativamente se emplea la *razón de ganancia de información*. Este criterio intenta evitar el problema anterior aplicando una especie de normalización a la ganancia de información basada en el valor de la *división de información*:

$$DI(D, M_i) = -\sum_{m \in valores(M_i)} \frac{|D_m|}{|D|} \cdot \log_2\left(\frac{|D_m|}{|D|}\right) \tag{3}$$

Este valor representa la información potencial que se genera al dividir el conjunto de observaciones D en función de los valores que toma la métrica M_i. La *razón de ganancia de información* se obtiene al dividir la *ganancia de información* entre la *división de información*:

$$RG(D, M_i) = \frac{GI(D, M_i)}{DI(D, M_i)}. \tag{4}$$

El criterio de preferencia entre métricas se mantiene y ahora se preferirá métricas que alcancen una mayor *razón de información*.

Tabla 2. Ganancia de información de las métricas en cada conjunto de datos

Proyecto	LCOM	WMC	CBO	RFC	DIT	NOC
antlr4	0.0164 (5)	0.0422 (2)	0.0422 (1)	0.0422 (3)	0.0000 (6)	0.0208 (4)
junit	0.0000 (6)	0.0201 (3)	0.0295 (1)	0.0253 (2)	0.0000 (5)	0.0000 (4)
MapDB	0.0000 (6)	0.0000 (3)	0.0000 (4)	0.0000 (5)	0.0000 (2)	0.0000 (1)
mct	0.0100 (4)	0.0111 (3)	0,0184 (1)	0,0169 (2)	0.0000 (5)	0.0000 (6)
mcMMO	0.0000 (6)	0.0000 (3)	0.0000 (4)	0.0000 (5)	0.0000 (2)	0.0000 (1)
oryx	0.0000 (6)	0.0000 (3)	0.0000 (4)	0.0242 (1)	0.0000 (5)	0.0000 (2)
titan	0.01015(3)	0.01294(1)	0,0087 (4)	0.01016(2)	0.0000 (5)	0.0000 (6)

Tabla 3. Razón de Ganancia de información de las métricas en cada conjunto de datos

Proyecto	LCOM	WMC	CBO	RFC	DIT	NOC
antlr4	0.0173 (5)	0.0461 (2)	0.0461 (1)	0.0414 (3)	0.0000 (6)	0.0289 (4)
junit	0.0000 (6)	0.0332 (3)	0.4109 (1)	0.0523 (2)	0.0000 (5)	0.0000 (4)
MapDB	0.0000 (6)	0.0000 (3)	0.0000 (4)	0.0000 (5)	0.0000 (2)	0.0000 (1)
mct	0.0526 (2)	0.0139 (4)	0.1035 (1)	0,0243 (3)	0.0000 (5)	0.0000 (6)
mcMMO	0.0000 (6)	0.0000 (3)	0.0000 (4)	0.0000 (5)	0.0000 (2)	0.0000 (1)
oryx	0.0000 (6)	0.0000 (3)	0.0000 (4)	0.0244 (1)	0.0000 (5)	0.0000 (2)
titan	0.2676 (1)	0.0222 (3)	0.0163 (4)	0.0617 (2)	0.0000 (5)	0.0000 (6)

5. Influencia de las métricas en la refactorización

Una vez definidos los criterios de valoración, pueden utilizarse para hacernos una idea de la influencia que tiene cada una de las métricas sobre la decisión de refactorizar. Para ello, se consideran los datos de cada proyecto de manera independiente y aplicamos los criterios de valoración con el fin de obtener una lista ordenada de métricas en base a su influencia sobre la decisión de refactorizar el código. Las tablas 2 y 3 muestran, respectivamente, los valores de ganancia de información y de razón de ganancia en cada conjunto de datos para cada métrica, junto con la posición relativa de la métrica respecto a las demás. Como era de esperar, no hay un único orden de preferencia de las métricas respecto a su influencia en la refactorización. Para establecer el orden de preferencia de consenso se aplica el método de conteo de Borda, asignando a cada métrica un número de puntos correspondiente al número de métricas situadas por debajo de ella en cada lista ordenada. De esta forma obtenemos un orden de preferencia para cada criterio de valoración. En el caso de la *ganancia de información* el orden es WMC, CBO, RFC, NOC, DIT, LCOM; mientras que para la razón de ganancia es CBO, WMC, RFC, NOC, DIT y LCOM. Cabe destacar la poca diferencia existente entre las tres primeras métricas, la coincidencia en las tres últimas y los bajísimos valores que alcanzan los criterios de valoración en algunas métricas (cómo DIT y NOC) y en los conjuntos MapDB y mcMMO. La poca influencia de DIT ha sido también resultado de otros estudios [3].

A fin de explorar el impacto que tienen los valores tan bajos en el orden de preferencia, se modifica la puntuación asignada a las métricas a la hora de aplicar el conteo de Borda. Si el valor del criterio de valoración es muy pequeño (prácticamente cero) la puntuación será cero. En el resto de casos, la puntuación queda inalterada. Con este esquema de puntuación, tanto para la *ganancia de información*, como para la *razón de ganancia* el orden obtenido coincide y es RFC, CBO, WMC, LCOM, NOC y DIT.

Explorar la influencia de las métricas en la refactorización implica tener en cuenta el valor alcanzado por el criterio de valoración, además del orden que induce. Esto se consigue ponderando la puntuación asociada a la ordenación con el valor obtenido al aplicar el criterio de valoración, lo que da lugar a la medida de consenso de incertidumbre siguiente:

$$C_{M_i} = \sum_{k=1}^{n} \frac{CV(D_k, M_i)}{n} \cdot puntos(D_k, M_i), \tag{5}$$

donde M_i es la métrica cuyo valor de consenso se quiere calcular, se dispone de n conjuntos de datos, $CV(D_k, M_i)$ es el criterio de valoración (GI o RG) que se aplica a M_i con el conjunto de datos D_k y $puntos(D_k, M_i)$ es la puntuación alcanzada por M_i en el conjunto D_k según el esquema de conteo de Borda. Si utilizamos la *ganancia de información* el orden es RFC, CBO, WMC, LCOM, NOC, DIT; mientras que para la razón de ganancia es CBO, LCOM, RFC, WMC, NOC, DIT.

La figura 1 muestra las valoraciones, calculadas por cada uno de los métodos aplicados, para la influencia de las métricas en la refactorización. La mayoría de las listas obtenidas, a partir de los datos y criterios de valoración utilizados, diferencian entre un primer grupo de métricas de complejidad y acoplamiento (RFC, CBO y WMC) y un segundo grupo de métricas de cohesión y herencia (LCOM, NOC y DIT). Este resultado refuerza la idea de que WMC es una de las métricas con influencia más efectiva en la refactorización. Esto sugiere que de cara a decidir aplicar la refactorización a nivel de clase, deberían tenerse en cuenta antes acoplamiento y complejidad, para recurrir después a cohesión y herencia. Cabe destacar que la cohesión estructural representada por LCOM no presenta una gran influencia en contra a los principios clásicos del diseño estructurado.

6. Conclusiones

En este trabajo investigamos la influencia de las métricas CK en la refactorización del código. Para hacer este estudio recurrimos a conjuntos de datos sobre refactorización, para valorar sobre ellos las métricas respecto a la decisión de refactorizar en función de la reducción en la *entropía*. Las valoraciones obtenidas se emplean como punto de partida para obtener una lista ordenada de métricas aplicando distintos métodos, incluso en situaciones en las que no todas las métricas pueden valorarse o su valoración no es representativa. En este último

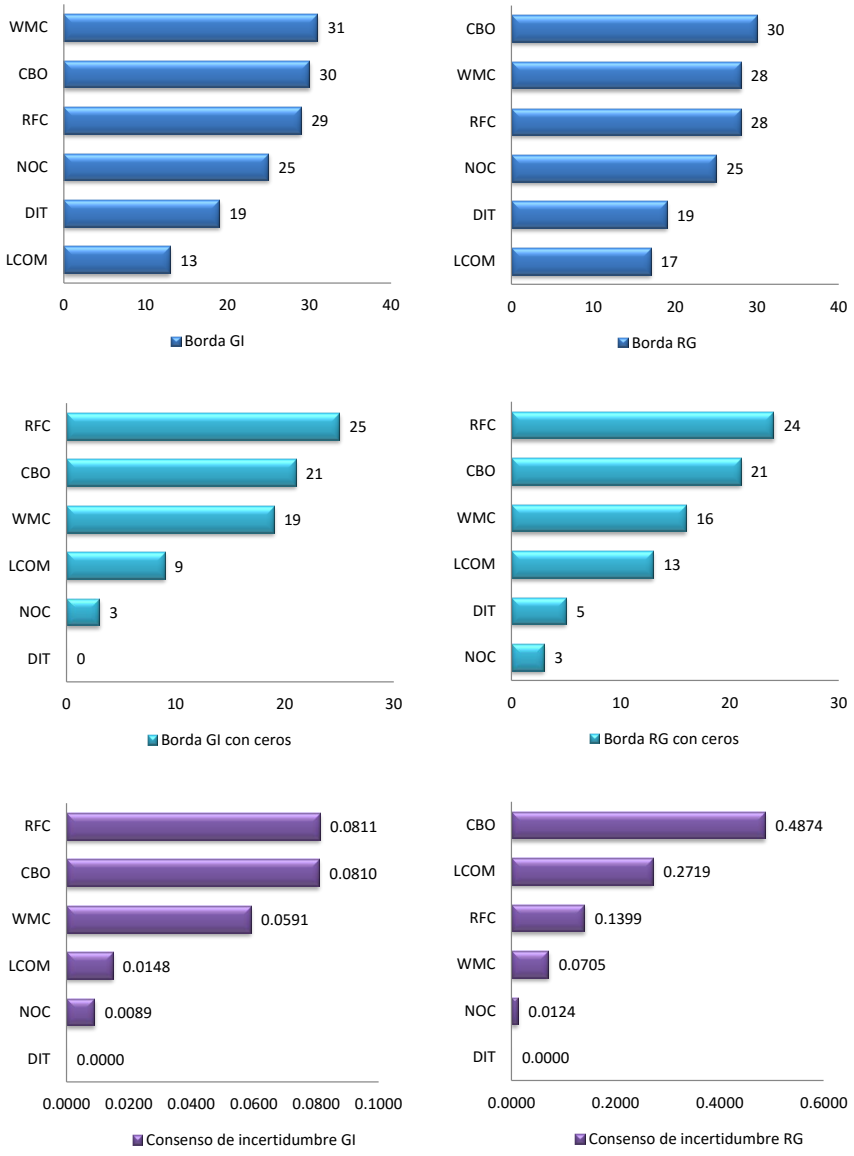

Figura 1. Influencia de las métricas CK sobre la refactorización

caso, proponemos una medida de evaluación que tiene en cuenta la valoración empleada.

Los resultados alcanzados sugieren que de cara a decidir cuándo refactorizar una clase dada deberían tenerse en cuenta el número de clases con las que se relaciona directamente, así como el número de métodos de la clase y la complejidad ciclomática de estos métodos. Las comunicaciones dentro de la propia clase y el nivel que ocupa dentro del árbol de herencia de la aplicación tienen menos influencia en la decisión. Estas conclusiones validan los resultados anteriores pero basándonos en lo informativas que son las métricas de cara a la decisión de refactorizar en distintos conjuntos de datos.

Como trabajos futuros se plantea la necesidad de aplicar esta misma aproximación pero diferenciando entre los distintos tipos de refactorizaciones, puesto que no es lo mismo *extraer un método* de una clase que *consolidar expresiones condicionales*

Agradecimientos Este trabajo ha sido financiado parcialmente por la Universidad de Almería, la Red de Excelencia SEBASENet (TIN2015-71841-REDT) y por el Ministerio de Economía y Competitividad a través del proyecto TIN2013-46638-C3-1-P.

Referencias

1. del Águila, I.M., del Sagrado, J., Bosch, A.: Flujo de trabajo para la experimentación colaborativa en Ingeniería del Software guiada por búsqueda. In: Proceeding of XX Jornadas de Ingeniería del Software y Bases de Datos. JISBD 2016, Sistedes, (2016)
2. Alshayeb, M.: Empirical investigation of refactoring effect on software quality. Information and Software Technology 51(9), 1319–1326 (2009)
3. Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R., Palomba, F.: An experimental investigation on the innate relationship between quality and refactoring. Journal of Systems and Software 107, 1–14 (2015)
4. Chaparro, O., Bavota, G., Marcus, A., Penta, M.D.: On the Impact of Refactoring Operations on Code Quality Metrics. 2014 IEEE International Conference on Software Maintenance and Evolution pp. 456–460 (2014)
5. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Transactions on software engineering 20(6), 476–493 (1994)
6. Fenton, N., Bieman, J.: Software metrics: a rigorous and practical approach. CRC Press (2014)
7. Fowler, M., Beck, K.: Refactoring: improving the design of existing code. Addison-Wesley Professional (1999)
8. Harman, M., Tratt, L.: Pareto Optimal Search Based Refactoring at the Design Level. Proceddings GECCO 2007 pp. 1106–1113 (2007)
9. Hegedus, P., Bakota, T., Illés, L., Ladányi, G., Ferenc, R., Gyimóthy, T.: Source code metrics and maintainability: A case study. Communications in Computer and Information Science 257 CCIS, 272–284 (2011)
10. Kádár, I., Hegedüs, P., Ferenc, R., Gyimóthy, T.: A Manually Validated Code Refactoring Dataset and Its Assessment Regarding Software Maintainability. In:

Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering. pp. 10:1—-10:4. PROMISE 2016, ACM, New York, NY, USA (2016)
11. Li, W., Henry, S.: Object-oriented metrics that predict maintainability. Journal of systems and software 23(2), 111–122 (1993)
12. Mens, T., Tourwé, T.: A survey of software refactoring. IEEE Transactions on Software Engineering 30(2), 126–139 (2004)
13. Olague, H.M., Etzkorn, L.H., Gholston, S., Quattlebaum, S.: Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly Iterative or agile software development processes. IEEE Transactions on Software Engineering 33(6), 402–419 (2007)
14. Schneider, J.G., Vasa, R., Hoon, L.: Do Metrics Help to Identify Refactoring? Workshop on Software Evolution and International Workshop on Principles of Software Evolution pp. 3–7 (2010)
15. Shatnawi, R., Li, W.: An empirical assessment of refactoring impact on software quality using a hierarchical quality model. International Journal of Software Engineering and Its Applications 5(4), 127–149
16. Simon, F., Steinbrückner, F., Lewerentz, C.: Metrics based refactoring. Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR pp. 30–38 (2001)
17. Subramanyam, R., Krishnan, M.S.: Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects. IEEE Transactions on Software Engineering 29(4), 297–310 (2003)