Tampereen teknillinen yliopisto. Ohjelmistotekniikan laitos. Raportti 5
Tampere University of Technology. Department of Software Systems. Report 5

Jari Peltonen (Ed.)

SPLST'09 & NW-MODE'09

Proceedings of 11th Symposium on Programming Languages and Software Tools and 7th Nordic Workshop on Model Driven Software Engineering

Tampereen teknillinen yliopisto. Ohjelmistotekniikan laitos
Tampere 2009

A Validation of Martin's Metric

Sami Hyrynsalmi and Ville Leppänen

Department of Information Technology, University of Turku, Finland

Abstract. Robert C. Martin presented a software metric for a set of classes i.e. a package. The objective of the package level metric is to identify poorly designed packages. The Martin's metric actually consists of eight metrics which measure a few different characteristics of packages. The metric is widely known, but there is lack of theoretical and empirical evaluation of the Martin's metric. This paper evaluates the theoretical background of the metric against an evaluation framework and presents an experimental evaluation of five open-source software applications. The theoretical validation reveals a weakness in Martin's definition for cohesion. We propose a modification which is valid according to the evaluation framework.

1 Introduction

The most common objective of the software metrics is to identify weak parts of software. With them, one can assign these parts for more careful testing and inspection. Software metrics are tools for improving the quality of software.

Briand, Daly and Wüst [1] divide object-oriented product metrics in five levels: the *attribute*, *method*, *class*, *set of classes*, and *system* levels. The definitions of the first three are obvious, but the fourth is too vague to be usable. For example arbitrary selected classes from the system fulfil the definition, but these kinds of sets are rarely interesting.

Software architecture consists of components. One component, in object-oriented programming, can be one class or even sets of classes. A component is a better choice for an abstraction level than a set of classes. We will use word *package* rather than component, because in software engineering the term 'component' is overused and overdefined. In this paper the term 'package' refers to a set of classes which have a reason to locate in the same set. The used hypothesis is that the designer has decided to put them together as the classes together carry out a common service.

The focus of package metrics is not in locating faulty or risky source code. The package metrics only know the classes which belong to the package or interact with it, and dependencies between these classes. The package metrics can help to identify poorly designed packages and subsystems by investigating how classes are divided in the packages and how classes depend upon each other.

The next section will discuss about the Martin's metric. Section 3 will present the validation framework of Briand, Morasca and Basili [2] and evaluate the Martin's metrics in the framework. Section 4 presents an empirical study of five open-source software. Finally, Sect. 5 will present our conclusion.

1.1 Metrics in Literature

Hundreds of software metrics are presented in literature for object-oriented (OO) programming. One of the most notable OO-metric suites was presented by Chidamber and Kemerer [3, 4]. Their suite consists of six metrics which evaluate class cohesion, coupling, inheritance and complexity. Also Lorenz and Kidd [5] and Abreu and Carapuça [6] have presented similar metric suites at the class level.

Briand et al. have evaluated cohesion [1] and coupling [7] metrics, and presented frameworks for them. The evaluated metrics were also empirically tested [8, 9]. A few new and creative cohesion metrics were inspired by their work. For example Mäkelä and Leppänen [10] have suggested a client-based cohesion metric which analyses how clients use the class. From the clients' point of view, a class is cohesive if all clients use it in the same way.

On the contrary, only a few package level metrics are presented. Ducasse, Lanza and Ponisio [11] described Butterflies, tools for visualizing a package. They use some metrics in their diagram, but the measures are simple size and coupling metrics. Ponisio [12] continued the work and presented *Contextual Package Cohesion* (CPC), which is a cohesion metric based on clients view.

Zhou, Xu, Shi, Zhou and Chen [13] presented another client view based cohesion metrics for packages. Hautus [14] as well as Melton and Tempero [15] defined metrics for evaluating package's cyclic dependencies. Although the easiest way to handle cyclic dependencies is to model them as a directed graph, and then search for cycles in the dependency graph [16]. Also Lakos [17] has presented a metric for dependencies of a package.

2 Martin's Metric

Robert C. Martin's [18] metric is one of the most widely known software package measures. The focus of the metric is in identifying packages which are hard to maintain and reuse. Actually, Martin's metric consists of eight measures which evaluate various features of a package. These metrics and an additional cohesion metric are presented in Table 1.

Two coupling measures in the metric suite are afferent and efferent coupling. Afferent coupling C_a is the number of classes outside the package which depend on the package. Efferent coupling C_e is the number of classes outside the package on which the package depends upon.

The import coupling of a package (C_a) represents the responsibility of the package. Every client of the package is a reason for not to change the package. If one changes the package, in the worst case one has to refactor every client of the package. On the contrary, the export coupling of the package (C_e) represents the reasons for a change.

Martin derives from these coupling measures a metric for stability of the package. The stability of the packages does not measure the frequency or possibilities of the change. The metric measures how hard, measured by workload,

Table 1. Software package metrics defined by Martin.

Symbol	Name
C_a	Afferent Coupling
C_e	Efferent Coupling
I	Instability
N_a	Number of Abstract Classes
N_c	Number of Classes
A	Abstractness
D	Distance from the Main Sequence
D'	Normalized Distance
H	Relative Cohesion

it is to change the package. A responsible package is hard to change because a modification of it may lead to refactoring of other packages.

Martin's instability I, in contrast of stability, is:

$$I = \frac{C_e}{C_a + C_e} \ . \tag{1}$$

The metric's range is [0,1], where zero indicates a stable package. The value one implies that the package is maximally instable.

Two size measures of the suite are N_c and N_a. N_c is the number of classes in the package. Equally, N_a is the number of abstract classes or interfaces in the package. The abstractness A of the package is:

$$A = \frac{N_a}{N_c} \ . \tag{2}$$

Naturally, the range of A is [0,1], where zero indicates that the package has no abstract parts.

Martin [18] claims that every stable package should be abstract. Stable packages are hard to change because of their responsibilities, but they should also be abstract so they can be extended. On the other hand, an instable package should be concrete so their code can be easily changed. With these principles, Martin defines the relationship between A and I. The IA-graph is presented in Fig. 1.

In the ideal case, all packages lie in the points (0,1) or (1,0). Stable and abstract packages should be at the upper left corner at (0,1). Instable and concrete packages should be at the lower right corner at (1,0). In the real world there are different degrees of abstraction and stability, so all packages will not reside on these two points.

Martin argues that the coordinates (0,0) and (1,1) represent bad design. In the *Zone of Pain*, which is the area near lower left corner at (0,0), packages are rigid. They are difficult to extend because of their lack of abstractness and hard to change because of their responsibilities. The area near the upper right corner

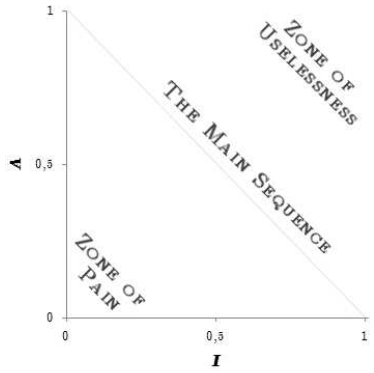

Fig. 1. The IA-graph.

at (1,1) is called *Zone of Uselessness*. Packages in this area are abstract but they do not have clients. Clearly, these kinds of packages are useless.

The ideal position for packages is as far as possible from these zones. *The Main Sequence* is a line that connects (1,0) to (0,1) and packages on it are in the right proportions of abstractness and stability. The *Martin's metric* for a package is the distance from the main sequence:

$$D = \frac{|A + I - 1|}{\sqrt{2}} \tag{3}$$

$$D' = |A + I - 1| \ . \tag{4}$$

Where D is the distance from the main sequence and it ranges between $[0, \frac{1}{\sqrt{2}}]$. D' is a normalized distance and its range is [0,1].

Martin [18] presented also a cohesion metric H which does not belong in the original metric suite. The metric is considered in this paper because it was defined by Martin, it is one of the few cohesion metrics purely presented for the package level and there is no validation of the metric. The metric H is an average of internal relationship per class of the package:

$$H = \frac{\rho + 1}{N_c} \ . \tag{5}$$

Where ρ is the number of internal class-to-class relationships in the package.

There are a few insufficiencies in the definitions. Martin did not define what are the criteria of coupling [1] or what are the internal relationships. In this paper, we assume that he means an explicit relationship between two classes, e.g. inheritance or use. Martin defines C_e as reasons for change, but did not mention how stable services should be handled. For example, Java's class library is stable because it has thousands of clients. One open question in calculating C_e is that should dependencies to stable services be ignored.

JDepend[1] is commonly used metric software, which calculates Martin's metrics. It defines afferent and efferent coupling with the number of the packages instead of the number of the classes. JDepend's version of the metrics is useful, because the package dependencies are known earlier than the class dependencies, so one can calculate the value of the instability on the analysis phase.

3 Theoretical Validation

One of the first attempts to provide criteria for complexity metrics is the research made by Weyuker [19] in 1988. As Weyuker's criteria have been proposed before the era of object-oriented paradigm, it is difficult to apply them for object-oriented classes or architectural level components. Besides, all metrics should not be seen similarly as complexity metrics. In complexity measures, a fundamental idea is that combining two or more elements as one produces a more complex element than any of its subelements. E.g., for coupling metrics this does not necessarily hold – the coupling "complexity" of the whole can be even less than any of its elements.

Briand, Morasca and Basili [2] presented in 1996 a set of required properties for different kind of metrics. More specifically, they proposed 3 requirements for size metrics, 4 requirements for cohesion metrics, and 5 requirements for length, complexity as well as coupling metrics. We consider that these requirements provide a proper framework for theoretical validation of metrics. In the following, we consider each of the metric measures of Martin's metric package with respect to the size, coupling and cohesions properties of Briand, Morasca and Basili (none of Martin's metrics fall into length and complexity categories).

As pointed out by Cherniavsky and Smith [20], it is possible create a software metric which fulfils properties but is neither practical nor reasonable. However, we argue that theoretical inspection can reveal possible flaws from the definition of the metric. Both theoretical and empirical study should be used in metric validation.

3.1 Size

Of the measures of Martin's metric, the measures N_a and N_c are clearly size measures. The derived measure $A = N_a/N_c$ is a relative size measure.

Briand et al. [2] assume that a measurable system consists of a set of elements E, and some relation $R \subseteq E \times E$. In the context of Martin's metrics package, the set E consists of either packages or classes within a package, and R is some usage/dependency relation. Briand et al. defined three properties for size metrics:

*Size Property 1 (**Non-negativity**).* The value of the package $P = (E_P, R_P)$ (as the system S) is non-negative:

$$SizeMetric(P) \geq 0 \tag{6}$$

Trivially, N_a, N_c and A fulfil the first size property.

[1] http://clarkware.com/software/JDepend.html

*Size Property 2 (**Null value**).* The value of the metric is null for package P when E_P is empty:

$$E_P = \emptyset \Rightarrow SizeMetric(P) = 0 \tag{7}$$

When the package is empty or there is no abstract classes, N_c, N_a and A are 0. (It can be defined that if $N_c = 0$, then $A = 0$.) The size metrics fulfil the second property.

*Size Property 3 (**Module Additivity**).* Let P' be the union of the packages P_1 and P_2. Assuming that there are no common classes in P_1 and P_2 then:

$$SizeMetric(P_1) + SizeMetric(P_2) = SizeMetric(P') \tag{8}$$

The property trivially holds for N_c and N_a. However, for relative size A, the Module Additivity property does not hold. We do not consider this as a failure for A, since A is derived from N_a and N_c, and Briand et al. did not intented their size propeties to be applied to relative sizes. It would rather seem that their framework should be extended with property requirements for relative sizes.

3.2 Coupling

The measures C_a and C_e of Martin are clearly coupling measures. Now, the system is $S(P) = (E_{in} \cup E_{out}, R)$, where $E_{in} \cap E_{out} = \emptyset$. E_{in} is the set of classes belonging to a package P and E_{out} contains all the other packages. Relation R is some subset of $E_{in} \times E_{out}$. The derived measure $I = C_e/(C_a + C_e)$ is a kind of relative coupling measure, and we consider that it cannot be evaluated with the next five property requirements defined by Briand et al (however, it fulfils the first two properties).

*Coupling Property 1 (**Non-negativity**).* The value of the package P coupling is non-negative:

$$CouplingMetric(P) \geq 0 \tag{9}$$

Clearly, C_a and C_e fulfil the property, since those both measure $|\{e_2 \in E_{out} | \exists e_1 \in E_{in} : (e_1, e_2) \in R\}|$, i.e. the number of certain kind of classes.

*Coupling Property 2 (**Null value**).* The value of the coupling of the package $P = (E_P \cup E_{out}, R_P)$ is null when E_P is empty:

$$E_P = \emptyset \Rightarrow CouplingMetric(P) = 0 \tag{10}$$

Clearly, C_a and C_e can be in the role of $CouplingMetric$.

*Coupling Property 3 (**Monotonicity**).* Let $P' = (E_{in} \cup E_{out}, R')$ be identical with package $P = (E_{in} \cup E_{out}, R)$ except that $R \subseteq R'$, then:

$$CouplingMetric(P) \leq CouplingMetric(P') \tag{11}$$

Measures C_a and C_e fulfil this property, since $R \subseteq R'$ represents for C_a and C_e, the increase of classes which are dependent on the package or on which the package depends, respectively.

*Coupling Property 4 (**Merging of connected systems**).* Let P' be the union of the package $P_1 = (E^1_{in} \cup E^1_{out}, R_1)$ and $P_2 = (E^2_{in} \cup E^2_{out}, R_2)$ such that P_1 and P_2 are connected in the sense of the relation, then:

$$CouplingMetric(P_1) + CouplingMetric(P_2) \geq CouplingMetric(P') \quad (12)$$

Because $P' = (E^{'}_{in} \cup E^{'}_{out}, R')$ is studied from the coupling viewpoint, $E^{'}_{in} = E^1_{in} \cup E^2_{in}$ and $E^{'}_{out} = (E^1_{out} \cup E^2_{out}) - E^{'}_{in}$. Since P_1 and P_2 are connected, then for C_a and C_e either

$$\exists (e_i, e_o) \in R_1 | e_o \in E^2_{in} \quad (13)$$

or

$$\exists (e_i, e_o) \in R_2 | e_o \in E^1_{in}. \quad (14)$$

As the resulting relation R' is union of R_1 and R_2 substracted by elements defined by Equations 13 and 14, then clearly the property holds for C_a and C_e.

*Coupling Property 5 (**Merging of unconnected systems**).* Let P' be the union of the packages $P_1 = (E^1_{in} \cup E^1_{out}, R_1)$ and $P_2 = (E^2_{in} \cup E^2_{out}, R_2)$ such that P_1 and P_2 are not connected in the sense of the relation, then:

$$CouplingMetric(P_1) + CouplingMetric(P_2) = CouplingMetric(P') \quad (15)$$

Since there are no connections, then $R' = R_1 \cup R_2$ and $R_1 \cap R_2 = \emptyset$. As C_a and C_e measure the right-hand size of the image of relation, then clearly by $R_1 \cap R_2 = \emptyset$ the property holds for C_a and C_e.

3.3 Cohesion

Martin's original metrics suite does not contain any cohesion measure, but later Martin has proposed $H = (\rho + 1)/N_c$ [18], which is a cohesion metric. The measure ρ is not well-defined by Martin, but it is expected to be some kind of size measure of the cohesion relation. In the following, we assume that $\rho = |R|$, when the system for a package $P = (E, R)$ as before. Briand et al. proposed four requirements for cohesion metrics. Next, we evaluate H with respect to those.

*Cohesion Property 1 (**Non-negativity and Normalization**).* For a component P, there is a fixed upper bound max such that

$$CohesionMetric(P) \in [0, max] \quad (16)$$

Unfortunately, the value 0 is not possible for H, since the numerator of H's formula is always at least 1 (as $\rho \geq 0$). There is neither a fixed upper bound for H.

If ρ will represent the amount of connected (by e.g. some usage relation) class pairs (as is usual in cohesion metrics), its value would vary between $[0, \frac{N_c(N_c-1)}{2}]$. Then, the maximum value for H would depend on the amount of classes, N_c, and there would be problems with the fixed upper bound criteria. Thus, we consider that H fails to fulfil the Non-negativity and Normalization property.

*Cohesion Property 2 (**Null value and maximum value**).* For a component $P = (E, R)$ the cohesion value is zero when R is empty and it is max when R is maximal:

$$\begin{aligned} R = \emptyset &\Rightarrow CohesionMetric(P) = 0 \\ R = maximum &\Rightarrow CohesionMetric(P) = max \end{aligned}$$

Clearly, H does not fulfil the above conditions. When $R = \emptyset$ and $N_c \neq 0$, the value of H is $\frac{1}{N_c}$. When R is maximal (e.g. $R = E \times E$), the value of H again depends on N_c having no fixed upper bound.

*Cohesion Property 3 (**Monotonicity**).* Let package $P' = (E', R')$ be identical with package $P = (E, R)$ except that $R \subseteq R'$.

$$CohesionMetric(P) \leq CohesionMetric(P') \tag{17}$$

As the ρ in the formula of H is a (monotone) size measure of R, clearly H fulfils the Monotonicity condition.

*Cohesion Property 4 (**Cohesive Modules**).* Let $P' = (E', R')$ be a combination of mutually unconnected components $P_1 = (E_1, R_1)$ and $P_2 = (E_2, R_2)$. It must hold for P' that

$$\max\{CohesionMetric(P_1), CohesionMetric(P_2)\} \geq CohesionMetric(P') \tag{18}$$

Because the components are mutually unconnected, we have $\rho' = \rho_1 + \rho_2$ and $N'_c = N_{c1} + N_{c2}$. Then, $H(P') = \frac{\rho_1+\rho_2+1}{N_{c1}+N_{c2}}$. Without loss of generality, we assume that $H(P_1) \geq H(P_2)$. Then

$$\begin{aligned} H(P_1) &\geq H(P') \\ \frac{\rho_1 + 1}{N_{c1}} &\geq \frac{\rho_1 + \rho_2 + 1}{N_{c1} + N_{c2}} \\ \frac{(N_{c1} + N_{c2})(\rho_1 + 1)}{N_{c1}} - (\rho_1 + 1) &\geq \rho_2 \\ \frac{N_{c2}(\rho_1 + 1)}{N_{c1}} &\geq \rho_2 \\ \frac{\rho_1 + 1}{N_{c1}} &\geq \frac{\rho_2}{N_{c2}} \\ \frac{\rho_1 + 1}{N_{c1}} + \frac{1}{N_{c2}} &\geq \frac{\rho_2 + 1}{N_{c2}} \\ H(P_1) + \frac{1}{N_{c2}} &\geq H(P_2) \end{aligned}$$

Since $1/N_{c2} > 0$, the metric H fulfils the condition.

3.4 Summary

We presented an analysis of the measures/metrics N_a, N_c, C_a, C_e, and H of Martin's metrics package. All of them, except H, were observed to fulfil the requirements of the evaluation framework of Briand, Morasca and Basili. The instability measure I and relative abstractness A were argued to be *relative* coupling and size measures. As the framework does not give requirements for relative measures, their theoretical validity could not be evaluated. Also, the measures D and D', which are derived from I and A, could not be evaluated.

There are two problems in the definition of the H. First, the additional one in the numerator of H prevents reaching the value 0. The other problem is that ρ depends polynomially on N_c (the denominator), and thus there is no fixed upper bound for H. The problem can be removed by replacing the denominator with the maximal size of relation: $N_c(N_c-1)/2$. However, then there would be a problem with $N_c = 1$. Thus, we end up proposing a variant H' for H:

$$H' = \begin{cases} 1 & \\ \frac{\rho}{\frac{N_c(N_c-1)}{2}} & \end{cases} = \begin{cases} 1 & \text{when } N_c = 1 \\ \frac{2\rho}{N_c(N_c-1)} & N_c > 1 \end{cases} . \tag{19}$$

It is straightforward to check that H' fulfils all the cohesion conditions of Briand et al. Finally, we observe that H' quite naturally resembles the cohesion metric Co' proposed by Briand, Daly and Wüst [1] for calculating the cohesion of a single class:

$$Co' = 2 \cdot \frac{|E|}{|V|(|V|-1)} . \tag{20}$$

Where V represents the elements of a class and E describes the (internal) connections of the elements of V.

4 Experimental Validation

There are a few different ways for empirical study of software package metrics. The most comprehensive approach is to calculate statistical differences between the maintenance cost of the packages and the Martin's metrics in many different versions. However, no such data is available in open source and rarely on any other software projects.

In this experimental validation, we measured five open source Java applications which have been developed over many years. Vuze, Tomcat, ArgoUML, Xerces, and jEdit are highly reputable applications from different fields. The outcome of our experimental study is that the applications believed to be well-made perform well according to Martin's metric suite, we argue that they provide a validation for the Martin's metric. Individual packages of application yield bad values indicating that the whole scale is indeed available for badly designed applications. In some cases below, we also study the reason for bad values. We admit that our validation is only partial, since we could not find a huge but badly designed application.

In this study, dependencies are explicit usage and inheritance relations from package to package. We will focus only to those parts of the software which have been produced by the project itself i.e. third party's packages are not included in the study.

Vuze[2] (v. 4.2.0.0), formerly Azureus, is a BitTorrent protocol client. Vuze's package `com.aelitis` and its subpackages IA-graph and D' histogram are shown in Fig. 2. As shown in the histogram, 10% of all packages are in the main sequence and 75% are no further than 0.21 distance from the line. Clearly, Vuze conforms to Martin's design principles.

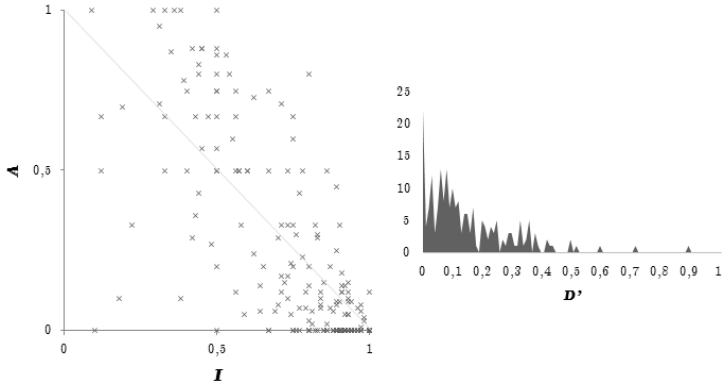

Fig. 2. Vuze's IA-graph and histogram of D'.

For example the package `com.aelitis.azureus.core.util` consists of 18 concrete and 2 abstract classes. The package is used by 47 clients and its normalized distance is 0.72. The package offers basic services, like calculating hash codes. Clearly, these kinds of classes should be implemented with interfaces if there is any possibility of change.

In the package `com.aelitis.azureus.login` is only one concrete class and it is used by one client. The D' value of package is 0.9, because the class uses services of 9 packages. All of these service packages are from Java's class library. They are not that kind of reason of change what Martin describes. If we skip the class library packages, the D' value will be zero.

Apache Tomcat[3] (v. 6.0.18) is an HTTP web server and a servlet container. In this study we evaluated Tomcat's packages under the `org.apache` package. IA-graph and histogram of the D' are shown in Fig. 3. As shown in the histogram, also Tomcat has a lot of the packages on the main line.

[2] http://azureus.sourceforge.net/
[3] http://tomcat.apache.org/

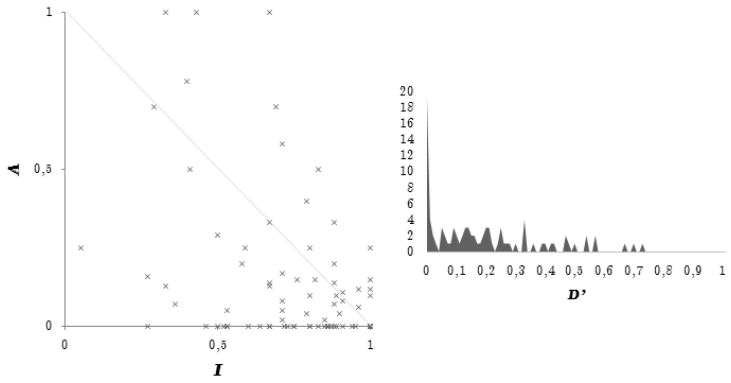

Fig. 3. Tomcat's IA-graph and histogram of D'.

The package `org.apache.tomcat.util.res` has the biggest D' value (0.73) of the system. There is only one class in the package and it has eight client packages. The class `StringManager` provides localization services for the system. If one follows Martin's principles, there should be an abstract package with one or more interfaces for localization services and the class `StringManager` should implement these abstractions. On the contrary, if there is absolutely no chance that the class will change, then the additional package for the interface would just increase the complexity of the system.

It should be noted that all services, which `org.apache.tomcat.util.res` uses, come from the Java's class library. If we ignore these classes, the normalized distance of package would be one. As shown, the stability of the server affects Martin's metric.

ArgoUML[4] (v. 0.28) is a UML diagram drawing application. Figure 4 presents ArgoUML's package `org.argouml` and its subpackages. The histogram of the ArgoUML differs a lot when compared to Vuze or Tomcat. On the other hand, the mean and median of the packages' D' values are under 0.20. Also the third quartile is 0.28 so most of the packages are still quite close to the main sequence.

The D' value of the package `org.argouml.i18n` is 0.90. The package offers localization service as `org.apache.tomcat.util.res` does in Tomcat. The `Translator` is only class in the package and there are 495 client classes in 43 packages. It is possible to hide the services of the class behind an interface, as in Tomcat.

The D' value of the package `org.argouml.application.helpers` is 0.76. It consists of three classes which offer miscellaneous services. Intuitively, the package is not a cohesive set of services. The class `ApplicationVersion` provides the version number of the application and the address to the online manual. The

[4] http://argouml.tigris.org/

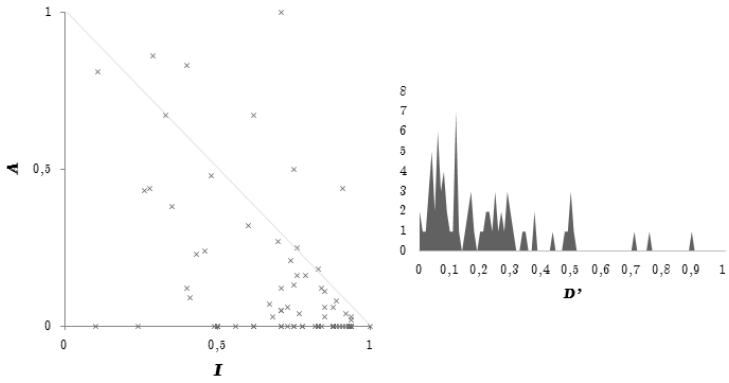

Fig. 4. ArgoUML's IA-graph and histogram of D'.

classes `ResourceLoader` and `ResourceLoaderWrapper` offer a list of system's resources. Over a hundred classes use the services of the package, and clearly the package could be refactored in two parts or the classes could be moved to other packages.

Apache Xerces2 J[5] (v. 2.9.1) is a XML parser library for Java. In Fig. 5 are shown the histogram of D' and the IA-graph of Xerces' package `org.apache` and its subpackages. The package `org.apache.xerces.impl.xs.util` consists of 14 concrete classes, which implement a few advanced data structures. In the evaluated system, there are 36 classes which depend upon the package. Clearly, these kinds of services could be defined with an interface.

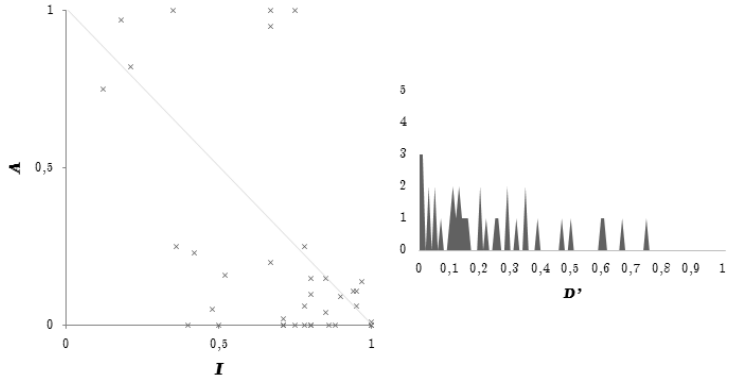

Fig. 5. Xerces' IA-graph and histogram of D'.

[5] http://xerces.apache.org/xerces2-j/

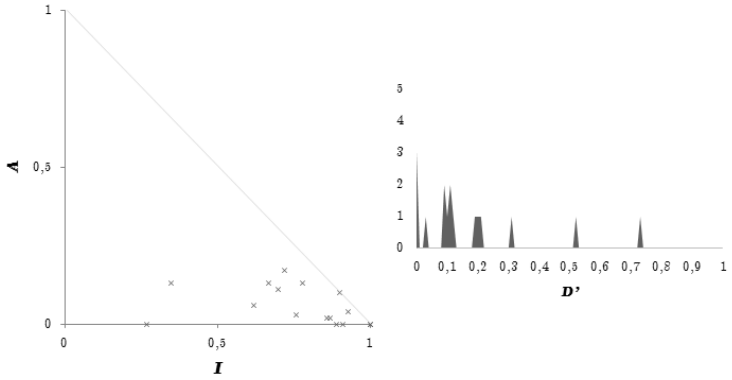

Fig. 6. jEdit's IA-graph and histogram of D'.

jEdit[6] (v. 4.2) is a text-editor for programmers. Figure 6 shows the IA-graph and the histogram of the package `org.gtj.sp` and its subpackages. The D' value of the package `org.gtj.sp.jedit.msg` is 0.72 and it consist of messages that the system uses in the internal communication. The package is in a cycle with the package `org.gtj.sp.jedit.gui`. The messages should be defiened by external interfaces, because messages are volatile. E.g. new functionality of the software or changes in the GUI may have effect to the `org.gtj.sp.jedit.msg` package.

The value of D' of the package `org.gtj.sp.util` is 0.51. The package consists of classes which are used by almost every other package in the system. Also the package is possible to implement by defining external interfaces for the services. Now, changes in a few classes might require refactoring of the whole system.

For a summary, the histogram of D' and IA-graph of the evaluated packages are shown in Fig. 7. As can be seen from the histogram, the most of the packages are quite near of the main sequence. In Table 2 are shown the statistical values of D' from measured systems. 75 % of the evaluated packages are no further than 0.25 distance from the main sequence. We argue that evaluated packages conform the Martin's principles.

5 Conclusion

This paper pointed out that four of Martin's eight metrics fulfil Briand's et al. properties. Three of the measures we argued to be derived relative measures of the first four measures. For such relative measures, Briand's et al. framework does not seem to be applicable. The package's cohesion metric H was found not to fulfil cohesion metric properties, and thus we proposed a variation H' for it, fulfilling the properties.

[6] http://www.jedit.org/

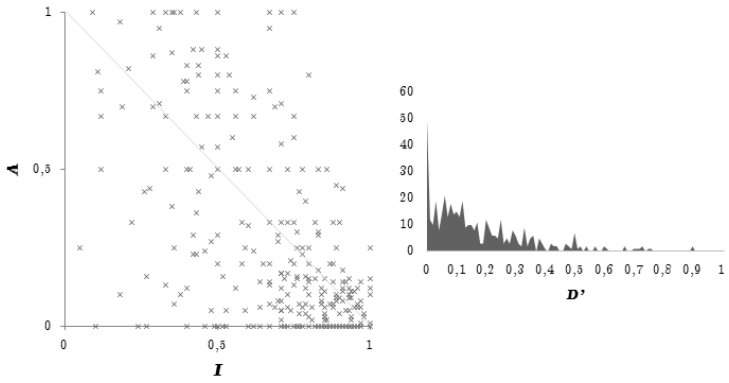

Fig. 7. All 430 packages in the same IA-graph and their histogram of D'.

Table 2. Statistical values of the evaluated packages.

	mean	**median**	**min**	**Q1**	**Q3**	**max**	**N**
Vuze	0.15	0.10	0	0.05	0.21	0.90	208
Tomcat	0.18	0.14	0	0.01	0.27	0.73	90
ArgoUML	0.20	0.16	0	0.07	0.28	0.90	80
Xerces	0.22	0.15	0	0.05	0.32	0.75	36
jEdit	0.18	0.11	0	0.08	0.20	0.73	16
All	0.17	0.12	0	0.05	0.25	0.90	430

Besides our theoretical validation, we made an attempt to validate Martin's metric suite also experimentally by evaluating five large open-source applications. Our experimental results show that one can recognize poorly designed packages. Since the outcome of our experimental study is that the application believed to be well-made perform well according to Martin's metric suite, we argue that they provide a validation for the Martin's metric. The validation is only partial, since we could not find a huge but badly designed application.

References

1. Briand, L.C., Daly, J.W., Wüst, J.: A unified framework for cohesion measurement in object-oriented systems. Empirical Software Engineering **3**(1) (1998) 65–117
2. Briand, L.C., Morasca, S., Basili, V.R.: Property-based software engineering measurement. IEEE Transactions on Software Engineering **22**(1) (January 1996) 68–86
3. Chidamber, S.R., Kemerer, C.F.: Towards a metrics suite for object oriented design. In: OOPSLA '91: Conference proceedings on Object-oriented programming systems, languages, and applications, New York, NY, USA, ACM (1991) 197–211
4. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Transactions on Software Engineering **20**(6) (June 1994) 476–493

5. Lorenz, M., Kidd, J.: Object-Oriented Software Metrics: A Practical Guide. The Object-Oriented Series. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1994)
6. Abreu, F.B.e., Carapua, R.: Object-oriented software engineering: Measuring and controlling the development process. In: Proceedings of the 4th International Conference on Software Quality, McLean, VA, USA (October 1994) 1–8 Revised version.
7. Briand, L.C., Daly, J.W., Wüst, J.K.: A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on Software Engineering **25**(1) (January/February 1999) 91–121
8. Briand, L.C., Wüst, J., Daly, J., Porter, V.: A comprehensive empirical validation of design measures for object-oriented systems. In: METRICS '98: Proceedings of the 5th International Symposium on Software Metrics, Washington, DC, USA, IEEE Computer Society (November 1998) 246–257
9. Briand, L.C., Wüst, J., Lounis, H.: Replicated case studies for investigating quality factors in object-oriented designs. Empirical Software Engineering **6**(1) (2001) 11–58
10. Mäkelä, S., Leppänen, V.: Client-based cohesion metrics for Java programs. Science of Computer Programming **73**(5-6) (March 2009) 355–378
11. Ducasse, S., Lanza, M., Ponisio, L.: Butterflies: A visual approach to characterize packages. In: METRICS '05: Proceedings of the 11th IEEE International Software Metrics Symposium, Washington, DC, USA, IEEE Computer Society (2005) 7
12. Ponisio, M.L.: Exploiting Client Usage to Manage Program Modularity. PhD thesis, University of Bern, Switzerland (June 2006)
13. Zhou, T., Xu, B., Shi, L., Zhou, Y., Chen, L.: Measuring package cohesion based on context. In: WSCS '08: Proceedings of the IEEE International Workshop on Semantic Computing and Systems, Washington, DC, USA, IEEE Computer Society (July 2008) 127–132
14. Hautus, E.: Improving Java software through package structure analysis. In: The 6th IASTED International Conference Software Engineering and Applications, ActaPress (November 2002)
15. Melton, H., Tempero, E.: The CRSS metric for package design quality. In: ACSC '07: Proceedings of the thirtieth Australasian conference on Computer science, Darlinghurst, Australia, Australian Computer Society, Inc. (January 2007) 201–210
16. Lakshmi Narasimhan, V., Hendradjaya, B.: Some theoretical considerations for a suite of metrics for the integration of software components. Information Sciences **177**(3) (2007) 844–864
17. Lakos, J.: Large-scale C++ software design. Addison Wesley Professional Computing Series. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA (1996)
18. Martin, R.C.: Agile Software Development: Principles, Patterns, and Practices. Alant Apt Series. Prentice Hall, Upper Saddle River, NJ, USA (2002)
19. Weyuker, E.J.: Evaluating software complexity measures. IEEE Transactions on Software Engineering **14**(9) (September 1988) 1357–1365
20. Cherniavsky, J.C., Smith, C.H.: On Weyuker's axioms for software complexity measures. IEEE Transactions on Software Engineering **17**(6) (June 1991) 636–638