DEPARTMENT OF COMPUTER SCIENCE

LUDWIG-MAXIMILIANS-UNIVERSITÄT

Bachelor's Thesis in Computer Science

# Complexity Measures in Software Engineering: A Systematic Comparison and Evaluation on Software-Component-Level

Simon Lund

DEPARTMENT OF COMPUTER SCIENCE

LUDWIG-MAXIMILIANS-UNIVERSITÄT

Bachelor's Thesis in Computer Science

# Complexity Measures in Software Engineering: A Systematic Comparison and Evaluation on Software-Component-Level

| | |
|---|---|
| Author: | Simon Lund |
| Supervisor: | Prof. Dr. Dirk Beyer |
| Mentor: | Thomas Lemberger |
| Submission Date: | 12.10.2020 |

I confirm that this bachelor's thesis in computer science is my own work and I have documented all sources and material used.

Munich, 12.10.2020 Simon Lund

## Acknowledgments

First, I would like to thank Thomas Lemberger for his great mentoring. Whenever I got stuck, you helped me to find a solution. Moreover, you patiently answered my questions and gave me many helpful tips for this thesis. Additionally many thanks to Anton and Joachim, who helped me a lot by reviewing and correcting this thesis. Last, I want to thank my family for their lifelong support.

# Abstract

There are many measures on class- and statement-level which assess different characteristics of software systems. However, the selection and diversity of package-level measures, especially of those focusing on the dependencies of a software system, is very limited. This is problematic because the complexity of a large software system emerges from the dependencies between the components of the system, i.e., we need to be able to measure these dependencies with regard to different characteristics of the dependency graph of a system. Therefore, we propose and evaluate several package-level measures that cover several of these characteristics. For this purpose we use Weyuker's Properties to conduct a formal analysis of the measures and implement the prototypical measurement tool Jade to evaluate the applicability of the measures on the example of CPAchecker. Based on the analysis we show that three of the proposed measures - $DCM_{CC}$, $DLM$ and *P-DepDegree* - are applicable and useful for measuring the complexity of a system based on its dependencies.

**Keywords:** software measures, package-level, dependencies, dependency graph, DepDegree, cohesion, locality

# Contents

# 1 Introduction

Software measures provide key indicators for software systems and allow us to compare and evaluate the complexity of software systems and their components. The obtained insights can be used, for instance, may be used during refactoring to identify overly complex components as well as to predict maintainability and testability of software components. [1]. For this purpose, many measures have been proposed in the last two decades. Most of them focus on certain aspects of the system's complexity on class- and statement-level. However, the selection and diversity of package-level measures, especially of those focusing on the dependencies of a software system, is very limited. This is problematic because the complexity of a large software system emerges from the dependencies between the components of the system, i.e., we need to be able to measure these dependencies with regard to different characteristics of the dependency graph of a system. For example, consider a software system of 1000 classes with many dependencies between these classes. A classic measures like LOC, the number of lines of code in a system, is not capable of capturing the complexity emerging from the dependencies. Therefore, the measurement value of LOC for a system can deviate strongly from reality and the actual complexity of a system. Hence, we need measures which take the dependencies of a system into account. Therefore, we propose the following five dependency measures on package-level focusing on three different characteristics - cohesion, locality and transitivity - of dependencies between the packages of a software system respectively the dependency graph of a software system. That is, we propose three variants for dependency cohesion $DCM_{LCOM3}$, $DCM_{SIM}$, $DCM_{CC}$ inspired by existing class cohesion measures, P-DepDegree inspired by DepDegree [4] focusing on transitivity as well as $DLM$ focusing on the locality of the dependencies of a package. In addition, we conduct a theoretical and practical evaluation of the proposed measures and compare the results with existing package-level measures. For the theoretical part, we establish Weykuer's Properties [29] on package-level to do a formal analysis of the proposed and existing measures. For the practical part of the evaluation we use our prototypical measurement tool Jade implemented in Python to calculate the proposed and existing measures for all packages on the example of CPAchecker [11]. Finally, we evaluate the applicability of the proposed measures based on the data sets containing the measurement values of the different measures for all packages of CPAchecker.

# 2 Related Work

## 2.1 Measures

Software measures can be categorized based on the level of the software system on which they work, i.e., system-level, package-level, class-level or statement-level. In this thesis we propose several package-level measures which focus on the package's dependencies. However, there are additional categories regarding special types of measures, a selection of which are presented below. These can be used together with the proposed measures to facilitate a more comprehensive analysis of a software system. Furthermore, there are also some package-cohesion measures [14, 23, 30, 3, 28, 26] similar to the proposed dependency cohesion measures. However, their definitions differ from the measures proposed in this thesis. For example, the measure $PCoh$ [14] calculates a measurement value for a package $P$ based on the relations between the classes, interfaces and subpackages of $P$ which can include aggregation, inheritance or reference. We, however, consider a package as a set of classes including interfaces and therefore the proposed dependency cohesion measures do not take into account the subpackages of a package, i.e, the hierarchical order of a system. Another example is *Common-Use* [26], which in contrast to the proposed dependency measures calculates the cohesion value of a package based on the incoming the dependencies. Furthermore, some of the measures are based on the actual interactions, i.e., method invocations, between the classes of a software system and not only their dependencies.

**Annotation Measures [20].** These are only applicable on software systems implemented in a programming languages which features the annotation of classes, methods and variables (e.g. `@Deprecated` in Java). Although annotations simplify metadata configuration, their abuse and misuse can reduce source code readability and complicate its maintenance [20]. Therefore, annotation measures are used to assess used source code annotations based on certain characteristics and help developers identify problematic code annotations.

**Inheritance Measures [19].** These quantify the complexity of the inheritance tree of a software systems. Key characteristics are the branching of the inheritance tree and its depth. Per definition they are only of interest regarding the evaluation of object oriented design systems as they are based on the aspects of inheritance and polymorphism.

**Change Measures [10, 12].** These are related to the changes made during software development and can be used to evaluate the complexity of software system regarding its development cycles. For this purpose the measures evaluate different aspects of the commit history of a software project. This helps to track the development progress of a project and identify possible causes of errors.

**Package Coupling Measures [13].** This measure suite consists of several measures for measuring coupling between packages. Hence, these focus on the interactions between packages and not similarities within a package. However, using these measures to reduce coupling between packages can result in packages with higher dependency cohesion as reducing coupling between packages may result in packages with classes that share more of their dependencies.

## 2.2 Evaluation Frameworks

We chose Weyuker's Properties for the theoretical evaluation of the proposed measures because they are considered to be simple and straightforward [25, 29]. However, there are other frameworks that could be also used to analyze the proposed measures. The following is a selection of such frameworks.

**Mathematical Properties [8].** This generic framework proposes properties for different measurement concepts such as size, length, cohesion and coupling. These concepts, in turn, are based on precise mathematical concepts and for each one of them several properties are proposed that cover different characteristics of the respective concept.

**Class Cohesion Metrics Properties [7].** This framework focuses on cohesion measures and uses the following four properties that are based on precise mathematical concepts [8] to analyze such measures: "Non-negativity and normalization", "null value and maximum value", "monotonicity", "merging of unconnected classes". As it provides a standardized terminology and formalism for expressing measures, it facilitates comparison, evaluation and empirical validation of existing cohesion measures.

# 3 Weyuker's Properties

A newly proposed complexity measure is acceptable only when its usefulness has been proven by a validation process [25]. For this reason, we use Weyuker's Properties, a set of nine abstract properties that allows us to formally evaluate and compare the measures proposed in this thesis. Although several authors criticize Weyuker's Properties, they still play an important role in the evaluation of software complexity measures [24, 25]. That is because Weyuker's Properties are considered to be simple and straightforward as they serve as a basis for the evaluation of software complexity measures [25, 29]. Hence, the properties are not meant to be a conclusive evaluation of software measures, but as formal prerequisites of measures to satisfy an intuitive understanding of software complexity [5].

## Definition

Since this thesis focuses on package-level measures, the following definition of Weyuker's Properties is based on software components and their dependencies, i.e., the system's packages and classes. Therefore, we consider $P_i \in \mathbb{P}$ our universe, i.e., the set of all possible packages $P_i$, and beyond that we consider arbitrary systems $S$ with $P_S$ the set of all packages of $S$, i.e, $P_S = P_1, P_2, ..., P_n$. Furthermore, we consider $C_P$ the set of all classes $c_1, c_2, c_3, ...$ of $P$ with $c_1, c_2, c_3, ...$ as unique identifiers of the classes in $P$ in regard to system $S$. Thus, we can distinguish between all packages and classes within a system S by their identifier. Based on these definitions, we also define the following operators on which Weyuker's Properties rely.

- $\mu(X)$ $-$ the measurement value of a package X for measure $\mu$
- $P \equiv Q$ $-$ the packages P and Q are functionally equivalent
- $P \triangleright Q$ $-$ the extension of $C_Q$ by the classes of $P$

We consider two packages $P$, $Q$ in to be functionally equivalent if, and only if, we can emulate the functionality of $P$ using $Q$ and vice versa. Thereby, we define the functionality of a package as the sum of the functionality of the classes in the package which, in turn, is defined by the relation of the class with other classes and its public methods. Furthermore, we also refer to $P \triangleright Q$ as the composition of $P, Q$ which extends

$C_Q$ by the classes of $P$ in system $S'$ with $P_{S'} = P_S$ where $C_Q = C_{P \triangleright Q} = C_Q \cup C_P$. In addition, if $\mu$ $\mu(P \triangleright Q) = \mu(Q \triangleright P)$ holds for any two packages $P, Q$ for a measure $\mu$, then we consider $P \triangleright Q$ to be commutative and write $P + Q$ instead. Having formally established these operators, we can now define Weyuker's Properties as follows.

**Property 1:** $\exists P, Q : \mu(P) \neq \mu(Q)$
This property states that a measure must not treat all packages as equally complex. If the property is not satisfied by a measure $\mu$, then each package of a system would have the same measurement value (i.e. $\mu(X) = c$; c constant). Consequently, a legitimate measure must satisfy Property 1.

**Property 2:** *There are only finitely many packages of complexity c, with* $c \geq 0$
This property is a strengthening of Property 1. Intuition implies that a measure is not sensitive enough if it divides all programs into just a few complexity classes. Property 2 is an attempt to formalize this intuition [29].

**Property 3:** $\exists P, Q : (P \not\equiv Q) \wedge (\mu(P) = \mu(Q)$
This property stresses the necessity that a legitimate measure must focus on the complexity, not the functionality of a package. Therefore, packages of the same complexity should have the same measurement value in regard to a measure $\mu$ despite their functionality.

**Property 4:** $\exists P, Q : P \equiv Q \wedge \mu(P) \neq \mu(Q)$
This property states that not only the functionality of a package, but also its implementation is significant for its measurement value. This means that a legitimate measure must also differentiate between implementation details on which the complexity for a package can be based.

**Property 5:** $\forall P, Q : \mu(P) \leq \mu(P \triangleright Q) \wedge \mu(Q) \leq \mu(P \triangleright Q)$
This property ensures that measure $\mu$ is a monotone function and that the measurement values of $\mu$ for a given set of packages increases with their complexity. Thus, merging two packages results in a measurement value greater than or equal to that of the individual packages for a measure $\mu$ satisfying Property 5.

**Property 6:**

a) $\exists P, Q, R : \mu(P) = \mu(Q) \wedge \mu(P \triangleright R) \neq \mu(Q \triangleright R)$

b) $\exists P, Q, R : \mu(P) = \mu(Q) \wedge \mu(R \triangleright P) \neq \mu(R \triangleright Q)$

This property states that by merging each of two packages $P, Q$ of the same complexity with a third package $R$ their measurement values of $\mu$ need not increase uniformly. If $P \triangleright Q$ is commutative regarding $\mu$, then we can simplify this property to $\exists P, Q, R : \mu(P) = \mu(Q) \wedge \mu(P + R) \neq \mu(Q + R)$.

**Property 7:** $\exists P, Q : Q = p(P) \wedge \mu(P) \neq \mu(Q)$
This property states that for a measure $\mu$ there are packages $P, Q$ such that $Q$ is derived from $P$ by permuting the order of its classes and $P, Q$ have different measurement values.

**Property 8:** *If we rename package P to Q, then* $\mu(P) = \mu(Q)$
This property states that measure $\mu$ is independent of the naming of the packages. Therefore, renaming a package has no impact on the measurement value of a measure $\mu$ satisfying this property.

**Property 9:** $\exists P, Q : \mu(P) + \mu(Q) < \mu(P \triangleright Q)$
This property expresses the possibility that merging two packages can result in increasing complexity. Reasons for this can be, for instance, the non-linear fashion of the measure or additional interactions between the packages as a result of the merging.

## Usage

We introduced all properties for the sake of completeness, but not all of the properties are of significance for the evaluation of the proposed package-level measures. This is because Weyuker's Properties were originally proposed for the evaluation of statement-level measures, i.e., measures used to determine the complexity of a single function. Thus, we will not use the following properties for the evaluation of the proposed package-level measures. The remaining properties will be considered.

- Property 1 (**Always True**): None of the measures considered in this thesis maps all packages to a constant value. Hence, all measures satisfy this property.
- Property 2 (**Always True**): There is a finite number of applications, each having a finite number of classes [6, 9]. Under the assumption that there can only be a maximum of two dependencies between two classes ($x \to y$, $y \to x$), there is also a finite number of dependencies. It follows that there is a finite number of packages that contain one or more classes. Thus, there is also a finite number of packages of the complexity $c$. Hence, all package-level measures defined in this thesis satisfy this property.

- Property 7 (**Not Applicable**): This property has no significance for package-level measures as one can not derive a package from another package only by changing the order of its classes.
- Property 8 (**Always True**): The package-level measures proposed in this thesis are solely based on the classes of a package and their dependencies. Thus, the name of the package has no impact on its corresponding measurement values.

# 4 Existing Measures

In addition to the measures proposed in this thesis we introduce the following existing measures that help us to assess the proposed measures, understand unexpected behavior and better interpret the measurement values of outliers.

**Number of Classes.** We define Number of Classes ($NOC$) as the number of classes in a package $P$ and formally define $NOC$ based on $C_P$ as follows.

$$NOC(P) = |C_P| \tag{4.1}$$

Hence, we can use this measure to better interpret measurement values, which are affected by the size of a package. This allows us to normalize measurement values of packages of different size and compare them under consideration of their size.

**Afferent Coupling.** We define the Afferent Coupling ($Ca$) [22] value for a package $P$ as the number of classes outside of $P$ that depend on classes of $P$. We consider therefor $D_c$ the set of dependencies for a class $c \in C_P$ and $Ca$ for $P$ as follows.

$$Ca(P) = \sum_{Q \in P_S \setminus P} |\{\, c \mid c \in C_Q : D_c \cap C_P \neq \emptyset \,\}| \tag{4.2}$$

Thus, $Ca$ allows us to estimate the relevance of a package $P$ since the more classes depend on classes of $P$ the more relevant $P$ is for the system $S$. This is an important measure because we can determine the responsibility of $P$ and the impact of change in $P$ on other packages in $S$.

**Efferent Coupling.** We define the Efferent Coupling ($Ce$) [22] value for a package $P$ as the number of classes in $P$ that depend on classes outside of $P$. By means of the definitions we can formally define $Ce$ as follows.

$$Ce(P) = |\{\, c \mid c \in C_P : D_c \setminus C_P \neq \emptyset \,\}| \tag{4.3}$$

Thus, $Ce$ measures the effect of changes in $S$ on $P$. It follows that the higher $Ce$ for $P$ the higher the probability that $P$ is affected by changes in the rest of $S$. At this point $NOC$ comes in handy as we can use it to determine the proportion of classes in $P$ that depend on classes outside of $P$, which facilitates a context-sensitive interpretation of $Ce$.

**Instability.** The Instability measure (I) [22] for a package $P$ calculates the stability of $P$ based on the measurement values of $Ca$ and $Ce$ for $P$. Thereby, we define the stability of a package as the difficulty of modifying the package where the difficulty depends on the number of classes that depend on the package. Hence, a package with a high $Ca$ value is very stable since modifications lead to many more obligatory modifications in the dependent classes. Thus, we can formally define the instability of $P$ in $S$ as follows.

$$I(P) = \frac{Ce}{Ce + Ca} \tag{4.4}$$

The measurement value for a package $P$ ranges between 0 and 1 where $I(P) = 0$ means that $P$ is stable in the sense that the classes of $P$ do not depend on classes outside of $P$ and $I(P) = 1$ signifies that $P$ is unstable as $P$ depends on other classes but no class outside of $P$ depends on classes in $P$, i.e., $Ca(P) = 0$. It follows that the closer $I(P)$ is to 0 the less responsible is $P$ and the easier are modifications within $P$.

# 5 Proposed Package Measures

In this chapter we propose the following three package measures in a uniform manner:

1. Dependency Cohesion Measure
2. Package DepDegree
3. Dependency Locality Measure

These measures focus on the $D_P$ the set of dependencies of a package $P$. The first measure focuses on the proportion of dependencies in $D_P$ that are shared among the classes of $P$, the second one determines the overall proportion of the system on which a package directly and indirectly depends and the last one measures the distribution of the dependencies of $P$ within the system's package tree. Note that the first two measures are based on existing measures defined on class- or statement-level. The last measure, however, was developed because existing package-level measures treat the dependencies of a package $P$ equally and their location has not been considered so far.

## 5.1 Dependency Cohesion Measure

The proposed Dependency Cohesion Measure ($DCM$) is based on the dependency cohesion of a package, i.e., the degree to which the classes of a given package have the same dependencies. The measure is inspired by existing class-level cohesion measures, which quantify the overall cohesion of the methods of a class with respect to the number of attributes that share the class's methods with each other. Thus, it helps to understand how the dependencies of a package $P$ are used and distributed over the classes of $P$ which is a useful information, for instance, to determine the impact of changes regarding the dependencies of $P$. After the evaluation of more than twenty measures [17], three measures were found to be fitting candidates. As the measures differ in their concepts, we decided to adapt all of them and define three variants of $DCM$.

### 5.1.1 Variant based on LCOM3

This variant is based on LCOM3 [16], an improved version of the original lack of cohesion methods (LCOM) measure [9]. It is a counting measure whereby the key idea is to count the number of pairs of classes for a package $P$ that share at least one dependency. Therefore, we define the set of pairs of classes $a$, $b \in C_P$ for which $D_a \cap D_b \neq \emptyset$ holds as follows.

$$M_P \coloneqq \{\ (c_i, c_j) \mid c_i, c_j \in C_P : i < j \wedge D_i \cap D_j \neq \emptyset\ \}\ (\text{with } c_1,\ c_2,\ ...,\ c_n \in C_P) \tag{5.1}$$

Note that $M_P$ is an asymmetrical relation over $P$ ($\forall x, y \in P : (x, y) \in M_P \implies (y, x) \notin M_P$) as we do not consider reflexive pairs of classes $(a, a)$ and want to count a pair of classes $a$, $b$ for which $a \cap b \neq \emptyset$ holds only once. With $M_P$ we now define the dependency complexity measure $DCM_{LCOM3}$ as the number of elements in $M_P$.

$$DCM_{LCOM3}(P) = |M_P| \tag{5.2}$$

The measurement value for a package $P$ ranges between 0 and the number of all possible pairs of classes of $P$. If there is no class of $P$ that shares a dependency with another class of $P$ then $M_P = \emptyset$ and $DCM_{LCOM3}(P) = 0$. If all classes of $P$ share at least one dependency with each other class in $P$ then $M_P$ contains all possible pairs of classes of $P$. This upper boundary $max_M$ can be calculated for every package $P$ as follows.

$$max_M(P) \overset{(1)}{=} (n-1) + (n-2) + ... + 1 = \frac{(n-1) * n}{2}\ (\text{with } n = |C_P|) \tag{5.3}$$

Hence, the closer $DCM_{LCOM3}(P)$ to $max_M(P)$ the more classes in $P$ share their dependencies.

### 5.1.2 Variant based on similarity measure

This variant is based on the similarity measure [6]. Contrary to the first variant, this measure does not directly count the number of elements in a subset of $P \times P$, but defines a *sim* function to calculate the similarity value for all possible pairs of classes in a Package $P$. So it differentiates between pairs of classes that have dependencies in common. Therefore, we define the set of pairs for all classes in $P$ as follows.

$$O_P \coloneqq \{\ (c_i, c_j) \mid c_i, c_j \in C_P : i < j\ \}\ (\text{with } c_1,\ c_2,\ ...,\ c_n\ \in C_P) \tag{5.4}$$

[1] *Explanation*: To get all possible pairs of classes - in which $(x, y)$ is identical with $(y, x)$, thus we only need $(x, y)$ - we take a class $a$ of $P$ and form pairs with all remaining $n-1$ classes of $C_P \setminus \{a\}$. Then we take another class $b$ of $P$ and form pairs with all remaining $n-2$ classes of $C_P \setminus \{a, b\}$. And so on and so forth.)

$O_P$ is an asymmetrical relation for the same reasons as $M_P$. However, $O_P$ is not limited to pairs of classes $a$, $b$ for which $a \cap b \neq \emptyset$ holds. Hence, $O_P$ contains all possible pairs of classes in $P$ regardless of whether the classes share dependencies. To determine the similarity value for all pairs in $O_P$ we define the $sim_P$ function as the ratio of the number of common dependencies both classes use to the number of all dependencies of both classes.

$$\begin{aligned} sim_P : \mathcal{P}(D_P) \times \mathcal{P}(D_P) \to [0,1] \text{ (with } D_P = \bigcup_{c \in C_P} D_c) \,, \\ (D_a, D_b) \mapsto \frac{D_a \cap D_b}{D_a \cup D_b} \end{aligned} \tag{5.5}$$

Note that the $sim_P$ function is 0 for all pairs of classes that share no dependencies and 1 for all pairs of classes which share all their dependencies. With the $sim_P$ function we now define the second variant of $DCM$ as the arithmetic mean of the similarity value for the pairs in $O_P$ as follows.

$$DCM_{SIM}(P) = \frac{\sum_{(a,b) \in O_P} sim(D_a, D_b)}{|O_P|} \tag{5.6}$$

Regardless of the number of classes and dependencies, $DCM_{SIM}$ ranges between 0 and 1 because the values of the sum in the counter of the fraction range between 0 and 1 (per definition of $sim$) so that the sum itself is less or equal to the number of pairs in $O_P$. Thus, the measurement value of $DCM_{SIM}$ for a package $P$ is normalized. This also means that the closer $DCM_{SIM}(P)$ to 1 the more classes in $P$ share more dependencies and that in contrast to the first variant the size of the package is not of any relevance.

### 5.1.3 Variant based on cohesion count

This variation is based on cohesion count [21]. In contrary to the other variants, it is not based on a relation over the classes of $P$. Instead, it takes a holistic approach and focuses on the dependencies that are used by at least one class of $P$. Therefore, we need to know the number of classes that use the dependency $d$ for each dependency $d \in D_P$. Thus, we define the count function $cnt_P$ to formalize the desired relation between $D_P$ and $C_P$.

$$cnt_P : D_P \to \mathbb{N} \,, \quad d \mapsto |\{\, c \mid c \in C_P : d \in D_c \,\}| \tag{5.7}$$

Note that $\forall d \in D_P : cnt_P(d) > 0$ is a result of the definition of $D_P$. With $cnt_P$ we can now define the third variant of $DCM$ as follows.

$$DCM_{CC}(P) = \frac{\sum_{d \in D_P} cnt_P(d)}{|C_P| * |D_P|} \tag{5.8}$$

$DCM_{CC}$ is normalized too and the measurement value for a package $P$ ranges between 0 and 1. This is because for all dependencies $d \in D_P$ the ratio of $cnt_P(d)$ to $|C_P|$ is less than or equal to 1 and therefore $DCM_{CC}$ is the arithmetic mean of the normalized count function for each dependency $d \in D_P$. Thus, $DCM_{CC}(P) = 0$ if $D_P = \emptyset$. Otherwise, $DCM_{CC}(P) = \frac{1}{|D_P|}$ if the classes of $P$ don't share any dependencies (i.e. $D_P = \dot{\bigcup}_{c \in C_P} D_c$) or else $DCM_{CC}(P) > \frac{1}{|D_P|}$. Furthermore, $DCM_{CC}(P) = 1$ if all dependencies of $P$ are used by all classes of $P$.

## 5.2 Package DepDegree

The proposed Package DepDegree (*P-DepDegree*) is based on the statement-level measure DepDegree [4]. The key idea of DepDegree is to derive a measurement value for the object of interest from a graph representing the object of interest in a meaningful manner. *P-DepDegree* uses this approach to derive the measurement value for a package $P$ from its transitive dependency graph $TDG_P$, a subgraph of the system's dependency graph $DG_S$ that contains all classes on which the classes of $P$ directly and indirectly depend. So, *P-DepDegree* helps us, for instance, to identify root-packages that directly or indirectly depend on all classes of the system. Consequently, we can also find base-packages that only have a few dependencies. Furthermore, we can identify core classes on which most of the classes indirectly depend by comparing the transitive dependency graphs of the packages in the system. In order to formalize *P-DepDegree* we first of all define the dependency graph $DG_S$ that represents all dependencies between the classes of $S$. For that, we consider $C_S$ the set of all classes in $S$ and define $DG_S = (C_S, E_{DG})$ with $E_{DG}$ as follows so that for all dependencies $d$ in $D_c$ for all classes $c$ in $C_S$ $(c, d)$ is an edge of $DG_S$, i.e., $c \rightarrow d \in DG_S$.

$$E_{DG} = \bigcup_{c \in C_S} \{\ (c, d) \mid d \in D_c\ \} \tag{5.9}$$

Then there exists a path $v_0 \rightarrow v_1 \rightarrow ... \rightarrow v_n \in DG_S$ between the vertices $v_0$, $v_n$ if $\forall\, 0 \leq i < n : d_i \rightarrow d_{i+1} \in DG_S$ and we consider $d_0 \rightarrow^* d_n$ the set of all edges of the path between $d_0$ and $d_n$. Based on these definitions we now define $TDG_P = (V_{TDG}, E_{TDG})$ with $V_{TDG}$ and $E_{TDG}$ as follows.

$$\begin{aligned} V_{TDG} &= C_P \cup \{\, d \in C_s \mid \exists\, c \in C_P : c \rightarrow^* d \,\} \\ E_{TDG} &= \bigcup_{c \in V_{TDG}} \{\, (c,d) \mid d \in D_c \,\} \end{aligned} \tag{5.10}$$

Having defined $TDG_P$, we now define *P-DepDegree* as the ratio of the number of edges of $TDG_P$ to the number of edges of $DG_S$.

$$\textit{P-DepDegree}(P) = \frac{|E_{TDG}|}{|E_{DG}|} \tag{5.11}$$

We use the edges instead of the vertices of the graphs to calculate *P-DepDegree* for $P$ because they represent the dependencies between the classes of $S$ and thereby better reflect the complexity of the dependency graphs $DG_S$, $TDG_P$.
*P-DepDegree* is also normalized and the measurement value for a package $P$ ranges between 0 and 1 as $|E_{TDG}| \leq |E_{DG}|$ since $E_{DG}$ is a subset of $E_{DG}$. It follows that *P-DepDegree* for $P$ is 0 if, and only if, the classes of $P$ have no dependencies so that $E_{TDG} = \emptyset$ with $|E_{TDG}| = 0$.

## 5.3 Dependency Locality Measure

The proposed Dependency Locality Measure ($DLM$) is inspired by the locality principle which states that an object is only dependent and influenced by its immediate environment. Regarding a software system this would mean that classes only interact with other classes in close proximity. This, of course, is only an idealistic goal as classes, for instance, often reference globally defined utility classes. We can, however, still use this concept to measure the complexity of a package based on the distance of its dependencies within the package tree. To do so, we first group the dependencies of $P$ by their package as follows.

$$P_D \coloneqq \{\, Q \mid Q \in P_S : C_Q \cap D_P \neq \emptyset \,\} \tag{5.12}$$

To calculate the distance between $P$ and each package in $P_D$ we now define the system's package tree $PT_S = (P_S, E_{PT})$ with $E_{PT}$ as follows.

$$E_{PT} = \{\, (Q,U) \mid Q, U \in P_S : Q \text{ is parent of } U \,\} \tag{5.13}$$

So, for each parent-child relation between two packages $Q, U \in P_S$ there is an edge $Q \rightarrow U$ in $PT_S$ and for all packages $Q \in P_S$ there exists a path $R \rightarrow^* Q$ from the root package $R$, i.e., the root of $PT_S$. Based on $PT_S$ we then define the distance function $dst$ to calculate the distance between two packages $Q, U \in PT_S$ as follows.

$$\begin{aligned} dst : P_S \times P_S \to \mathbb{N}\,, \\ (Q, U) \mapsto |R \to^* Q \triangle R \to^* U| \text{ (with } R \text{ root of } PT_S) \end{aligned} \tag{5.14}$$

Hence the distance between two packages $Q, U$ is the symmetric difference of their paths from the root package, i.e., the number of edges that the paths $R \to^* Q$ and $Q \to^* U$ do not have in common. With $dst$ and $|C_Q \cap D_P|$ as the number of dependencies of $P$ in the package $Q$ we can now define the dependency locality measure $DLM$ for $P$ as follows.

$$DLM(P) = \sum_{Q \in P_D} dst(P, Q) * |C_Q \cap D_P| \tag{5.15}$$

Note that $DLM$ is not normalized and therefore ranges between 0 and $+\infty$.

# 6 Theoretical Evaluation

For the evaluation of the measures we begin with the theoretical analysis. For this purpose we use Weyuker's properties and prove or disprove the properties for each measure. Also note that the composition operator $\triangleright$ is commutative for all measures except $DLM$.

## 6.1 Proofs

We prove each relevant property of Weyuker separately for all measures defined in this thesis. Most properties are existential, such that we give a witness for the properties (cf. [5].

### Number of Classes

For the evaluation of $NOC$ we consider the packages $P = \{a, b, c, d\}$, $Q = \{e, f\}$ and $R = \{g, h, i, j\}$ with $P \equiv Q$ and $P \not\equiv R$. Then the measurement values of $NOC$ for $P, Q, R$ are $NOC(P) = 4$, $NOC(Q) = 4$ and $NOC(R) = 2$.

**Properties 3 and 4.** With the packages $P, R$ it immediately follows that $NOC$ satisfies Property 3. Furthermore, with the packages $P, Q$ it immediately follows that $NOC$ also satisfies Property 4.

**Properties 5, 6 and 9** As the measurement value of $NOC$ for a package is equal to the number of its classes it follows per definition of the "$+$"-operator that $NOC(U) + NOC(V) = NOC(U + V)$ for any two packages $U, V$. Thus, $NOC(U) \leq NOC(U + V)$ and $NOC(V) \leq NOC(V + U)$ such that $NOC$ satisfies Property 5. Furthermore, with $NOC(U) + NOC(V) = NOC(U + V)$ it follows that $NOC$ satisfies neither Property 6 nor Property 9.

### Afferent Coupling

To disprove Property 5 and to show the existence of the properties 3, 4 and 6 for $Ca$, we consider the packages $P, Q, R, U$ and their dependencies as shown in Figure 6.1 with $P \equiv U$ and $P \not\equiv Q$. The measurement values of $Ca$ for $P, Q, R, U$ are $Ca(P) = 3$, $Ca(Q) = 3$, $Ca(R) = 0$ and $Ca(U) = 1$.

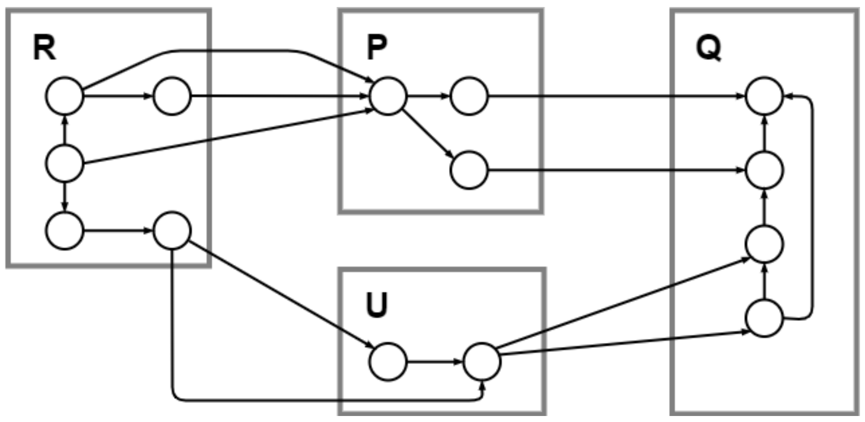


Figure 6.1: Exemplary dependency graph for the theoretical evaluation of $Ca$

**Properties 3 and 4** With the packages $P, Q$ it immediately follows that $Ca$ satisfies Property 3. Furthermore, with the packages $P, U$ it immediately follows that $Ca$ also satisfies Property 4.

**Property 5.** The composition $P + R$ with $Ca(P + R) = 0$ shows $Ca(P) > Ca(P + R)$ as no classes outside of $P + R$ depend on classes within $P + R$. Thus, $Ca$ does not satisfy this property.

**Property 6.** The compositions $P+U$ with $Ca(P+U) = 4$ and $Q+U$ with $Ca(Q+U) = 3$ have different measurement values such that $Ca(P) = Ca(Q) \wedge Ca(P + U) \neq Ca(Q + U)$ holds. Thus, $Ca$ satisfies this property.

**Property 9.** We consider the composition $V + W$ of any two packages $V, W$ for which we know that the classes depending on $V + W$ depend either on one or both packages $V, W$. Thus, $Ca(V + W)$ can not be greater than $Ca(V) + Ca(W)$ per definition of $Ca$ since a class that depends on $V$ and $W$ is only counted once for $Ca(V + W)$ but twice in $Ca(V) + Ca(W)$. Furthermore, classes in $V$ that depend on classes in $W$ and vice versa are also not considered for $Ca(V + W)$ as the classes of $V$ and $W$ are in $V + W$ so that the dependencies between $V$ and $W$ are dependencies within $V + W$. Consequently, $Ca$ does not satisfy this property.

## Efferent Coupling

To disprove Property 5 and to show the existence of the properties 3, 4 and 6 for $Ce$, we consider the packages $P, Q, R, U$ and their dependencies as shown in Figure 6.2 with

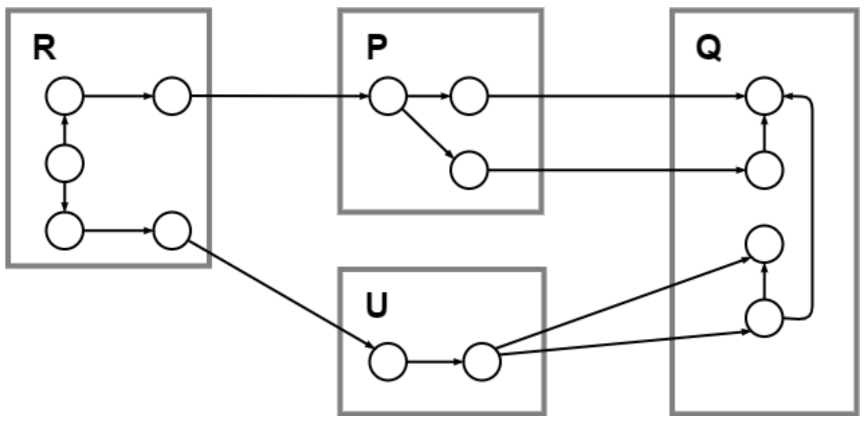


Figure 6.2: Exemplary dependency graph for the theoretical evaluation of $Ce$

$P \equiv U$ and $P \not\equiv R$. The measurement values of $Ce$ for $P, Q, R, U$ are $Ce(P) = 2$, $Ce(Q) = 0$, $Ce(R) = 2$ and $Ce(U) = 1$.

**Properties 3 and 4** With the packages $P, R$ it immediately follows that $Ce$ satisfies Property 3. Furthermore, with the packages $P, U$ it immediately follows that $Ca$ also satisfies Property 4.

**Property 5.** The composition $P + Q$ with $Ce(P + Q) = 0$ shows $Ce(P) > Ce(P + Q)$ as no classes within $P + Q$ depend on classes outside of $P + R$. Thus, $Ce$ does not satisfy this property.

**Property 6.** The compositions $P{+}U$ with $Ce(P{+}U) = 3$ and $R{+}U$ with $Ce(R{+}U) = 2$ have different measurement values such that $Ce(P) = Ce(R) \wedge Ce(P + U) \neq Ce(R + U)$ holds. Thus, $Ce$ satisfies this property.

**Property 9.** We consider the composition $V + W$ of any two packages $V, W$. Then $Ce(V + W)$ is the number of classes in $V$ that depend on classes outside of $V + W$ plus the number of classes in $W$ that depend on classes outside of $V + W$. However, classes in $V$ that only depend on classes in $W$ and vice versa are not considered for $Ce(V + W)$ as the classes in $V$ and $W$ are in $V + W$ so that the dependencies between $V$ and $W$ are dependencies within $V + W$. Thus, $Ce(V + W) \leq Ce(V) + Ce(W)$ per definition of $Ce$ such that $Ce$ does not satisfy this property.

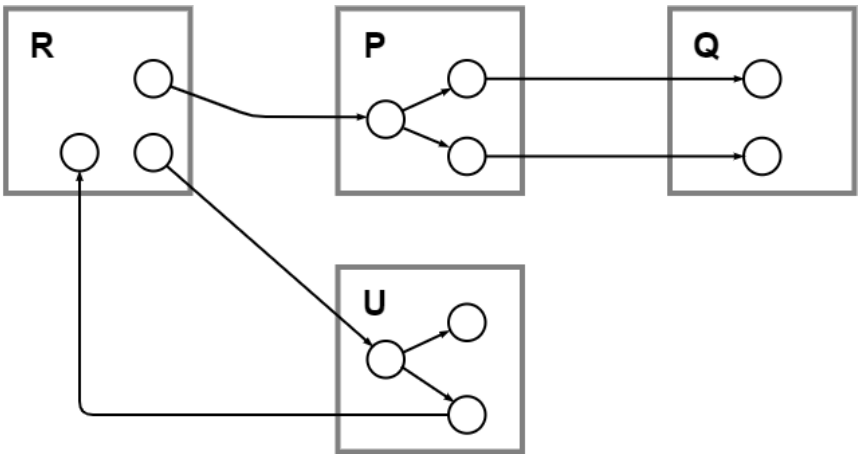


Figure 6.3: Exemplary dependency graph for the theoretical evaluation of $I$

## Instability

To disprove Property 5 and to show the existence of the properties 3, 4 and 6 for $I$, we consider the packages $P, Q, R, U$ and their dependencies as shown in Figure 6.3 with $P \equiv U$ and $P \not\equiv R$. The measurement values of $I$ for $P, Q, R, U$ are $I(P) = \frac{2}{3}$ ($Ce(P) = 2$, $Ca(P) = 1$), $I(Q) = 0$ ($Ce(Q) = 0$, $Ca(Q) = 2$), $I(R) = \frac{2}{3}$ ($Ce(R) = 2$, $Ca(R) = 1$) and $I(U) = \frac{1}{2}$ ($Ce(U) = 1$, $Ca(U) = 1$).

**Properties 3 and 4** With the packages $P, R$ it immediately follows that $I$ satisfies Property 3. Furthermore, with the packages $P, U$ it immediately follows that $I$ also satisfies Property 4.

**Property 5.** The composition $P + Q$ with $I(P + Q) = 0$ as a result of $Ce(P + Q) = 0$ shows $I(P) > I(P + Q)$ as no classes within $P + Q$ depend on classes outside of $P + R$. Thus, $I$ does not satisfy this property.

**Property 6.** The compositions $P+U$ with $I(P+U) = \frac{3}{4}$ ($Ce(P+U) = 3$, $Ca(P+U) = 2$) and $R+U$ with $I(R+U) = 1$ ($Ce(R+U) = 1$, $Ca(R, U) = 0$) have different measurement values such that $I(P) = I(R) \wedge Ce(P + U) \neq Ce(R + U)$ holds. Thus, $I$ satisfies this property.

**Property 9.** Unfortunately, we were neither able to disprove this property for $I$ nor able to find an example to show the existence of this property for $I$.

## DCM based on LCOM3

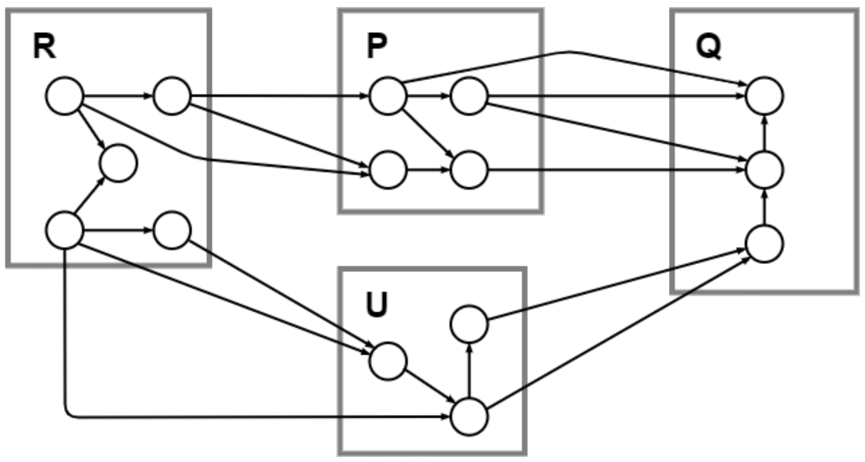


Figure 6.4: Exemplary dependency graph for the theoretical evaluation of $DCM_{LCOM3}$

To disprove property 5 and to show the existence of the properties 3, 4 and 6 for $DCM_{LCOM3}$ we, consider the packages $P, Q, R, U$ and their dependencies as shown in Figure 6.4 with $P \equiv U$ and $P \not\equiv R$. The measurement values of $DCM_{LCOM3}$ for $P, Q, R, U$ are $DCM_{LCOM3}(P) = 3$, $DCM_{LCOM3}(Q) = 0$, $DCM_{LCOM3}(R) = 3$ and $DCM_{LCOM3}(U) = 1$.

**Properties 3 and 4** With the packages $P, R$ it immediately follows that $DCM_{LCOM3}$ satisfies Property 3. Furthermore, with the packages $P, U$ it immediately follows that $DCM_{LCOM3}$ also satisfies Property 4.

**Property 5.** This property must be modified as high cohesion is a desirable criterion and Weyuker's properties are developed to evaluate complexity measures. Here, high cohesion means lower complexity and vice versa [6]. Thus, we have to reverse the comparative sign in the property's definition. Then with the composition $Q + P$ with $DCM_{LCOM3}(Q + P) = 7$ the equation $DCM(Q) > DCM(Q + P)$ does not hold such that $DCM_{LCOM3}$ does not satisfy this property.

**Property 6.** The compositions $P + Q$ with $DCM_{LCOM3}(P + Q) = 7$ and $R + Q$ with $DCM_{LCOM3}(R+Q) = 3$ have different measurement values such that $DCM_{LCOM3}(P) = DCM_{LCOM3}(R) \wedge DCM_{LCOM3}(P+Q) \neq DCM_{LCOM3}(R+Q)$ holds. Thus, $DCM_{LCOM3}$ satisfies this property.

**Property 9.** For the same reasons as for Property 5, we need to reverse the comparative sign in the property's definition. We consider the composition $V+W$ of any two packages $V, W$. Then all pairs of classes in $V$ and $W$ that share dependencies are also in $V+W$ per definition of the "+"-operator. Furthermore, there could be new pairs of classes in $V+W$ that share dependencies. Thus, the formula $DCM_{LCOM3}(V)+DCM_{LCOM3}(W) \leq DCM1(V+W)$ is true for all packages $V, W$ such that $DCM_{LCOM3}$ does not satisfy this property.

## DCM based on similarity measure

The measure $DCM_{SIM}$ is based on, has already been analyzed with Weyuker's Properties by other authors [6]. It has been shown that it satisfies the properties 1,2,3,4 and 6. Furthermore, it has also been shown that it does not satisfy properties 5 and 9. Since we have accurately transferred the definition of this measure to define $DCM_{SIM}$ as a cohesion measure on package-level, this also holds for $DCM_{SIM}$.

## DCM based on cohesion count

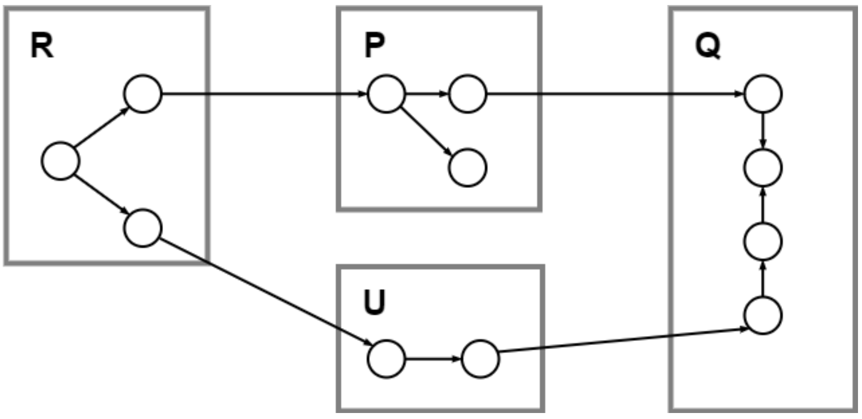


Figure 6.5: Exemplary dependency graph for the theoretical evaluation of $DCM_{CC}$

To show the existence of the properties 3, 4, 6 and 9 for $DCM_{CC}$, we consider the packages $P, Q, R, U$ and their dependencies as shown in Figure 6.5 with $P \equiv U$ and $P \not\equiv R$. The measurement values of $DCM_{CC}$ for $P, Q, R, U$ are $DCM_{CC}(P) = \frac{1+1+1}{3*3} = \frac{1}{3}$, $DCM_{CC}(Q) = \frac{2+1}{4*3} = \frac{3}{8}$, $DCM_{CC}(R) = \frac{1+1+1+1}{3*4} = \frac{1}{3}$ and $DCM_{CC}(U) = \frac{1+1}{2*2} = \frac{1}{2}$.

**Properties 3 and 4** With the packages $P, R$ it immediately follows that $DCM_{CC}$ satisfies Property 3. Furthermore, with the packages $P, U$ it immediately follows that $DCM_{CC}$ also satisfies Property 4.

**Property 5.** This property must be modified as high cohesion is a desirable criterion and Weyuker's properties are developed to evaluate complexity measures. Here, high cohesion means lower complexity and vice versa [6]. Thus, we have to reverse the comparative sign in the property's definition. We consider the composition $V + W$ of any two packages $V, W$. We know that the composition does not result in new dependencies such that the number of dependencies in $V + W$ is equal to the sum of the dependencies of $V, W$. Furthermore, the denominator of the formula of $DCM_{CC}$ increases for $V + W$ as the number of classes of $V + W$ is the sum of the number of classes of $V, W$. Thus, it follows that $DCM_{CC}(V) \geq DCM_{CC}(V+W) \wedge DCM_{CC}(W) \geq DCM_{CC}(V+W)$ holds for $V, W$ such that $DCM_{CC}$ satisfies this property.

**Property 6.** The compositions $P + Q$ with $DCM_{CC}(P + Q) = \frac{1+1+2+1}{7*4} = \frac{5}{28}$ and $R+Q$ with $DCM_{CC}(R+Q) = \frac{1+1+1+1+1+2+1}{7*6} = \frac{1}{6}$ have different measurement values such that $DCM_{CC}(P) = DCM_{CC}(R) \wedge DCM_{CC}(P + Q) \neq DCM_{CC}(R + Q)$ holds. Thus, $DCM_{CC}$ satisfies this property.

**Property 9.** For the same reasons as for Property 5, we need to reverse the comparative sign in the property's definition. Then for the composition $R+Q$ with $DCM_{CC}(R+Q) = \frac{1}{6}$ the equation $DCM_{CC}(R) + DCM_{CC}(Q) > DCM_{CC}(R + Q)$ holds. Thus, $DCM_{CC}$ satisfies this property.

### Package DepDegree

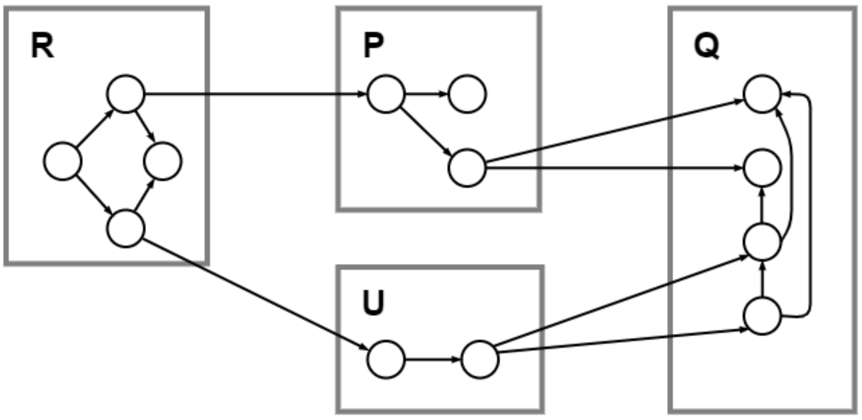


Figure 6.6: Exemplary dependency graph for the theoretical evaluation of *P-DepDegree*

Although DepDegree has already been evaluated using Weyuker's Properties, we need to reevaluate Weyuker's Properties for *P-DepDegree* since it does not use the same graph as DepDegree but instead is based on the dependency graph of a system. To show the existence of the properties 3, 4 and 6 for *P-DepDegree*, we consider the packages $P, Q, R, U$ and their dependencies as shown in Figure 6.6 with $P \equiv U$ and $P \not\equiv Q$. The measurement values of *P-DepDegree* for $P, Q, R, U$ are $\textit{P-DepDegree}(P) = \frac{4}{17}$, $\textit{P-DepDegree}(Q) = \frac{4}{17}$, $\textit{P-DepDegree}(R) = \frac{17}{17}$ and $\textit{P-DepDegree}(U) = \frac{7}{17}$

**Properties 3 and 4** With the packages $P, Q$ it immediately follows that *P-DepDegree* satisfies Property 3. Furthermore, with the packages $P, U$ it immediately follows that *P-DepDegree* also satisfies Property 4.

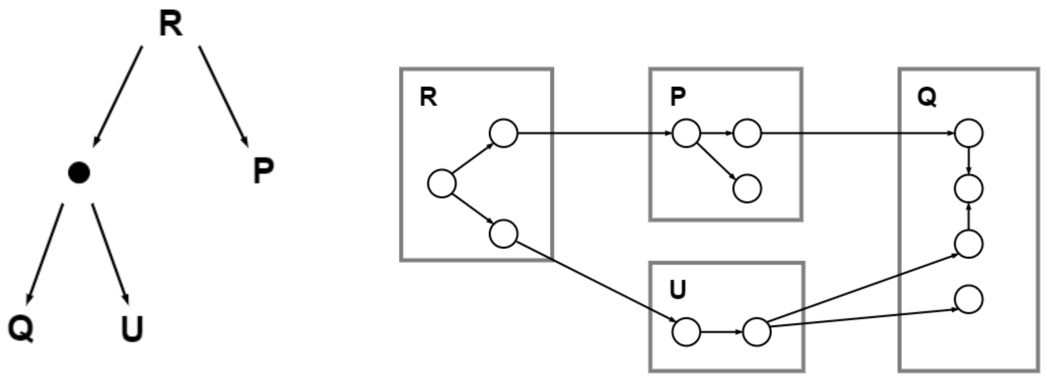
**Property 5.** We consider the composition $V + W$ of any two packages $V, W$. Then per definition of the "$+$"-operator all classes of $V$ and $W$ are also in $V + W$ and it follows that the edges of $TDG_V$ and $TDG_W$ are also edges of $TDG_{V+W}$. Thus, $\textit{P-DepDegree}(V) \leq \textit{P-DepDegree}(V + W) \wedge \textit{P-DepDegree}(W) \leq \textit{P-DepDegree}(V + W)$ such that *P-DepDegree* satisfies this property.

**Property 6.** The compositions $P + U$ with $\textit{P-DepDegree}(P + U) = \frac{11}{17}$ and $Q + U$ with $\textit{P-DepDegree}(Q+U) = \frac{7}{17}$ have different measurement values such that $\textit{P-DepDegree}(P) = \textit{P-DepDegree}(Q) \wedge \textit{P-DepDegree}(P+U) \neq \textit{P-DepDegree}(Q+U)$ holds. Thus, *P-DepDegree* satisfies this property.

**Property 9.** We consider the composition $V + W$ of any two packages $V, W$. We know that the edges of $TDG_V$ and $TDG_W$ are also edges of $TDG_{V+W}$. Furthermore, we know that $TDG_V$ and $TDG_W$ are complete. Thus, it follows per definition of the "$+$"-operator that each edge of $TDG_{V+W}$ is either an edge of one or both TDGs of $V, W$. The latter is the case when $TDG_V$ and $TDG_W$ overlap, i.e., the classes in $V$ and $W$ indirectly depend on the same classes. Consequently, it follows that $\textit{P-DepDegree}(V) + \textit{P-DepDegree}(W) \geq \textit{P-DepDegree}(V + W)$ such that *P-DepDegree* does not satisfy this property.

## Dependency Locality Measure

For the evaluation of $DLM$ we consider the packages $P, Q, R, U$ and their dependencies as well as their package tree as shown in Figure 6.7 with $P \equiv U$ and $P \not\equiv R$. Then the measurement values of $DLM$ for $P, Q, R, U$ are $DLM(P) = 3$, $DLM(Q) = 0$, $DLM(R) = 3$ and $DLM(U) = 4$. Furthermore, the distance values for the packages are $dst(R, P) = 1$, $dst(R, Q) = dst(R, U) = 2$, $dst(P, Q) = dst(P, U) = 3$ and $dst(Q, U) = 2$.



(a) Package tree (b) Dependency graph

Figure 6.7: Exemplary dependency graph and package tree for the theoretical evaluation of *DLM*

**Properties 3 and 4** With the packages $P, R$ it immediately follows that $DLM$ satisfies Property 3. Furthermore, with the packages $P, U$ it immediately follows that $DLM$ also satisfies Property 4.

**Property 5.** For the composition $P \triangleright Q$ with $DLM(P \triangleright Q) = 0$ the equation $DLM(P) < DLM(P \triangleright Q)$ does not hold. Thus, $DLM$ does not satisfy this property.

**Property 6.** The compositions $R \triangleright Q$ with $DLM(R \triangleright Q) = 5$ and $P \triangleright Q$ with $DLM(P \triangleright Q) = 0$ have different measurement value. Furthermore, the compositions $Q \triangleright R$ with $DLM(Q \triangleright R) = 3$ and $Q \triangleright P$ with $DLM(Q \triangleright P) = 0$ have also different measurement value. Thus, $DLM$ satisfies this property.

**Property 9.** For the composition $R \triangleright Q$ the equation $DLM(R) + DLM(Q) < DLM(R \triangleright Q)$ holds. Thus, $DLM$ satisfies this property.

## 6.2 Summary

Table 6.1 shows that only $DCM_{CC}$ satisfies all Weyuker's properties. This is because the composition of two packages results only for $DCM_{CC}$ in a disproportionately lower measurement value. In contrast, *P-DepDegree* satisfies Property 5 but not Property 9, because its measurement values are not disproportional in regard to the composition of two packages. Regarding Property 9, the same applies to the measures *NOC*, *Ca*, *Ce*,

| Measures | 1 | 2 | 3 | 4 | 5 | 6 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| $NOC$ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ |
| $Ca$ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✗ |
| $Ce$ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✗ |
| $I$ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ? |
| $DCM_{LCOM3}$ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✗ |
| $DCM_{SIM}$ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✗ |
| $DCM_{CC}$ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| *P-DepDegree* | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |
| $DLM$ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ |

Table 6.1: Overview of the Weyuker's Properties fulfilled by the measures

$I$, $DCM_{LCOM3}$ and $DCM_{SIM}$. Similar to $DCM_{CC}$, however, the measurement values of $DLM$ are disproportional for some compositions of two packages, but there are also exceptions such that the complexity of a composition of two packages is lower than the measurement values of the packages, which is why $DLM$ does not satisfy Property 5. This is because the distance of dependencies within a composition is 0. Hence, the composition of two packages that heavily depend on each other has most likely a lower measurement value than that of the individual packages.
The measures $Ca$, $Ce$, $I$, $DCM_{LCOM3}$ and $DCM_{SIM}$ also do not satisfy Property 5. $Ce$ and $Ca$ focus on the outgoing and incoming dependencies of a package. Thus, dependencies within a package are not considered such that the measurement value of a composition of packages that depend on each other is lower than of the individual packages. Since $I$ is based on $Ce$ and $Ca$, $I$ does also not satisfy Property 5 for the same reasons as $Ce$ and $Ca$. $DCM_{LCOM3}$ in turn does not satisfy Property 5 because it is not normalized such that the cohesion value for the composition of two packages is higher than or equal to that of the individual packages. $DCM_{SIM}$ does not satisfy Property 5 because of some exceptions where the composition of two packages contains more pairs of classes that share their dependencies than the packages themselves.
Lastly, $NOC$ does not satisfy Property 6 because it is a counting measure. As such the measurement values of $NOC$ increase uniformly. Hence, the measurement value of the composition of two packages is equal to the sum of the measurement values of the packages.

# 7 Practical Evaluation

Following the theoretical analysis of the defined measures, we conduct a practical evaluation of the measures on the example of the open-source project CPAchecker (v.1.9.1), a configurable software-verification platform for C and Java programs, to identify correlations between the defined measures and evaluate the practical applicability of the proposed measures. The code base of CPAchecker is written in Java and consists of 230 packages with 3596 classes including interfaces, abstract and static classes of which 1440 are nested classes. Furthermore, we also found 115 test classes, i.e., classes whose name ends on "Test" and 1015 external references to classes which are not in the domain "org.sosy_lab.cpachecker" of the project. We listed these classes and references separately because we focus for the practical evaluation on classes which are in the domain and also relevant for the functionality of CPAchecker. Thus, we only consider dependencies between classes of CPAchecker which are not test classes and ignore dependencies to classes of external libraries.

## 7.1 Implementation

Since we needed to implement the measures for practical evaluation, we decided to develop our own prototypical measurement tool Jade for maximum flexibility instead of using an existing software-analysis platform. However, we used the command line tool jdeps [18], which is part of the JDK, to generate the dependency graph for CPAchecker. It uses the Java class dependency analyzer to generate and store the dependency graph of a Java project including external dependencies as a directed graph in a dot file.
Jade itself was developed with Python and consists of several scripts, the code can be found online [(1)]. The first script *depgraph.py* parses the dependency graph generated by jdeps from the dot file, removes the vertices and edges from the graph which should be ignored according to the restrictions mentioned above and stores the refined graph as json file under "./data/depgraph.json". The second script *measures.py* reads the refined graph from the json-file, calculates the measurement values of the defined measures for all packages and stores the calculated values for each measure in a separate json file under

[1]Github code repository for the measurement tool Jade: `https://github.com/simon-lund/jade/tree/304a35e8cad74b9bbfca71ff67782ef790b1e777`

"./data/<measure>.json". The third script *report.py* reads the json files of all measures as well as the json file storing the dependency graph and generates stores a report in markdown, the colored graphs of the different measures as shown below as well as a correlation matrix for all measures defined in this thesis.
The scripts can be called separately, however, it is recommended to use *main.py* which runs the scripts consecutively. Note that the scripts *depgraph.py* and *report.py* depend on *config.py* which specifies the domain of the software project, the location of the dot file generated by jdeps and a deny list containing classes that should also be excluded from the dependency graph of the software system under consideration. Furthermore, Jade also includes a test suite with several smaller, manually crafted dependency graphs for which we calculated the different measures by hand to test the script *measures.py*.

## 7.2 Graph Analysis

The dependency graph of CPAchecker has 3596 vertices according to the number of classes in CPAchecker and 32 183 edges, i.e, there are 32 183 dependencies defined between the classes of CPAchecker. The json file with the dependency graph of CPAchecker, the data sets for the measures as well as the generated graphs can also be found online [(2)] Based on this data, we will analyze the distribution of the measurement values for the different measures and evaluate the practical applicability of the proposed measures. Note that the graphs, if not stated otherwise, are sorted by their y-values so that the y-values of the different graphs do not necessarily belong to one and the same package.

### Number of Classes

The graph for the measurement values of *NOC* (cf. Figure 7.1) shows that the majority of the packages of CPAchecker, i.e, 197 packages, contain less than 30 classes. This fact is underlined by the arithmetic mean of the measurement values of *NOC*, i.e., the ratio of the number of all classes to the number of all packages of CPAchecker which is $\frac{3596}{230} \approx 16$. In the respective data set for *NOC* we found 158 packages with less than 16 classes including 17 packages containing less than 3 classes. In contrast, Table 7.1 shows the five packages of CPAchecker with the highest *NOC* measurement values. Thus, the package "cpa.automaton" is with 132 classes by far the largest package.

[2]Github data repository for CPAChecker: `https://github.com/simon-lund/cpachecker-data/tree/9e3932a617ca39999536d16847522cf11f1db759`

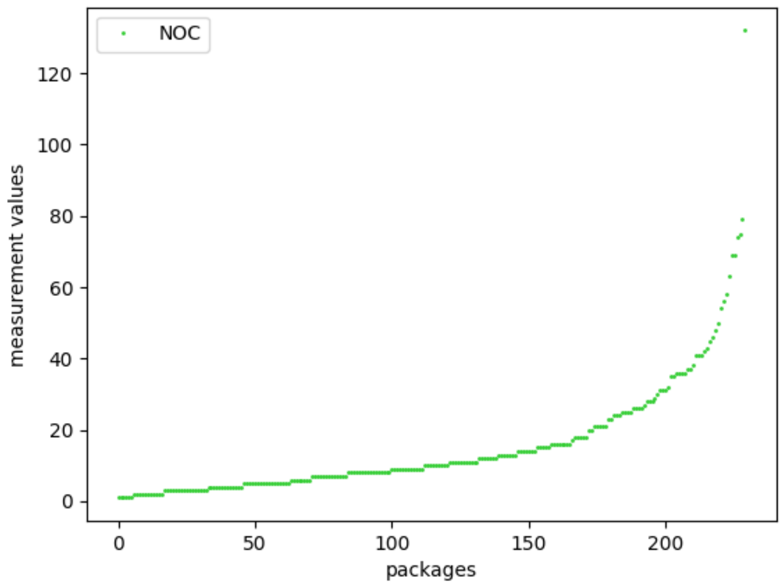


Figure 7.1: Graph for the measurement values of $NOC$

## Coupling Measures

If we take a look at the graphs of $Ce$ and $Ca$ as shown in Figure 7.2, we notice that the range of the measurement values of $Ca$ is by far larger than the one of $Ce$. This is because the maximum value of $Ce$ for a package $P$ is $NOC(P)$, i.e., the number of classes in $P$, while the maximum value of $Ca$ for $P$ is the number of classes in the system (in this case 3596 minus $NOC(P)$. Furthermore, in both cases most packages of CPAchecker have a relatively low value. However, this effect is more extreme for $Ca$ than for $Ce$, because of the larger range of the measurement values of $Ca$. There are 203 packages

| Rank | Package | $NOC$ |
|---|---|---|
| 1 | cpa.automaton | 132 |
| 2 | util | 79 |
| 3 | cpa.invariants | 75 |
| 4 | cpa.predicate | 74 |
| 5 | core.algorithm.bmc | 69 |

Table 7.1: The five packages with the highest $NOC$ values

whose $Ce$ value is lower than 25 and 191 packages with a $Ca$ value lower than 45. Thus, neither the measurement values of $Ce$ nor of $Ca$ are uniformly distributed.

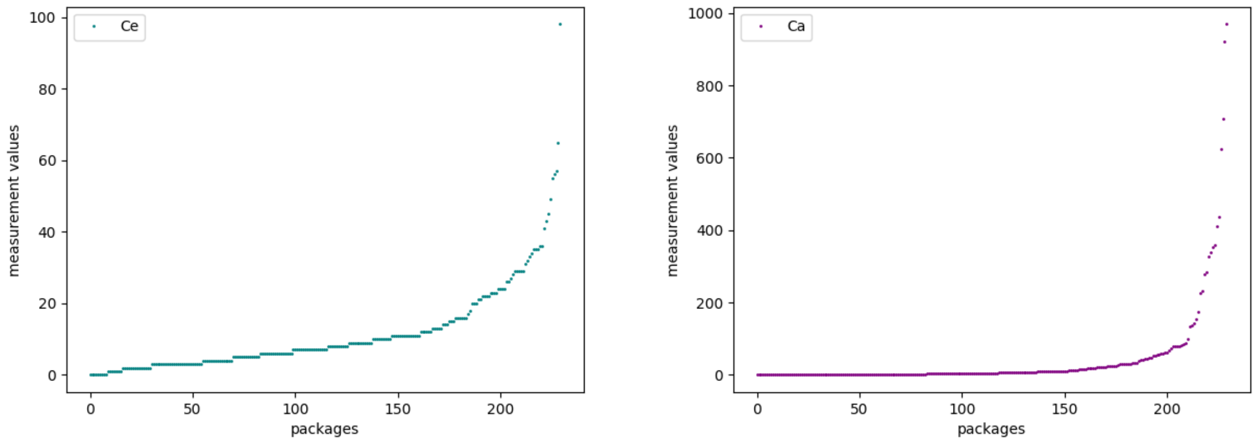


Figure 7.2: Graphs for the measurement values of $Ce$ and $Ca$

Now, if we take a look at the five packages of CPAchecker with the highest $Ca$ values and their corresponding $Ce$ and $I$ measurement values as shown in Table 7.2, we notice that the $Ce$ values of these core packages are relatively small compared to the $Ca$ values resulting in a very low $I$ value for the packages. However, the $Ce$ values of these packages are in the upper third of the $Ce$ values for the packages of CPAchecker.

| Rank | Package | $Ca$ | $Ce$ | $I$ |
|---|---|---|---|---|
| 1 | cfa.model | 969 | 14 | 0.014 |
| 2 | core.interfaces | 921 | 23 | 0.024 |
| 3 | exceptions | 708 | 11 | 0.015 |
| 4 | util | 623 | 55 | 0.081 |
| 5 | cfa.ast.c | 438 | 56 | 0.113 |

Table 7.2: The five packages with the highest $Ca$ values and corresponding $Ce$ and $I$ values

Furthermore, if we look at the five packages of CPAchecker with the highest $Ce$ values and their corresponding $Ca$ and $I$ values (cf. Table 7.3), we notice that the $Ca$ values of these package are in the upper fifth of the $Ca$ values. Hence, there is a weak correlation between the $Ce$ and $Ca$ values for the packages of CPAchecker.

| Rank | Package | $Ce$ | $Ca$ | $I$ |
|---|---|---|---|---|
| 1 | cpa.automaton | 98 | 41 | 0.705 |
| 2 | cpa.predicate | 65 | 78 | 0.455 |
| 3 | core.algorithm.bmc | 57 | 30 | 0.655 |
| 4 | cfa.ast.c | 56 | 438 | 0.113 |
| 5 | util | 55 | 623 | 0.081 |

Table 7.3: The five packages with the highest $Ce$ values and corresponding $Ca$ and $I$ values

Having discussed $Ca$ and $Ce$, we now take a closer look on the graph for the normalized measure $I$ as shown in Figure 7.3. In contrast to $Ce$ and $Ca$, the measurement values of $I$ are very uniformly distributed. So, despite the non-uniformity of the measurement values of $Ce$ and $Ca$ for the packages of CPAchecker the stability of the packages varies considerably.

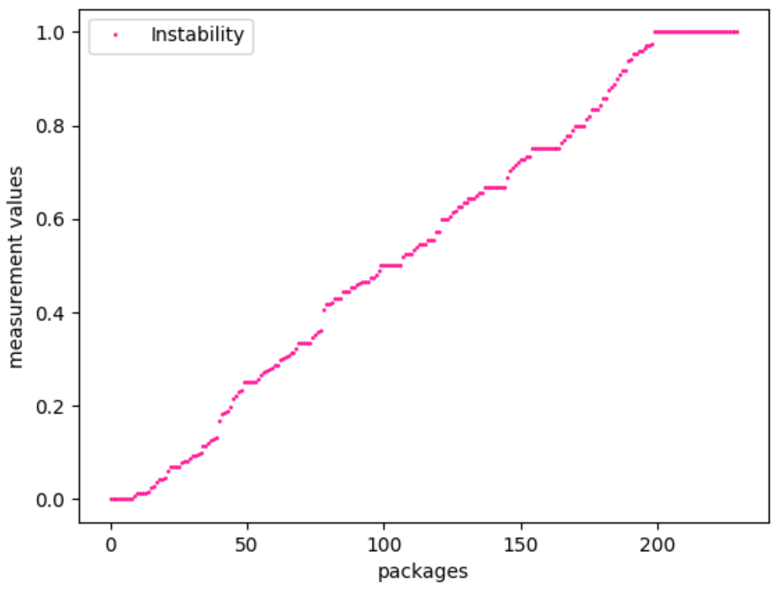


Figure 7.3: Graph for the measurement values of $I$

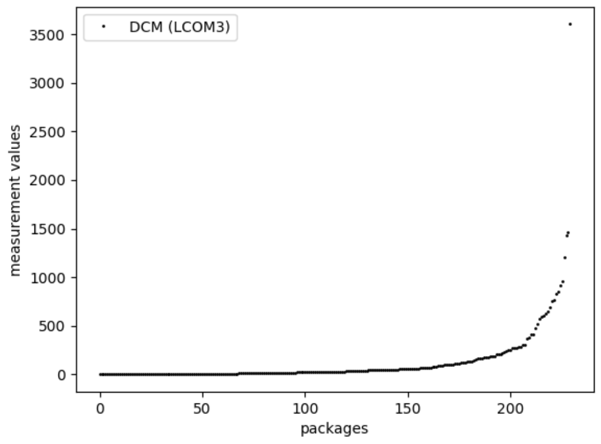
However, the graph has also 7 bigger tiers that stand out particularly; one with y-value 0 at the beginning and one with y-value 1 at the end and five in between. Especially interesting for us is the tier at the end of the graph which implies that there are packages with an $I$ measurement value of 1. Hence, these packages have a $Ca$ value of 0. According to the data sets of $I$ and $Ca$, there are 31 of such packages in CPAchecker. However, an in-depth analysis to investigate the reasons that these packages are not used by other packages of CPAchecker is beyond the scope of this thesis.

### Dependency Cohesion Measures

Figure 7.4 shows the graphs of the three variants of $DCM$, $DCM_{LCOM3}$ and the two normalized measures $DCM_{SIM}$, $DCM_{CC}$. In comparison with the other unnormalized measures $DCM_{LCOM3}$ has by far the largest measurement values with a maximum value of 3605 for the package "cpa.automaton" which is also the package with the most classes. As a consequence, the measurement values are poorly distributed because the majority of the packages, i.e., 206 packages, have a measurement value below 300 whereby the arithmetic mean of the measurement values is 131. However, this does not mean that most packages of CPAchecker have low dependency cohesion because for the interpretation of $DCM_{LCOM3}$ one should take the number of the packages into account which is 16 on average.
The normalized measures are more uniformly distributed with $DCM_{CC}$ being slightly better distributed than $DCM_{SIM}$ where the majority of the packages, i.e., 197 packages, have a measurement value below 0.25. Also note that the measurement values of $DCM_{CC}$ range between 0 and 1 while the measurement values of $DCM_{SIM}$ only range between 0 and 0.55. Thus, according to $DCM_{CC}$ there are packages that are fully cohesive while on other hand according to $DCM_{SIM}$ there are no such packages.

Furthermore, we discovered that the packages with a $DCM_{CC}$ measurement value of 1, i.e., the packages with the highest measurement values all have a $DCM_{SIM}$ measurement value of 0. This is the case because all of these packages have only one class so that there are no pairs of classes for which a similarity value could be determined resulting in a $DCM_{SIM}$ measurement value of 0. This explains the tiers at the end of the graph of $DCM_{CC}$ and at the beginning of the graph of $DCM_{SIM}$. In addition, we listed in Table 7.4 the five packages with the highest value of $DCM_{SIM}$ and their corresponding $DCM_{CC}$ value which are, except for the value of the fourth package, in the upper tenth of the measurement values.

However, the second and third package show that $DCM_{SIM}$ and $DCM_{CC}$ measure different aspects in regard to the dependency cohesion of a package. By counting the number of classes that use the dependency for all dependencies of the second package, we found that only 5 of the 43 different dependencies of the package are used by almost all classes of the package. This explains the low value for $DCM_{CC}$ which measures the overall dependency cohesion of the package. On the other hand, the package has a $DCM_{LCOM3}$ value of 410, i.e., 410 of 465 pairs of classes share at least one dependency. Furthermore,



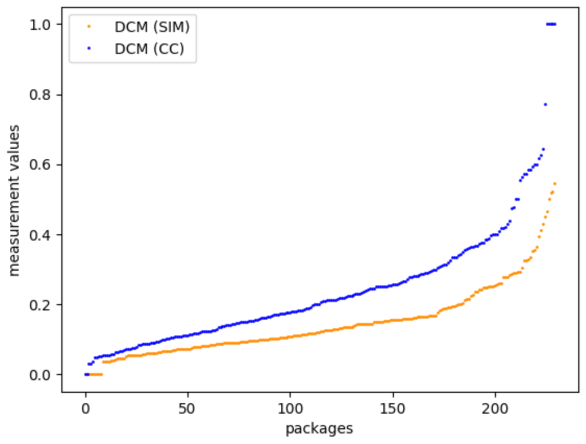


Figure 7.4: Graphs for the measurement values of $DCM_{LCOM3}$, $DCM_{SIM}$ and $DCM_{CC}$

| Rank | Package | $DCM_{SIM}$ | $DCM_{CC}$ |
|---|---|---|---|
| 1 | cpa.value.symbolic.refiner.interpolant | 0.544 | 0.773 |
| 2 | cpa.value.symbolic.type | 0.523 | 0.165 |
| 3 | cpa.invariants.operators.bitvector | 0.521 | 0.163 |
| 4 | cpa.pointer2.util | 0.5 | 0.563 |
| 5 | cpa.octagon.coefficients | 0.467 | 0.617 |

Table 7.4: The five packages with the highest $DCM_{SIM}$ values and corresponding $DCM_{CC}$ values

each class of the package has only 7 dependencies on average so that even a low number of common dependencies of two classes results in a comparatively high similarity value. Both points together explain the relatively high measurement value for $DCM_{SIM}$. From this follows that $DCM_{CC}$ is better suited than $DCM_{SIM}$ to measure the dependency cohesion of the package as it uses a more holistic approach. Also, $DCM_{CC}$ is more useful than $DCM_{LCOM3}$ because it allows better comparison and interpretation of the measurement values than $DCM_{LCOM3}$. This is because $DCM_{LCOM3}$ only counts the pairs of classes of a package that share dependencies. Thus, it does not properly reflect the dependency cohesion for a package and we also need the $NOC$ value of a package for comparison and a more comprehensive interpretation of its $DCM_{LCOM3}$ value. Therefore, only $DCM_{CC}$ meets our expectation for a dependency cohesion measure and we recommend to use $DCM_{CC}$ as the measure of choice over $DCM_{LCOM3}$ and $DCM_{SIM}$.

## Package DepDegree

In contrast to the other normalized measures, the graph for *P-DepDegree* as shown in Figure 7.5 stands out because the measurement values of *P-DepDegree* are clustered which results in a jump of the measure's graph. We found only 21 packages in the corresponding data set with a measurement value close to 0 and therefore lower than 0.7. Thus, the majority of the packages have a measurement value slightly higher than 0.71.

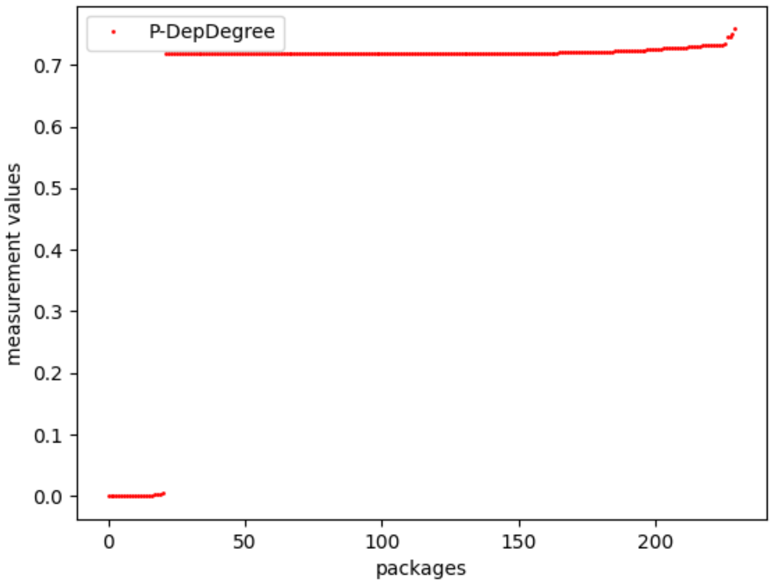


Figure 7.5: Graph for the measurement values of *P-DepDegree*

This is also reflected in the highest values of *P-DepDegree* (cf. Table 7.5) of which none is higher than 0.76. However, most packages have a *P-DepDegree* value between 0.71 and 0.73 whereby 95 packages have a measurement value of 0.718. Thus, the data set as a whole implies that the transitive dependency graphs for 209 packages each cover more than 71% of the system's dependency graph.

Thus, we compared the transitive dependency graphs of the packages with a *P-DepDegree* measurement value greater than 0.71. With this limitation we were able to find a subgraph consisting of 2646 classes which all of the transitive dependency graphs share. A svg and a corresponding dot file of this core dependency graph can be found online in the data repository for CPAchecker (> core_depgraph). The analysis of this graph, is beyond the scope of this thesis. Although we expected a more balanced distribution of the

| Rank | Package | *P-DepDegree* |
|---|---|---|
| 1 | cpa.value.symbolic.refiner.delegation | 0.758 |
| 2 | cpa.value.symbolic.refiner | 0.751 |
| 3 | cpa.usage.refinement | 0.747 |
| 4 | cpa.value.refiner | 0.747 |
| 5 | cfa.parser.eclipse.c | 0.734 |

Table 7.5: The five packages with the highest *P-DepDegree* values

measurement values, the extreme distribution of the data emphasized the fact that *P-DepDegree* is a good indicator to identify base-packages and core classes on which most of the classes and packages of a system depend.

## Dependency Locality Measure

Last, we take a look at the graph of $DLM$ as shown in Figure 7.6 which describes a flatter curve in comparison to the graphs of $Ce$, $Ca$, $DCM_{LCOM3}$ and $NOC$. Therefore in contrast to the other unnormalized measures, there is no comparatively small value such that the majority of the measurement values of $DLM$ are below this value. Hence, the measurement values of $DLM$ are more uniformly distributed over the range of 0 to 850 which is similar to the range of the measurement values of $Ca$.

In the corresponding data set we found 9 packages whose measurement value of $DLM$ is 0. These packages have per definition of $DLM$ also a $Ce$ value of 0. In addition, Table 7.6 shows the five packages with the highest $DLM$ measurement values and corresponding $NOC$ and $Ce$ values. The differences between these values show the impact of the locality of the dependencies of a package. Except for the second and third packages, the values of $NOC$ an $Ce$ are smaller values. Thus, should $DLM$ be considered for the evaluation of software systems as it sets the dependencies of a package in relation to the package tree of a software system and therefore provides additional, useful insights with regard to the location of dependencies within a system.

| Rank | Package | $DLM$ | $NOC$ | $Ce$ |
|---|---|---|---|---|
| 1 | cpa.value | 849 | 26 | 23 |
| 2 | cpa.predicate | 787 | 74 | 65 |
| 3 | cpa.automaton | 715 | 132 | 98 |
| 4 | cpa.smg | 668 | 46 | 34 |
| 5 | cpa.invariants | 646 | 75 | 43 |

Table 7.6: The five packages with the highest $DLM$ values

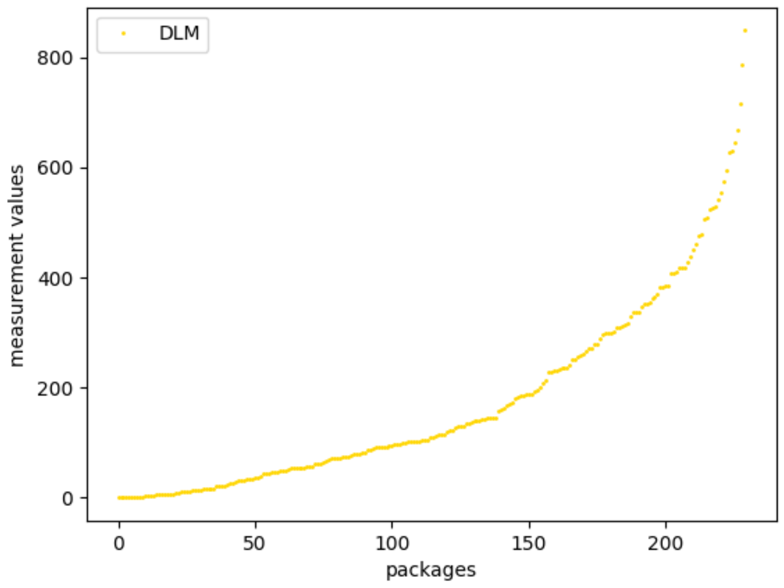


Figure 7.6: Graph for the measurement values of $DLM$

## 7.3 Correlations

Figure 7.7 shows the correlation matrix[3] for all measures defined in this thesis based on their measurement values for the packages of CPAchecker, calculated using Jade. According to the matrix, a strong correlation exists between the measures $NOC$ and $Ce$ with a correlation value of 0.94. Thus, the graphs of $NOC$ and $Ce$ as shown in Figure 7.8 are very similar to each other whereby the $NOC$ for package is the upper boundary for its $Ce$ value. Furthermore, there is also a strong correlation between $DCM_{LCOM3}$ and $NOC$ with a value of 0.89. The reason for this correlation is that the more classes a package has, the higher the probability of pairs of classes sharing dependencies, which results in a higher $DCM_{LCOM3}$ value for packages with a higher $NOC$ value and a lower $DCM_{LCOM3}$ value for packages with a lower $NOC$ value. Also, the value of $DCM_{LCOM3}$ is limited by the number of classes, i.e., the $NOC$ value of a package such that there are not disproportional outliers regarding the data set of $DCM_{LCOM3}$ for packages with a low number of classes. In addition, there is also a strong correlation between $DCM_{LCOM3}$ and $Ce$ with a value of 0.86 because of the strong correlation between $NOC$ and $Ce$. Beyond that, there are also weak correlations between the measures $Ce$ and $DLM$ with

[3]Used method of correlation: Pearson

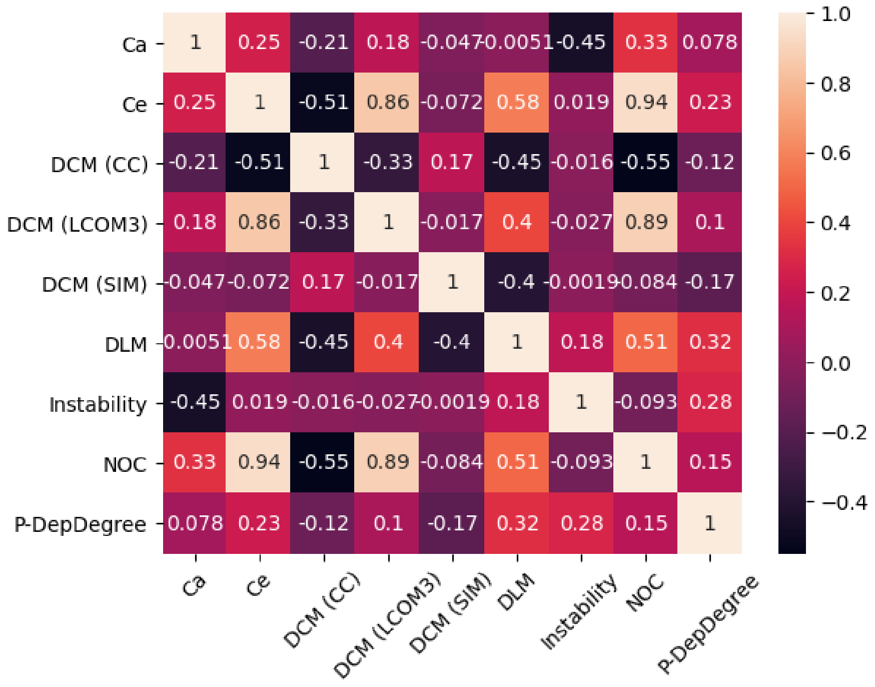


Figure 7.7: Correlation matrix regarding all defined measures

a value of 0.58 as well as between $DCM_{CC}$ and $NOC$ with a negative value of -0.55. $Ce$ and $DLM$ correlate with each other because $DLM$ is, among other things, based on the number of dependencies in a package and the more classes of a package depend on classes outside the package, the higher the number of dependencies of package. The negative correlation between $DCM_{CC}$ and $NOC$ exists because $DCM_{CC}$ is defined as fraction whereby the denominator is defined as $|C_P| * |D_P|$, i.e., $NOC(P) * |D_P|$ for a package $P$. Thus, the larger the number of classes for package, the lower is its $DCM_{CC}$ value. Also, because of the strong correlation between $NOC$ and $Ce$ there are additional weak correlations between $DLM$ and $NOC$ with a value of 0.51 as well as between $DCM_{CC}$ and $Ce$ with a negative value of -0.51.
As you can see, there are also correlations with a value lower than $|0.5|$ between some measures. Because we consider them as very weak relations between the defined measures, we will not discuss them in more detail. Note, however, that these correlations may be stronger for other software projects. This is to be evaluated by further research.

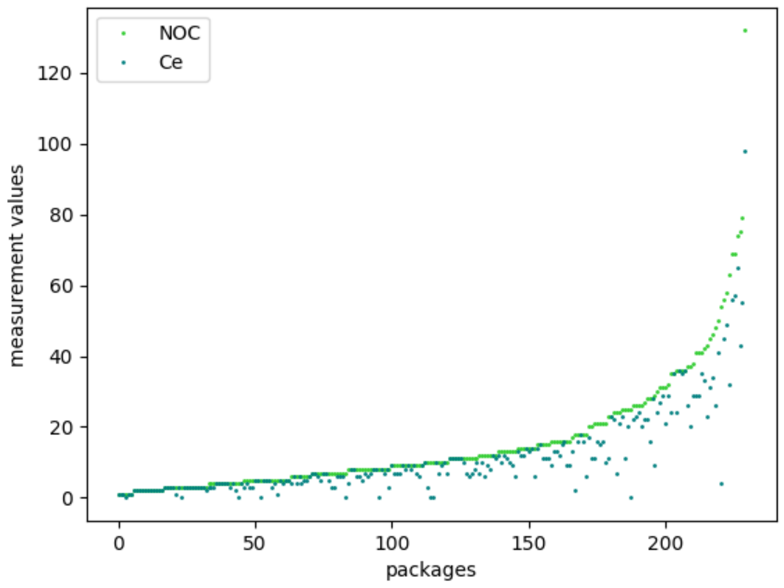


Figure 7.8: Graph for the measurement values of $NOC$ with the corresponding values of $Ce$

## 7.4 Threats to Validity

The following points might impact the computer-aided calculation of the measurement values of the different measures. Therefore, they are considered threats to validity.

**Bugs.** We wrote several test cases with smaller, manually crafted graphs and calculated the measurement values of each measure by hand to check the correctness of the implemented scripts. Using these tests, we found and fixed several bugs such that all tests pass. Nevertheless, our program may still contain bugs we have not discovered yet. We assume, however, that this is very unlikely since the test cases cover various key characteristics and different variants of dependency graphs.

**Jdeps.** While studying the dependency graph generated by Jdeps, we noticed dependencies between nested classes and their parent classes which apparently were added because of their parent-child relation. Hence, in some cases the nested classes did not actually use the parent classes. However, we do not know if a nested classes uses its parent class and therefore we can not remove the dependencies which only exist because of the parent-child relation. Thus, we decided to keep these dependencies since this concerns only 1440 of the 32 183 dependencies, i.e., 4.47% of the dependencies of CPAchecker. Therefore, we assume that these dependencies only have a low impact on the overall result.

**Deny list.** While analyzing the data sets we identified several classes that were only related to test classes, for example, nested classes of test classes which were not automatically removed from the dependency graph. Thus, we implemented a deny list with the found classes so that they were also removed by the script. But there may be a few other classes that should also be excluded for similar reasons which we did not find. If this should be the case, however, we assume that it would only slightly affect the overall result.

**Codebase.** We evaluated the measures using the dependency graph of CPAchecker. This means that the practical evaluation depends on the architecture and code quality of CPAchecker. Therefore, it is possible that certain factors were underestimated, overestimated or even not considered in the evaluation of the data. However, we have been able to show the practical applicability of the measures and therefore consider the practical evaluation to be sufficient, although the measures will likely perform differently on other software systems.

# 8 Future Work

With the theoretical and practical evaluation we have established a solid basis for future work on the proposed measures. Future work regarding this thesis may include the following topics.

**Theoretical Evaluation.** We evaluated the proposed measures using Weyuker's Properties. However, Weyuker's Properties have been criticized by different authors and do not cover all aspects of software measures. Thus, future work could include the theoretical evaluation of the proposed measures using other evaluation frameworks suitable for the analysis of the dependency measures on package-level.

**Practical Evaluation.** We used CPAchecker for the practical analysis of the proposed measures. In addition, practical evaluations of the proposed measures could also be conducted on the example of other open-source software systems with different architecture. Furthermore, the influence of different software architectures on the proposed measures could be analyzed.

**Metric Suite.** The analysis of software systems requires a well-balanced metric suite that covers as many different aspects of software systems as possible. The proposed metrics, however, cover only package-level characteristics of software systems. Hence, it would be useful to combine the proposed metric with other measures that focus on other aspects of software systems. In this context the measures could also be implemented and added to existing tools for measuring software systems.

**SonarQube [27]** This platform provides the capability to not only show health of an application but also to highlight issues newly introduced. For this purpose it uses software metrics to assess the code quality of software systems. As it provides a plugin system for extension with new software measures, it is a good idea to implement the proposed measures with SonarQube to make them accessible to a wide range of developers.

**Automatic Optimization.** The evaluation of large software systems using a metric suite and the search for changes that reduce the complexity of the software is time-consuming and costly. Therefore, the development of tools for automatic optimization is promising. For instance, one could try to use simulated annealing [15] with a fixed number of packages to reduce the overall complexity of a software system (the sum of the measurement values of the packages) by moving the classes between the packages.

**CPAchecker** To avoid further increase of the complexity of CPAchecker in regard to the dependencies of the packages, one could use ArchUnit [2] to enforce a architecture with low complexity according to the proposed measures. ArchUnit is an open-source library to automatically test Java architectures as plain unit tests. That is, ArchUnit can check dependencies between packages and classes, check for cyclic dependencies and more. Furthermore, one could analyze the core dependency graph found during the practical evaluation of the measures to evaluate whether the overall complexity of CPAchecker could be reduced by refactoring and optimization of this graph. In addition, an analysis could be conducted to determine why there are 30 packages which are not used by other packages of CPAchecker.

# 9 Conclusion

We proposed five dependency measures on package-level, four of which were inspired by existing class- or statement-level measures. In addition, we conducted a theoretical evaluation and established Weyuker's Properties on package-level for a formal analysis of of the measures. We also conducted a practical evaluation using the code base and the dependency graph of CPAchecker to evaluate the practical applicability of the measures for which we implemented the prototypical measurement tool Jade, in Python. Regarding the evaluation, only three of the proposed measures $DCM_{CC}$, *P-DepDegree* and $DLM$ met our expectations. However, further evaluations are required to clearly prove the usefulness and applicability of the measures.

# List of Figures

# List of Tables

# Bibliography

[1] S. Almugrin, W. Albattah, and A. Melton. “Using indirect coupling metrics to predict package maintainability and testability.” In: *J. Syst. Softw.* 121 (2016), pp. 298–310. DOI: 10.1016/j.jss.2016.02.024.

[2] *ArchUnit.* https://www.archunit.org/.

[3] M. Bauer and M. Trifu. “Architecture-Aware Adaptive Clustering of OO Systems.” In: *8th European Conference on Software Maintenance and Reengineering (CSMR 2004), 24-26 March 2004, Tampere, Finland, Proceedings.* IEEE Computer Society, 2004, pp. 3–14. DOI: 10.1109/CSMR.2004.1281401.

[4] D. Beyer and A. Fararooy. “A Simple and Effective Measure for Complex Low-Level Dependencies.” In: *The 18th IEEE International Conference on Program Comprehension, ICPC 2010, Braga, Minho, Portugal, June 30-July 2, 2010.* IEEE Computer Society, 2010, pp. 80–83. DOI: 10.1109/ICPC.2010.49.

[5] D. Beyer and P. Häring. “A formal evaluation of DepDegree based on weyuker’s properties.” In: *22nd International Conference on Program Comprehension, ICPC 2014, Hyderabad, India, June 2-3, 2014.* Ed. by C. K. Roy, A. Begel, and L. Moonen. ACM, 2014, pp. 258–261. DOI: 10.1145/2597008.2597794.

[6] C. Bonja and E. Kidanmariam. “Metrics for class cohesion and similarity between methods.” In: *Proceedings of the 44st Annual Southeast Regional Conference, 2006, Melbourne, Florida, USA, March 10-12, 2006.* Ed. by R. Menezes. ACM, 2006, pp. 91–95. DOI: 10.1145/1185448.1185469.

[7] L. C. Briand, J. W. Daly, and J. Wüst. “A Unified Framework for Cohesion Measurement in Object-Oriented Systems.” In: *Empir. Softw. Eng.* 3.1 (1998), pp. 65–117. DOI: 10.1023/A:1009783721306.

[8] L. C. Briand, S. Morasca, and V. R. Basili. “Property-Based Software Engineering Measurement.” In: *IEEE Trans. Software Eng.* 22.1 (1996), pp. 68–86. DOI: 10.1109/32.481535.

[9] S. R. Chidamber and C. F. Kemerer. “A Metrics Suite for Object Oriented Design.” In: *IEEE Trans. Software Eng.* 20.6 (1994), pp. 476–493. DOI: 10.1109/32.295895.

[10] G. R. Choudhary, S. Kumar, K. Kumar, A. Mishra, and C. Catal. "Empirical analysis of change metrics for software fault prediction." In: *Comput. Electr. Eng.* 67 (2018), pp. 15–24. DOI: `10.1016/j.compeleceng.2018.02.043`.

[11] *CPAchecker*. `https://cpachecker.sosy-lab.org/`.

[12] M. O. Elish. "An exploratory study of package metrics as change size indicators in evolving object-oriented software." In: *Comput. Syst. Sci. Eng.* 28.4 (2013).

[13] V. Gupta and J. K. Chhabra. "Package Coupling Measurement in Object-Oriented Software." In: *J. Comput. Sci. Technol.* 24.2 (2009), pp. 273–283. DOI: `10.1007/s11390-009-9223-6`.

[14] V. Gupta and J. K. Chhabra. "Package level cohesion measurement in object-oriented software." In: *J. Braz. Comput. Soc.* 18.3 (2012), pp. 251–266. DOI: `10.1007/s13173-011-0052-4`.

[15] D. Henderson, S. H. Jacobson, and A. W. Johnson. "The Theory and Practice of Simulated Annealing." In: *Handbook of Metaheuristics*. Ed. by F. W. Glover and G. A. Kochenberger. Vol. 57. International Series in Operations Research & Management Science. Kluwer / Springer, 2003, pp. 287–319. DOI: `10.1007/0-306-48056-5\_10`.

[16] M. Hitz and B. Montazeri. "Measuring coupling and cohesion in object-oriented systems." In: *Proceedings of International Symposium on Applied Corporate Computing*. 1995, pp. 25–27.

[17] H. Izadkhah and M. Hooshyar. "Class Cohesion Metrics for Software Engineering: A Critical Review." In: *Comput. Sci. J. Moldova* 25.1 (2017), pp. 44–74.

[18] *jdeps*. `https://docs.oracle.com/javase/9/tools/jdeps.htm`.

[19] R. Kumar, A. Choudhary, and A. Agrawal. "Inheritance Metrics for Object-Oriented Design." In: *International Journal of Computer Science & Information Technology* 2 (Dec. 2010). DOI: `10.5121/ijcsit.2010.2602`.

[20] P. Lima, E. Guerra, P. Meirelles, L. Kanashiro, H. Silva, and F. F. Silveira. "A Metrics Suite for code annotation assessment." In: *J. Syst. Softw.* 137 (2018), pp. 163–183. DOI: `10.1016/j.jss.2017.11.024`.

[21] S. Mal and K. Rajnish. "New Class Cohesion Metric: An Empirical View." In: *International Journal of Multimedia and Ubiquitous Engineering* 9 (June 2014), pp. 367–376. DOI: `10.14257/ijmue.2014.9.6.35`.

[22] R. C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, 2003. ISBN: 0135974445.

[23] V. B. Misic. "Cohesion is Structural, Coherence is Functional: Different Views, Different Measures." In: *7th IEEE International Software Metrics Symposium (METRICS 2001), 4-6 April 2001, London, England, UK.* IEEE Computer Society, 2001, p. 135. DOI: `10.1109/METRIC.2001.915522`.

[24] S. Misra. "Modified Set of Weyuker's Properties." In: *Proceedings of the Firth IEEE International Conference on Cognitive Informatics, ICCI 2006, July 17-19, Beijing, China.* Ed. by Y. Yao, Z. Shi, Y. Wang, and W. Kinsner. IEEE Computer Society, 2006, pp. 242–247. DOI: `10.1109/COGINF.2006.365703`.

[25] S. Misra and I. Akman. "Applicability of Weyuker's properties on OO metrics: Some misunderstandings." In: *Comput. Sci. Inf. Syst.* 5.1 (2008), pp. 17–23. DOI: `10.2298/CSIS0801017M`.

[26] M. L. Ponisio and O. Nierstrasz. "Using Contextual Information to Assess Package Cohesion." In: 2006.

[27] *SonarQube.* `https://www.sonarqube.org/`.

[28] N. Tagoug. "Object-Oriented System Decomposition Quality." In: *7th IEEE International Symposium on High-Assurance Systems Engineering (HASE 2002), 23-25 October 2002, Tokyo, Japan.* IEEE Computer Society, 2002, pp. 230–236. DOI: `10.1109/HASE.2002.1173127`.

[29] E. J. Weyuker. "Evaluating Software Complexity Measures." In: *IEEE Trans. Software Eng.* 14.9 (1988), pp. 1357–1365. DOI: `10.1109/32.6178`.

[30] B. Xu, Z. Chen, and J. Zhao. "Measuring cohesion of packages in Ada95." In: *Proceedings of the 2003 Annual ACM SIGAda International Conference on Ada: The Engineering of Correct and Reliable Software for Real-Time & Distributed Systems using Ada and Related Technologies 2003, San Diego, CA, USA, December 7-11, 2003.* Ed. by R. C. Leif and R. E. Sward. ACM, 2003, pp. 62–67. DOI: `10.1145/958420.958429`.