Online Appendix

1 A BRIEF DESCRIPTION OF SUBJECT SYSTEMS

Our experiments are performed on a set of 18 subject systems (see Table 1). Ant is a Java library and command-line tool to build Java applications; ArgoUML is an OO design tool; jEdit is a programmer's text editor; jHotDraw is a two-dimensional graphics framework for structured drawing editors; jMeter is an application designed to test performance both on static and dynamic resources; wro4j is a web resource optimizer; GWT Portlets is a framework and programming model for building Google Web Toolkit applications; javaclient is a tool supporting the development of applications for Player/Stage; JGAP is a Genetic Algorithms and Genetic Programming package; Mars is a project aiming at modeling important aspects of establishing human settlements on Mars planet; Maze is a Micro-Mouse maze editor and simulator; Neuroph is a lightweight Java neural network framework; tomcat is an open-source Java servlet container; JPMC is a collection of portfolio management components; log4j is a logging utility; PDFBox is a tool for working with PDF documents; Xerces is a library for parsing, validating and manipulating XML documents; and xuml-compiler is a Java Model compiler based on "*Executable UML*" profile.

2 DEFINITIONS OF DESIGN METRICS

Table 2 shows the definitions of design metrics used in our study. These definitions are borrowed from Refs. [7, 10].

3 DEFINITIONS OF UNWEIGHTED NETWORK METRICS

Thung et al. [10] introduced an unweighted network metric suite for KCP, which is composed of 7 network metrics for unweighted networks. These metrics are mainly used to measure the *centrality* (or *importance*) of nodes in the whole network. The definitions listed below are directly copy-pasted from Ref. [10]. We apologize for any lack of originality in our definitions of these metrics.

- **Barycenter Centrality**
  Barycenter centrality is defined based on the sum of shortest distances of node v to all other nodes in a network. The barycenter centrality of node v is computed using the following formula:

$$baryC(v) = \frac{1}{\sum\limits_{u \neq v} sdist(v, u)}, \tag{1}$$

  In the equation, $sdist(v, u)$ refers to the shortest distance from node v to node u.

- **Betweenness Centrality**
  Betweenness centrality is defined based on the number of shortest paths between all possible pairs of other nodes that pass through node v. The betweenness centrality of node v is formulated as follows:

$$betweenC(v) = \sum_{a \neq b \neq v} \frac{spath(a, b, v)}{spath(a, b)}, \tag{2}$$

  In the equation, $spath(a, b, v)$ refers to the number of shortest paths between node a and node b that pass through node v. $spath(a, b)$ refers to the number of shortest paths between node a and node b.

- **Closeness Centrality**
  Closeness centrality is defined based on the mean shortest distance of node v to all the other nodes in a network. The closeness centrality of node v is computed using the following formula:

$$closeC(v) = \sum_{v \neq u} \frac{n-1}{sdist(v, u)}, \tag{3}$$

  In the equation, $sdist(v, u)$ refers to the shortest distance from node v to node u. n refers to the number of nodes in the graph.

- **Eigenvector Centrality**
  Eigenvector centrality measures the importance of node v based on the importance of its neighboring nodes. The eigenvector centrality *EigenC* for a network is measured using the following formula:

$$EigenC(\alpha, \beta) = \alpha(I - \beta R)^{-1} R1 \tag{4}$$

  In the equation, α is a scaling vector for normalizing the score, I is the identity matrix, R is the adjacency matrix representing the network, β is the weighting factor for the adjacency matrix, and 1 is a matrix where the contents of all its cells are ones.

- **Hyperlink-Induced Topic Search (HITS)**
  HITS is an algorithm for ranking nodes using two different scores: hub and authority score. A node with a high hub score represents a node that links to many other nodes and a node

*Zijiang Yang is the corresponding author.

Table 1: Descriptions of the Subject Systems

Systems	Version	Directory	*KLOC*	*#P*	*#C*	*#M*	*#A*	*#KC*	*IR*	Website
ArgoUML	0.9.5	src	74.334	67	846	6,178	2,851	12	1.4%	http://argouml.sourceforge.net
Mars	3.06	src	132.589	95	1,085(32)	11,105	5,738	29	2.6%	http://mars-sim.sourceforge.net/
javaclient	2	src	12.053	39	215	1,479	1,009	57	26.5%	http://java-player.sourceforge.net/
JGAP	3.6.3	src	29.043	27	411(5)	3,186	1,271	18	4.3%	http://sourceforge.net/projects/jgap/
Neuroph	2.2	src	13.657	24	172(4)	1,063	916	24	13.6%	http://neuroph.sourceforge.net/
JPMC	20020123	src	9.283	15	147	926	353	24	16.3%	http://jpmc.sourceforge.net/
wro4j	1.6.3	src	33.736	99	567(9)	3,256	1,274	12	2.1%	http://code.google.com/p/wro4j/
xuml-compiler	0.4.8	all	25.502	58	388(3)	2,919	1,544	37	9.5%	http://code.google.com/p/xuml-compiler/
Maze	1	src	8.881	6	63(6)	563	284	27	39.1%	http://code.google.com/p/maze-solver/
Ant	1.6.1	src/main	81.515	67	900	7,691	4,167	10	1.1% [2]	http://ant.apache.org/
jEdit	5.1.0	src	112.492	41	1,082(9)	7,601	4,085	7	0.6%	http://www.jedit.org/
jHotDraw	6.0b.1	src	28.330	30	544	5,205	865	9	1.7%	http://sourceforge.net/projects/jhotdraw/
jMeter	2.0.1	src/core	22.701	42	260	2,000	834	14	5.4%	http://jmeter.apache.org/
GWT Portlets	0.9.5beta	src	8.501	10	131	1,145	424	27	20.6%	http://code.google.com/p/gwtportlets/
tomcat	7.0.10	src	187.060	136	1,900(24)	16,631	7,950	28	1.5%	http://tomcat.apache.org/
log4j	2.3	src	69.456	91	1,212(25)	6,689	3,101	9	0.7%	https://logging.apache.org/log4j/
PDFBox	2.0.7	all[1]	135.514	109	1,285(33)	10,482	4,792	12	0.9%	https://pdfbox.apache.org/
Xerces	2.11.0	all	133.663	71	1,256	10,481	5,819	6	0.4%	https://xerces.apache.org/

[1] "*all*" means all directories in the source code distribution of the system.
[2] The cells whose *IR* value ≤3 are marked in gray.

Table 2: The definition of design metrics [7, 10]

Metrics	Category	Description
NumAttr	Size	The number of attributes in the class.
NumOps	Size	The number of methods in the class. Also known as WMC (weighted method complexity) in Ref. [3] and NM (Number of Methods) in Ref. [6].
NumPubOps	Size	The number of public methods in the class. Also known as NPM (Number of Public Methods) in Ref. [6].
Setters	Size	The number of methods with a name starting with "set".
Getters	Size	The number of methods with a name starting with "get", "is", or "has".
Dep_Out	Coupling	The number of dependencies where the class is the client.
Dep_In	Coupling	The number of dependencies where the class is the supplier.
EC_Attr	Coupling	The number of times the class is externally used as an attributes type. This is a version of OAEC+AAEC in Ref. [2].
IC_Attr	Coupling	The number of attributes in the class having another class or interface as their types. This is a version of OAIC+AAIC in Ref. [2].
EC_Par	Coupling	The number of times the class is externally used as parameter type. This is a version of OMEC+AMEC in Ref. [2].
IC_Par	Coupling	The number of parameters in the class having another class or interface as their types. This is a version of OMIC+AMIC in Ref. [2].

with a high authority score represents a node that is linked by many different nodes. These scores are computed by the following formulas:

$$hub(v) = \sum_{i=1}^{n} auth(v), \quad (5)$$

$$auth(v) = \sum_{i=1}^{n} hub(v), \quad (6)$$

In the equation, n refers to the number of node in a network, $hub(v)$ refers to the hub score for node v, and $auth(v)$ refers to the authority score for node v. Notice that the definition is a recursive one. To actually arrive with the hub and authority scores for all nodes, one must

first assign an initial value of 1 as hub and authority scores to each of the nodes in the network. The scores would then be updated iteratively until they converge (i.e., there is no further change in any node's hub and authority scores in the entire network). Both hub and authority values are then normalized.

- **PageRank**
  PageRank is an algorithm for measuring node importance proposed by Brin and Page [10]. It suggests that nodes with more incoming links are more important than nodes with less incoming links. It computes the probability that a random walker visits a node from an arbitrary node. Initially, all nodes are assigned with the same initial probability. The scores are then iteratively updated. The PageRank score of node v at iteration i can be computed following the formula:

$$PR(v,i) = \frac{1-r}{T} + r \times \sum_{u \in k(v)} \frac{PR(q,i-1)}{|L(u)|}, \tag{7}$$

In the equation, r is the probability that a random walker continues to visit other nodes (a.k.a. the *damping factor*), T is the number of nodes in the network, $K(v)$ is the set of nodes that link to v, and $L(u)$ is the set of nodes that u links to. The iteration continues until all the scores converge.

4 SOFTWARE NETWORK REPRESENTATION

For our study, we introduce an improved *class dependency network*, CCN_{WD} (Weighted Directed Class Dependency Network) [8], to represent classes[1] and their dependencies in software projects.

Definition 1 (Weighted Directed Class Dependency Network — CCN_{WD}). *The CCN_{WD} of a piece of Java software is actually a weighted directed network (or graph) and can be formally defined as*

$$\text{CDN}_{\text{WD}} = (N, L, W) \tag{8}$$

where N is a set of nodes, which denotes all the classes in the system; $L = \{\langle u,v\rangle \,|u,v \in N \wedge u \neq v \wedge w\langle u,v\rangle > 0\}$ is a set of links, which denotes all the dependencies between any pairs of classes; and $W = \{w\langle u,v\rangle \,|\, \langle u,v\rangle \in L\}$ is a set of weights, which denotes the weights associated with the links in the L. In CCN_{WD}s, we do not allow ≥ 2 links from u to v ($u,v \in N$) — we only keep one if ≥ 2 couplings exist. The weight associated with the link $\langle u,v\rangle \in L$, $w\langle u,v\rangle$, is computed by

$$w\langle u,v\rangle = \sum_{c \in CS} (frq_c \times s_c) \tag{9}$$

where CS={LVA, GVA, INH, IMP, PAR, RET, INS, ACC, MEC} (cf. Definition 2) is a set that contains all the dependencies from u to v, c is a specific type of dependency, frq_c is the dependency frequency of c, and s_c is the dependency strength of c. frq_c is ≥ 1 if there exists at least one c dependency from u to v, and 0 otherwise.

Definition 2 (Dependency Types Between Classes). *For any pair of classes u and v ($u,v \in N$), we take into account the following nine dependency types [1]:*

- *Local VAriable (LVA): If u defines a method m which in turn defines a local variable of type v, then there is an LVA dependency from u to v.*
- *Global VAriable (GVA): If u defines a field f of type v, then there is a GVA dependency from u to v.*
- *INHeritance (INH): If u inherits v via keyword "extends", then there is an INH dependency from u to v.*
- *IMPlementation (IMP): If class u implements interface v via keyword "implements", then there is an IMP dependency from u to v.*
- *PARameter type (PAR): If u defines a method m that has a parameter of type v, then there is a PAR dependency from u to v.*
- *RETurn type (RET): If u defines a method m that has a return type v, then there is an RET dependency from u to v.*
- *INStantiates (INS): If u instantiates an object of v, then there is an INS dependency from u to v.*
- *ACCess (ACC): If one method m defined in u accesses a field f on an object of v, then there is an ACC dependency from u to v.*
- *MEthod Call (MEC): If one method m_1 defined in u calls a method m_2 on an object of v, then there is an MEC dependency from u to v.*

To compute the weight associated with each link, we need to determine three parts beforehand, i.e., *CS*, frq_c, and s_c (cf. Equation (9)). *CS* and frq_c can be resolved by counting their occurrences in the code. But how to determine s_c?

In software engineering literature, three weighting schemes exist to estimate the value of s_c [8]: Ordinal-scale-based Weighting Mechanism (OWM) [5], Empirical Weighting Mechanism (EWM) [9], and Distribution-based Weighting Mechanism (DWM) [4]. OWM estimates the weights according to the relative strengths of different dependency types, and the weight for each dependency type is shown in Table 3. EWM estimates the weights by a weight-tuning process in the context of architectural reconstruction, and the weight for each dependency type is shown in Table 3. DWM estimates the weights according to the distributions of each dependency type across packages, and the weight for dependency type c, s_c, is computed by

$$s_c = \begin{cases} 10, & \text{if } N^c_{intra} \neq 0 \wedge N^c_{\text{inter}} = 0 \\ 1, & \text{if } N^c_{\text{intra}} = 0 \wedge N^c_{\text{inter}} = 0 \\ round(0.5 + 10 \times \frac{N^c_{\text{intra}}}{N^c_{\text{intra}} + N^c_{\text{inter}}}), & \text{otherwise} \end{cases} \tag{10}$$

where N^c_{intra} and N^c_{inter} denote the number of intra- and inter-package couplings of dependency c, respectively. Intra-package dependencies mean the classes where the couplings occur are defined in the same package, while inter-package dependencies mean the classes where the dependencies occur are defined in two separate packages. $round(y)$ rounds y to the nearest integer.

Note that in Table 3, the weight for *ACC* and *INS* is "N/A", which means that OWM does not assign weights to the two dependency types. Thus, if we employ OWM to assign the weights for different dependency types, when building the CCN_{WD}, we need to neglect the two dependency types. Different weighting schemes can produce distinct weight distributions, and thus might affect the importance of classes in the CCN_{WD}. In the experiments, we built the CCN_{WD} using each of the three weighting schemes and validated KEEPER comprehensively.

[1] If not mentioned explicitly, the term *class* designates classes, interfaces, and enum types hereafter.

```java
public interface Song {
    void sTitle();
}//Song is defined in package P1
class Composer implements Song/*IMP*/ {
    public void sTitle() {
        System.out.println("Composer");
    }
}//Composer is defined in package P1
class Copyright extends Composer/*INH*/ {
    public void sTitle() {
        System.out.println("Copyright");
    }
}//Copyright is defined in package P1
class Company {
    public int cCnt;
    public void desc() {
        System.out.println("Company");
    }
}//Company is defined in package P3
class Propaganda {
    private Company myCompany;/*GVA*/
    public void setCompany(Company company/*PAR*/) {
        company.cCnt++/*ACC*/;
    }
    public Copyright/*RET*/ getCopyright() {
        Copyright myCopyright/*LVA*/ = new Copyright()/*INS*/;
        return myCopyright;
    }
    public void say() {
        myCompany.desc()/*MEC*/;
        myCompany.cCnt++/*ACC*/;
    }
}//Propaganda is defined in package P2
```

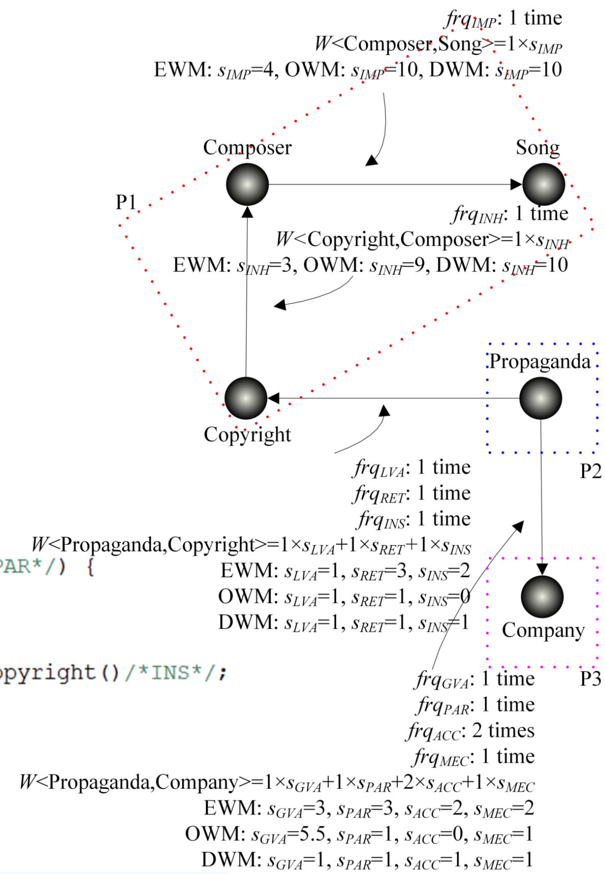

Figure 1: A simple Java code snippet (the left part) and its corresponding $\mathrm{CCN_{WD}}$ (the right part). The text beside each link denotes the dependency types (i.e., *CS*) that the link represents, the frq_c of each coupling c, the formula to compute the weight associated with each link, and the s_c of each dependency c.

Table 3: Weights assigned by OWM and EWM [8]

OWM		EWM	
Weights	Coupling Types	Weights	Coupling Types
1	*MEC, PAR, LVA, RET*	4	*IMP*
5.5	*GVA*	3	*INH, PAR, RET, GVA*
7	*INH* (concrete parent)	2	*MEC, ACC, INS*
9	*INH* (abstract parent)	1	*LVA*
10	*IMP*	—	—
N/A	*ACC, INS*	—	—

Figure 1 gives a simple example to illustrate the idea to build the $\mathrm{CCN_{WD}}$ for a Java code snippet. There is one interface (viz. Song) and four classes (viz. Composer, Copyright, Propaganda, and Company) defined in the Java code snippet, and thus we can create five nodes in the $\mathrm{CCN_{WD}}$. Besides, the interface and classes were coupled through ten dependencies that have been marked explicitly using comments "/**/" in the code snippet. For example, five dependencies exist from Propaganda class to Company class, i.e., one instance of *GVA* dependency (cf. `private Company myCompany; /*GVA*/`), one instance of *PAR* dependency (cf. `public void setCompany(Company company/*PAR*/)`), two instances of *ACC* dependencies (cf. `company.cCnt++/*ACC*/;` and `myCompany.cCnt++/*ACC*/;`), and one instance of *MEC* dependency (cf. `myCompany.desc()/*MEC*/;`). Thus, we can create a link from the node of Propaganda class to the node of Company class. Note that though five dependencies exist, we keep only one (cf. Definition 1). Then, the weight associated with this link is computed by $w\langle Propaganda, Company\rangle = 1{\times}s_{GVA}{+}1{\times}s_{PAR}{+}2{\times}s_{ACC}{+}1{\times}s_{MEC}$. If we use DWM to assign the weights, then $w\langle Propaganda, Company\rangle = 1\times1+1\times1+2\times1+1\times1 = 5$. Other links in the $\mathrm{CCN_{WD}}$ can be similarly created.

REFERENCES

[1] Lionel C. Briand, John W. Daly, and Jürgen Wüst. 1999. A Unified Framework for Coupling Measurement in Object-Oriented Systems. *IEEE Trans. Software Eng.* 25, 1 (1999), 91–121.

[2] Lionel C. Briand, Premkumar T. Devanbu, and Walcélio L. Melo. 1997. An Investigation into Coupling Measures for C++. In *Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997.* ACM, 412–421.

[3] Shyam R. Chidamber and Chris F. Kemerer. 1994. A Metrics Suite for Object Oriented Design. *IEEE Trans. Software Eng.* 20, 6 (1994), 476–493.

[4] Fernando Brito e Abreu and Miguel Goulão. 2001. Coupling and Cohesion as Modularization Drivers: Are We Being Over-Persuaded?. In *Fifth Conference on Software Maintenance and Reengineering, CSMR 2001, Lisbon, Portugal, March*

14-16, 2001, Pedro Sousa and Jürgen Ebert (Eds.). IEEE Computer Society, 47–57.

[5] Dazhou Kang, Baowen Xu, Jianjiang Lu, and William C. Chu. 2004. A Complexity Measure for Ontology Based on UML. In *10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2004), 26-28 May 2004, Suzhou, China*. IEEE Computer Society, 222–228.

[6] Al Lake and Curtis Cook. 1994. Use of factor analysis to develop OOP software complexity metrics. In *Proc. 6th Annual Oregon Workshop on Software Metrics, Silver Falls, Oregon*. Citeseer.

[7] Mohd Hafeez Osman, Michel R. V. Chaudron, and Peter van der Putten. 2013. An Analysis of Machine Learning Algorithms for Condensing Reverse Engineered Class Diagrams. In *2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, September 22-28, 2013*. IEEE Computer Society, 140–149.

[8] Weifeng Pan, Hua Ming, Dae-Kyoo Kim, and Zijiang Yang. 2022. PRIDE: Prioritizing documentation effort based on a PageRank-like algorithm and simple filtering rules. *IEEE Transactions on Software Engineering* 49, 3 (2022), 1118–1151.

[9] Ioana Sora and Ciprian-Bogdan Chirila. 2019. Finding key classes in object-oriented software systems by techniques based on static analysis. *Inf. Softw. Technol.* 116 (2019).

[10] Ferdian Thung, David Lo, Mohd Hafeez Osman, and Michel R. V. Chaudron. 2014. Condensing class diagrams by analyzing design and network metrics using optimistic classification. In *22nd International Conference on Program Comprehension, ICPC 2014, Hyderabad, India, June 2-3, 2014*. ACM, 110–121.