Supplemental Material for: Cohesion-Driven Decomposition of Service Interfaces Without Access to Source Code

Dionysis Athanasopoulos, Apostolos V. Zarras, *Member, IEEE*, George Miskos, Valerie Issarny,
and Panos Vassiliadis *Member, IEEE*

◆

APPENDIX A
SUMMARY OF RELATED WORK

Table 5, provides a summary of metrics-driven refactoring approaches that have been proposed in the past and highlights the contribution of our approach for the cohesion-driven decomposition of service interfaces. Moreover, Table 6, briefly summarizes the relation between the cohesion metrics that we employ in our approach, the object-oriented cohesion metrics surveyed in [21], and the service-oriented cohesion metrics that have been proposed in [5], [6], [22].

TABLE 5
A summary of metrics-driven refactoring approaches.

Refactoring Approach	*Purpose*	*Type of relations*
[18]	Class coupling	Implementation-level
[7]	Class cohesion	Implementation-level
[8]	Class cohesion	Implementation-level
[9]	Class cohesion	Implementation-level
[10]	Class coupling, cohesion	Implementation-level
[11]	Class coupling, cohesion	Implementation-level
[12]	Class coupling, cohesion, code complexity	Implementation-level
[13]	Class coupling, cohesion, code complexity	Implementation-level
[14]	Class coupling, cohesion, code size	Implementation-level
[15]	Class coupling, cohesion, code size, code complexity	Implementation-level
Proposed approach	Service interfaces cohesion	Interface-level

TABLE 6
A summary of cohesion metrics [21], [5], [6], [22].

	Implementation-level	*Interface-level*
Class cohesion	$LCOM1$, $LCOM2$, $LCOM3$, $LCOM4$, $LCOM5$, TCC, LCC, DCD, DCI, CC, $SCOM$, DMC, $CBMC$, $LSCC$	$CAMC$, NHD, $SNHD$, $MMAC$ $CAMC$,
Service cohesion	$SIIC$, SCV	$SIDC$, $SISC$, $SIUC$
		Proposed metrics: LoC_{msg}, LoC_{conv}, LoC_{dom}

APPENDIX B
ANALYTIC VALIDATION OF COHESION METRICS

In the 90's, Briand et al. [25] proposed a mathematical framework for the theoretical validation of cohesion metrics. Here, we rely on this framework for the validation of the metrics that we employ for the cohesion-driven decomposition of service interfaces. Briefly, in [25], a software system is represented by a graph $S = (E, R)$, where E is the set of elements that constitute the system and $R \subseteq E \times E$ is a set of relations between elements. A module of the system is represented by a graph $m = (E_m, R_m)$, where $E_m \subseteq E$, and $R_m \subseteq R$. According to Briand et al., a cohesion metric has to satisfy the following properties:

- *Nonnegativity and normalization*: The cohesion of a module $m = (E_m, R_m)$ belongs to a specified interval, i.e., $cohesion(m) \in [0, M]$.
- *Null value*: The cohesion of a module $m = (E_m, R_m)$ is null if R_m is empty, i.e., $R_m = \emptyset \Rightarrow cohesion(m) = 0$.
- *Monotonicity*: Let $m = (E_m, R_m)$ and $m' = (E_m, R_{m'})$ be two modules (with the same set of elements), such that $R_m \subseteq R_{m'}$. Then, $cohesion(m) \leq cohesion(m')$.
- *Cohesive modules*: Let $m_1 = (E_{m_1}, R_{m_1})$ and $m_2 = (E_{m_2}, R_{m_2})$ be two unrelated modules and $m_{1\cup 2}$ is the union of m_1, m_2. Then, $cohesion(m_{1\cup 2}) \leq \max(cohesion(m_2), cohesion(m_2))$.

- *D. Athanasopoulos, A. Zarras, G. Miskos and P. Vassiliadis are with the Department of Computer Science, University of Ioannina, Greece. E-mail: {dathanas, zarras, pvassil}@cs.uoi.gr, gmiskos@gmail.com*
- *V. Issarny is with the Inria research center of Paris-Rocquencourt, France E-mail: Valerie.Issarny@inria.fr*

For brevity, we focus our validation on LoC_*. The three different refinements of LoC_* can be validated identically. Differently from [25], the metrics that we consider measure the *lack of cohesion* of service interfaces. Moreover, the interface-level graphs that we employ are *weighted*. Hence, we appropriately adapt the properties that should hold for the proposed metrics. We begin our validation with the proof of a supportive lemma, which concerns the similarity functions that we employ. Then, we prove that LoC_* satisfies the properties of the Briand et al. framework.

***Lemma** 1:* The similarity functions that we use for the different notions of cohesion belong to the interval $[0, 1]$.

Proof: In the case of *message-level cohesion*, the similarity, $OpS_{msg}(op_i, op_j)$, between two operations is the average of the similarities between the input/output messages of op_i, op_j. The similarity, $MsgS(m_i, m_j)$, between two messages is measured based on the message-level graphs G_{m_i}, G_{m_j} of the messages. On the one extreme, the maximum common subgraph $G_{m_i \cap m_j}$ of G_{m_i}, G_{m_j} may be trivial if G_{m_i}, G_{m_j} have nothing in common. In this case we have:

$$|V_{m_i \cap m_j}| = 0 \tag{1}$$

On the other extreme, G_{m_i}, G_{m_j} may be identical, in which case we have:

$$|V_{m_i}| = |V_{m_j}| = |V_{m_i \cap m_j}| = |V_{m_i \cup m_j}| \tag{2}$$

From (1) and (2) we have:

$$\begin{aligned} &0 \leq MsgS(m_i, m_j) \leq 1 \Rightarrow \\ &0 \leq OpS_{msg}(op_i, op_j) \leq 1 \end{aligned} \tag{3}$$

In *conversation-level cohesion*, the similarity $OpS_{conv}(op_i, op_j)$ is also an average of message similarities. Hence:

$$0 \leq OpS_{conv}(op_i, op_j) \leq 1 \tag{4}$$

In *domain-level cohesion*, the similarity $OpS_{dom}(op_i, op_j)$ is measured based on the sets of domain-level terms T_{op_i} and T_{op_j}. On the one extreme, T_{op_i} and T_{op_j} may have nothing in common. In this case we have:

$$|T_{op_i} \cap T_{op_j}| = 0 \tag{5}$$

On the other extreme, T_{op_i} and T_{op_j} may be identical, in which case we have:

$$|T_{op_i}| = |T_{op_j}| = |T_{op_i} \cup T_{op_j}| = |T_{op_i} \cap T_{op_j}| \tag{6}$$

(5) and (6) imply that:

$$0 \leq OpS_{dom}(op_i, op_j) \leq 1 \tag{7}$$

□

***Theorem** 1:* For a service interface, si, and the interface-level graph, $G^*_{si} = (V_{si}, E_{si})$, that represents the interface, $0 \leq LoC_*(si, OpS_*) \leq 1$.

Proof: Based on Lemma 1, for any two operations op_i, op_j of si we have:

$$0 \leq OpS_*(op_i, op_j) \leq 1 \tag{8}$$

From graph theory we further have:

$$|E_{si}| \leq \frac{|V_{si}| * (|V_{si}| - 1)}{2} \tag{9}$$

Based on (8) and (9) the following holds:

$$\begin{aligned} &0 \leq \frac{\sum_{(op_i, op_j) \in E_{si}} OpS_*(op_i, op_j)}{\frac{|V_{si}|*(|V_{si}|-1)}{2}} \leq 1 \Rightarrow \\ &0 \leq LoC_*(si, OpS_*) \leq 1 \end{aligned} \tag{10}$$

□

***Theorem** 2:* Let si be a service interface, represented by the interface-level graph, $G^*_{si} = (V_{si}, E_{si})$. If $E_{si} = \emptyset$, then $LoC_*(si, OpS_*) = 1$.

Proof: Given that E_{si} is empty we have:

$$\begin{aligned} &E_{si} = \emptyset \Rightarrow \sum_{(op_i, op_j) \in E_{si}} OpS_*(op_i, op_j) = 0 \Rightarrow \\ &LoC_*(si, OpS_*) = 1 \end{aligned} \tag{11}$$

□

***Theorem** 3:* Let si, si' be two service interfaces, represented by the interface-level graphs, $G^*_{si} = (V_{si}, E_{si})$, $G^*_{si} = (V_{si}, E_{si'})$. G^*_{si} and $G^*_{si'}$ have the same nodes. Moreover, $E_{si} \subseteq E_{si'}$ and $\sum_{(op_i, op_j) \in E_{si}} OpS_*(op_i, op_j) \leq \sum_{(op_i, op_j) \in E_{si'}} OpS_*(op_i, op_j)$. Then, $LoC_*(si, OpS_*) \geq LoC_*(si', OpS_*)$.

Proof: Given the initial assumptions of the theorem for si and si' (i.e., G^*_{si} and $G^*_{si'}$ have the same nodes, $E_{si} \subseteq E_{si'}$, and $\sum_{(op_i, op_j) \in E_{si}} OpS_*(op_i, op_j) \leq \sum_{(op_i, op_j) \in E_{si'}} OpS_*(op_i, op_j)$), the following implications hold:

$$\begin{aligned} &\frac{\sum_{(op_i, op_j) \in E_{si}} OpS_*(op_i, op_j)}{\frac{|V_{si}|*(|V_{si}|-1)}{2}} \leq \\ &\qquad \frac{\sum_{(op_i, op_j) \in E_{si'}} OpS_*(op_i, op_j)}{\frac{|V_{si}|*(|V_{si}|-1)}{2}} \Rightarrow \\ &1 - \frac{\sum_{(op_i, op_j) \in E_{si}} OpS_*(op_i, op_j)}{\frac{|V_{si}|*(|V_{si}|-1)}{2}} \geq \\ &\qquad 1 - \frac{\sum_{(op_i, op_j) \in E_{si'}} OpS_*(op_i, op_j)}{\frac{|V_{si}|*(|V_{si}|-1)}{2}} \Rightarrow \\ &LoC_*(si, OpS_*) \geq LoC_*(si', OpS_*) \end{aligned} \tag{12}$$

□

***Theorem** 4:* Let si_1, si_2 be two unrelated service interfaces, represented by the interface-level graphs, $G^*_{si_1} = (V_{si_1}, E_{si_1})$, $G^*_{si_2} = (V_{si_2}, E_{si_2})$. Let $si_{1\cup 2}$ be the union of si_1, si_2, represented by, $G^*_{si_{1\cup 2}} = (V_{si_{1\cup 2}}, E_{si_{1\cup 2}})$. Then, $LoC_*(si_{1\cup 2}, OpS_*) \geq \max(LoC_*(si_1, OpS_*), LoC_*(si_2, OpS_*))$.

Proof: Without loss of generality, we assume that si_2 is more cohesive than si_1. Based on this assumption, we have:

$$\begin{gathered} LoC_*(si_1, OpS_*) \geq LoC_*(si_2, OpS_*) \Rightarrow \\ \frac{\sum_{(op_i,op_j)\in E_{si_1}} OpS_*(op_i, op_j)}{\frac{|V_{si_1}|*(|V_{si_1}|-1)}{2}} \geq \\ \frac{\sum_{(op_i,op_j)\in E_{si_2}} OpS_*(op_i, op_j)}{\frac{|V_{si_2}|*(|V_{si_2}|-1)}{2}} \Rightarrow \\ |V_{si_2}| * (|V_{si_2}| - 1) * \sum_{E_{si_1}} OpS_*(op_i, op_j) \geq \\ |V_{si_1}| * (|V_{si_1}| - 1) * \sum_{E_{si_2}} OpS_*(op_i, op_j) \end{gathered} \tag{13}$$

Given that si_1, si_2 are unrelated we have that:

$$\begin{gathered} \forall (op_{si_1}, op_{si_2}) \in V_{si_1} \times V_{si_2}, \\ OpS_*(op_{si_1}, op_{si_2}) = 0 \end{gathered} \tag{14}$$

From (14) we derive the following for the interface-level graph $G^*_{si_{1\cup 2}}$ that represents the union of si_1, si_2:

$$V_{si_{1\cup 2}} = V_{si_1} \cup V_{si_2} \tag{15}$$

$$E_{si_{1\cup 2}} = E_{si_1} \cup E_{si_2} \tag{16}$$

From (15), (16) we have that:

$$\begin{gathered} LoC_*(si_{1\cup 2}, OpS_*) = \\ 1 - \frac{\sum_{E_{si_1}} OpS_*(op_i, op_j) + \sum_{E_{si_2}} OpS_*(op_i, op_j)}{\frac{(|V_{si_1}|+|V_{si_2}|)*(|V_{si_1}|+|V_{si_2}|-1)}{2}} \end{gathered} \tag{17}$$

Given (17), to prove the theorem we have to show that the following inequality holds:

$$\begin{gathered} 1 - \frac{\sum_{E_{si_1}} OpS_*(op_i, op_j) + \sum_{E_{si_2}} OpS_*(op_i, op_j)}{\frac{(|V_{si_1}|+|V_{si_2}|)*(|V_{si_1}|+|V_{si_2}|-1)}{2}} \geq \\ 1 - \frac{\sum_{(op_i,op_j)\in E_{si_1}} OpS_*(op_i, op_j)}{\frac{|V_{si_1}|*(|V_{si_1}|-1)}{2}} \end{gathered} \tag{18}$$

From (18), with trivial algebraic operations, we derive the following inequality that must hold to prove the theorem:

$$\begin{gathered} 2 * |V_{si_1}| * |V_{si_2}| + \\ |V_{si_2}| * (|V_{si_2}| - 1) * \sum_{(op_i,op_j)\in E_{si_1}} OpS_*(op_i, op_j) \geq \\ |V_{si_1}| * (|V_{si_1}| - 1) * \sum_{(op_i,op_j)\in E_{si_2}} OpS_*(op_i, op_j) \end{gathered} \tag{19}$$

Given that (13) holds, (19) is also true.

□

APPENDIX C
DECOMPOSITION METHOD TERMINATION & COMPLEXITY

The analysis of the proposed decomposition method focuses on two issues. First, we prove that the cohesion-driven decomposition of service interfaces terminates. Second, we show that the complexity of decomposing a given interface with the proposed method is, in the worst case, cubic to the number of operations, offered by the given interface.

***Theorem** 5:* Given a service interface si, Algorithm $decomposeInterface$ terminates.

Proof: $decomposeInterface$ performs a number of iterations, until the size of Q is 0. During each iteration, $decomposeInterface$ picks a service interface r_i from Q. If the cohesion of r_i can not be improved, r_i is put in the results set R_I. Otherwise, the cohesion of r_i is improved by splitting it in two new interfaces r_r, r_s, which are stored in Q. $decomposeInterface$ can not perform infinite splits because:

- The lower bound for the lack of cohesion of a service interface is 0 (Theorem 1).
- The lower bound for the number of operations of a service interface is 1.

Therefore, the size of Q eventually becomes 0 and $decomposeInterface$ terminates.

□

***Theorem** 6:* In the worst case, the complexity of decomposing a service interface, si, is cubic to the number of operations of si.

Proof: In the worst case scenario, Algorithm $decomposeInterface$ starts with si that provides $|si.O|$ operations and results in $|si.O|$ new interfaces, one per operation. To achieve this, the algorithm starts with si and splits it in two new interfaces r_r and r_s. The new interfaces are enqueued in Q. In the i-th iteration, one of the intermediate interfaces, r_i is chosen and split again. Hence, at the end of the i-th iteration, Q contains $i+1$ new interfaces. Once, the size of Q becomes $|si.O|$, there are another $|si.O|$ iterations to dequeue the interfaces that are held in Q (again in the worst case). Therefore, in the worst case $decomposeInterface$ performs $2 * |si.O|$ iterations.

The two factors that affect the complexity of splitting an intermediate interface r_i in two interfaces is the creation (Algorithm *createSplinter*) and the population (Algorithm *populateSplinter*) of the splinter interface r_s.

- *createSplinter* performs $|r_i.O|$ iterations to find the operation, op_s, whose removal maximizes the cohesion improvement of r_i. Then, it creates r_s that contains op_s, and r_r that contains the rest of the r_i operations.

TABLE 7

Amazon services: Changes per participant: % of moved operations and % of decomposition size decrease.

	Participant 1		*Participant 2*		*Participant 3*		*Participant 4*		*Participant 5*	
ID	*% move oper.*	*% DS decr.*	*% move oper.*	*% DS decr.*	*% move oper.*	*% DS decr.*	*% move oper.*	*% DS decr.*	*% move oper.*	*% DS decr.*
A1	01.15	14.81	02.30	03.70	03.45	11.11	02.30	14.81	00.00	07.41
A2	03.70	00.00	00.00	10.00	03.70	10.00	03.70	10.00	00.00	30.00
A3	00.00	00.00	07.41	06.25	07.41	12.50	00.00	00.00	00.00	16.67
A4	04.35	16.67	00.00	16.67	04.35	16.67	04.35	16.67	00.00	00.00
A5	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00
A6	00.00	00.00	00.00	00.00	-	-	00.00	00.00	00.00	25.00
A7	-	-	06.25	00.00	06.25	00.00	12.50	16.67	-	-
A8	00.00	16.67	00.00	16.67	00.00	00.00	00.00	16.67	00.00	16.67
A9	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00
A10	07.69	00.00	07.69	00.00	15.38	00.00	07.69	00.00	00.00	00.00
A11	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	15.38	00.00

TABLE 8

Yahoo services: Changes per participant: % of moved operations and % of decomposition size decrease.

	Participant 6		*Participant 7*		*Participant 8*		*Participant 9*		*Participant 10*	
ID	*% move oper.*	*% DS decr.*	*% move oper.*	*% DS decr.*	*% move oper.*	*% DS decr.*	*% move oper.*	*% DS decr.*	*% move oper.*	*% DS decr.*
Y1	00.00	20.00	00.00	00.00	00.00	13.33	-	-	-	-
Y2	00.00	11.11	00.00	00.00	00.00	11.11	03.57	22.22	00.00	33.33
Y3	14.29	00.00	07.14	18.18	10.71	14.29	00.00	14.29	00.00	28.57
Y4	00.00	00.00	13.04	00.00	17.39	12.50	13.04	12.50	00.00	20.00
Y5	-	-	00.00	00.00	05.00	18.18	00.00	09.09	-	-
Y6	-	-	00.00	11.11	00.00	22.22	00.00	11.11	-	-
Y7	00.00	00.00	00.00	00.00	00.00	11.11	00.00	11.11	00.00	33.33
Y8	08.33	00.00	16.67	00.00	25.00	14.29	00.00	00.00	08.33	25.00
Y9	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00
Y10	00.00	25.00	20.00	00.00	00.00	00.00	20.00	00.00	00.00	00.00
Y11	00.00	00.00	00.00	00.00	-	-	00.00	00.00	00.00	00.00

- $populateSplinter$, takes as input the interfaces r_r, r_s that result from $createSplinter$. Hence, $|r_r.O| = |r_i.O| - 1$ and $|r_s.O| = 1$. To improve the cohesion of the two interfaces $populateSplinter$ moves operations from r_r to r_s. In the worst case, we can have $|r_r.O| - 1 = |r_i.O| - 2$ operations moved, i.e., the outer loop of $populateSplinter$ performs $|r_i.O| - 1$ iterations. To find the first operation, the inner loop of $populateSplinter$ performs $|r_i.O| - 1$ iterations. To find the i-th operation, the inner loop of $populateSplinter$ performs $|r_i.O| - 1 - i + 1$ iterations, and so on. Therefore, the overall number of iterations performed is $\sum_{i=1}^{|r_i.O|-1} |r_i.O| - i = \sum_{i=1}^{|r_i.O|-1} i = \frac{|r_i.O| * (|r_i.O|-1)}{2}$.

Based on the previous analysis, in the worst case the complexity of decomposing si is $O(|si.O|^3)$.

□

APPENDIX D
INDIVIDUAL PARTICIPANTS' SUGGESTIONS FOR IMPROVEMENT

Tables 7, 8 give a detailed summary of the changes that have been performed by the participants on the decompositions that they selected. In particular, for each one of the examined interfaces and each participant we provide the percentage of the moved operations (over the size of the examined interface) and the percentage of the decomposition size decrease.

Overall, the percentage of moved operations ranged from 1.15% to 15.38% whenever this happened in Amazon services and 03.57% to 25% for the Yahoo services. The percentage of the decomposition size decrease ranged from 3.70% to 25% for Amazon and 11.11% to 33.33% for Yahoo services.

REFERENCES

[1] W. Stevens, G. Myers, and L. Constantine, "Structured Design," *IBM Systems Journal*, vol. 13, no. 2, pp. 115–139, 1974.

[2] M. Papazoglou and W.-J. van den Heuvel, "Service-Oriented Design and Development Methodology," *International Journal on Web Engineering and Technology*, vol. 2, no. 4, pp. 412–442, 2006.

[3] C. Legner and T. Vogel, "Design Principles for B2B Services - An Evaluation of two Alternative Service Designs," in *Proceedings of the IEEE International Conference on Service Computing (SCC)*, 2007, pp. 372–379.

[4] T. Kohlborn, A. KortHaus, T. Chan, and M. Rosemann, "Identification and Analysis of Business and Software Services - A Consolidated Approach," *IEEE Transactions on Services Computing*, vol. 2, no. 1, pp. 1–15, 2009.

[5] M. Perepletchikov, C. Ryan, and Z. Tari, "The Impact of Service Cohesion on the Analyzability of Service-Oriented Software," *IEEE Transactions on Services Computing*, vol. 3, no. 2, pp. 89–103, 2010.

[6] D. Athanasopoulos and A. Zarras, "Fine-Grained Metrics of Cohesion Lack for Service Interfaces," in *Proceedings of the 9th IEEE International Conference on Web Services (ICWS)*, 2011, pp. 588–595.

[7] N. Tsantalis and A. Chatzigeorgiou, "Identification of Move Method Refactoring Opportunities," *IEEE Transactions on Software Engineering*, vol. 99, no. 3, pp. 347–367, 2009.

[8] F. Simon, F. Steinbrückner, and C. Lewerentz, "Metrics Based Refactoring," in *Proceedings of the 5th IEEE European Conference on Software Maintenance and Reengineering (CSMR)*, 2001, pp. 30–39.

[9] M. Fokaefs, N. Tsantalis, A. Chatzigeorgiou, and J. Sander, "Decomposing Object-Oriented Class Modules Using an Agglomerative Clustering Technique," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*, 2009, pp. 93–101.

[10] D. Doval, S. Mancoridis, and B. S. Mitchell, "Automatic Clustering of Software Systems Using a Genetic Algorithm," in *Proceedings of the 9th IEEE Software Technology and Engineering Practice (STEP)*, 1999, pp. 73–81.

[11] B. D. Bois, S. Demeyer, and J. Verelst, "Refactoring: Improving Coupling and Cohesion of Existing Code," in *Proceedings of the 11th IEEE Working Conference on Reverse Engineering (WCRE)*, 2004, pp. 144–151.

[12] O. Seng, J. Stammel, and D. Burkhart, "Search-Based Determination of Refactorings for Improving the Class Structure of Object-Oriented Systems," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, 2006, pp. 1909–1916.

[13] L. Tahvildari and K. Kontogiannis, "Improving Design Quality Using Meta-Pattern Transformations: A Metric-Based Approach," *Journal of Software Maintenance*, vol. 16, no. 4-5, pp. 331–361, 2004.

[14] M. O'Keeffe and M. í Cinnéide, "Search-Based Refactoring for Software Maintenance," *Journal of Systems and Software*, vol. 81, no. 4, pp. 502–516, 2008.

[15] M. Bowman, L. C. Briand, and Y. Labiche, "Solving the Class Responsibility Assignment Problem in Object-Oriented Analysis with Multi-Objective Genetic Algorithms," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 817–837, 2010.

[16] W. F. Opdyke, "Refactoring Object-Oriented Frameworks," Ph.D. dissertation, Univ. of Illinois - Urbana Champaign, 1992.

[17] T. Mens and T. Tourwé, "A Survey of Software Refactoring," *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 126–139, 2004.

[18] M. Harman and L. Tratt, "Pareto Optimal Search Based Refactoring at the Design Level," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2007, pp. 1106–1113.

[19] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[20] L. C. Briand, J. W. Daly, and J. Wüst, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems," *Empirical Software Engineering*, vol. 3, no. 1, pp. 65–117, 1998.

[21] J. A. Dallal and L. Briand, "A Precise Method-Method Interaction-Based Cohesion Metric for Object-Oriented Classes," *ACM Transactions on Software Engineering and Methodology*, vol. 21, no. 2, 2012.

[22] A. Kazemi, A. Rostampour, A. Zamiri, P. Jamshidi, H. Haghighi, and F. Shams, "An Information Retrieval Based Approach for Measuring Service Conceptual Cohesion," in *Proceedings of the 11th IEEE International Conference on Quality Software (QSIC)*, pp. 102–111.

[23] Y. Ma, k. Lu, Y. Zhang, and B. Jin, "Measuring Ontology Information by Rules Based Transformation," *Knowledge-Based Systems*, vol. 50, no. 0, pp. 234–245, 2013.

[24] G. Bordogna and G. Pasi, "A quality driven hierarchical data divisive soft clustering for information retrieval," *Knowledge-Based Systems*, vol. 26, no. 0, pp. 9–19, 2012.

[25] L. Briand, S. Morasca, and V. R. Basili, "Property-Based Software Engineering Measurement," *IEEE Transactions on Software Engineering*, vol. 22, no. 1, pp. 68–86, 1996.

[26] G. Valiente, *Algorithms on Trees and Graphs*. Springer, 2000.

[27] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and A. Lau, "An Empirical Study on Web Service Evolution," in *Proceedings of the 9th IEEE International Conference on Web Services (ICWS)*, 2011, pp. 49–56.

ACKNOWLEDGEMENTS

We would like to warmly thank all the developers that participated in the user study, for their time, effort and valuable comments. This work received funding from the European Community's FP7/2007-2013 under grant agreement number 257178 (project CHOReOS).