Multi-objective Module Clustering for Kate

Matheus Paixao, Mark Harman and Yuanyuan Zhang

University College London, CREST Centre, UK

Abstract. This paper applies multi-objective search based software re-modularization to the program Kate, showing how this can improve cohesion and coupling, and investigating differences between weighted and unweighted approaches and between equal-size and maximising clusters approaches. We also investigate the effects of considering omnipresent modules. Overall, we provide evidence that search based modularization can benefit Kate developers.

Keywords: Software Module Clustering, Multi-objective Optimization, Search Based Software Engineering

1 Introduction

This paper reports on experiments with multi-objective search based software re-modularization through module clustering applied to the system Kate [1], a C/C++ editor for KDE platforms. Both unweighted and weighted data were considered, as well as omnipresent modules. We follow the approach initially introduced by Mitchell and Mancoridis [2], in which a Module Dependency Graph (MDG) is remodularized to improve cohesion and coupling, as more recently amended and extended by Praditwong et al [3] to the multi-objective optimization paradigm. In the unweighted MDG, an edge between two modules denotes a dependency between these modules. For a weighted MDG, an edge denotes the strength of the dependency, which is represented by the edge weight [4]. By using the MDG, the optimization algorithm can then search for a partition of this graph that optimizes the considered quality metrics.

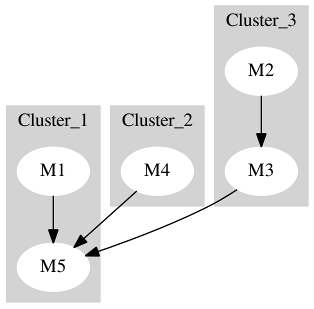


Fig. 1: Modularization Example

Search based modularization seeks partitions that cluster modules to favour high cohesion and low coupling. Consider the simple example (shown in Figure 1), in which edges denote inter-module dependencies. A simple solution would be $X =< 1, 3, 3, 2, 1 >$. This vector of module assignments denotes a modularization solution of the five modules into three clusters. Modules m_1 and m_5 are in cluster c_1, m_2 and m_3 in c_3, and finally m_4 in c_2. The MDG that represents these five modules is depicted with this modularization solution in Figure 1, on the left.

The set of fitness functions considered by the two approaches are presented next:

– **Maximizing Clusters (MCA)**
  - cohesion (max)
  - coupling (min)
  - number of clusters (max)
  - MQ (max)
  - number of isolated clusters (min)

– **Equal-size Clusters (ECA)**
  - cohesion (max)
  - coupling (min)
  - number of clusters (max)
  - MQ (max)
  - cluster size difference (min)

The metrics of cohesion and coupling are related to the dependencies between modules. Cohesion is the sum of the weights of all edges that start and finish in the same cluster. On the other hand, coupling is the sum of the weights of all edges that start in a cluster and finish in another cluster. MQ means Modularization Quality [2], which is the metric used in the previous single objective works. An isolated cluster is the one that has only one module inside it. To illustrate each fitness function of both MCA and ECA, consider the modularization example given in Figure 1. The set of metrics would be assigned as *cohesion: 2, coupling: 2, number of clusters: 3, MQ: 0.66, isolated clusters: 1, cluster size difference: 1.*

2 Modularizing Kate using SBSE

Kate's source code is organized in two folders, *src* and *session*, where each folder accommodates some classes. First, the call graph of each function and the inheritance graph between classes were extracted using Doxygen [5]. Kate's unweighted and weighted MDGs were created from these graphs, where each class is considered a module, and a function call or inheritance represents a dependency between modules. The weight of an edge in the weighted MDG is the number of functions calls between the classes. For the unweighted MDG, all edges have the same weight of 1. The clusters are the folders the classes are in. The original Kate's unweighted MDG can be seen in Figure 2. Function calls are represented by black arrows and inheritance relationships by red arrows. As can be seen, Kate has only two clusters, corresponding to *src* and *session.*

We used the Two-Archive Genetic Algorithm [6], configured based on previous work [3] with crossover probability 0.8 and mutation probability $0.004 \log_2(M)$, where M is the number of modules. The population size used was $10M$, and the algorithm was executed for 10000 generations.

We used both MCA and ECA optimization approaches, each of which was executed 30 times. Each execution generates a pareto front. In order to compare the two approaches, the solution with highest cohesion was selected as a representative of each execution. This set of representative solutions was then used to compute the average and standard deviations of each quality metric. We also performed non-parametric statistical testing and effect size assessment using a paired Wilcoxon and Vargha-Delaney tests, respectively, as recommended in guidance on assessing algorithms differences for SBSE [7, 8]. These tests were carried out using the systematic metaheuristic comparison tool Astraiea [9].

Space is limited to six pages. Although this paper is self-contained, the interested reader can find more examples of modularization results, analysis and

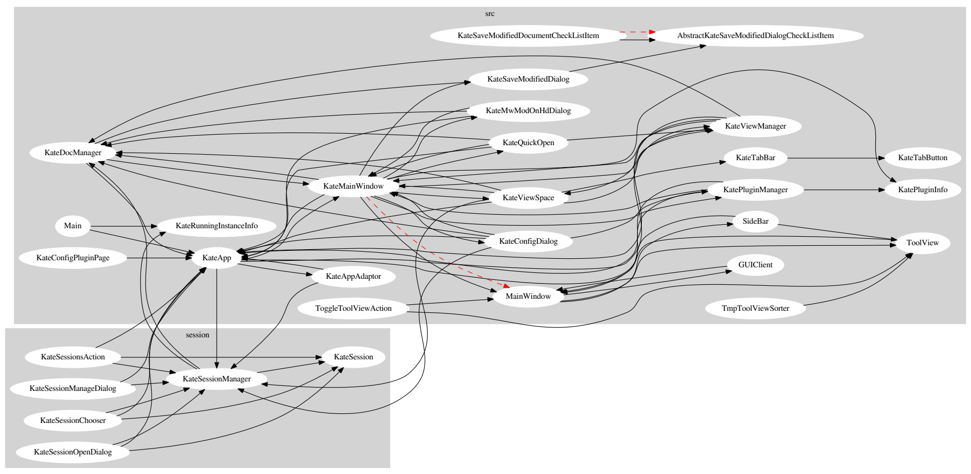


Fig. 2: Kate's original unweighted modularization, where black arrows represent function calls and red arrows represent inheritance.

discussion in the complementary Technical Report [10]. We also make available Kate's modularization data at `www0.cs.ucl.ac.uk/staff/m.paixao/kateMod/`, to support replication and further studies. Finally, the multi-objective software module clustering tool we developed for this work will be made available in the near future. We pose and answer three research questions, which occupy the remainder of this paper.

RQ$_1$: How much Kate's modularization can be improved for the unweighted and weighted MDGs? Table 1 presents the results for both MCA and ECA approaches for the unweighted and weighted MDGs in comparison to Kate's original modularization. In case of statistical difference between MCA and ECA, the value is highlighted and the effect size is presented. As one can see, both multi-objective approaches were able to find solutions with better quality metrics than the original modularization for the two different datasets. Regarding cohesion, coupling and MQ, MCA could improve such metrics in 16.3%, 83% and 7.88% for unweighted data, and 3.93%, 46.8% and 70.41% for weighted data, respectively. Considering ECA, these values are similar, 16.4%, 83.7% and 1.65% for unweighted, and 3.49%, 41.57% and 60.35% for weighted.

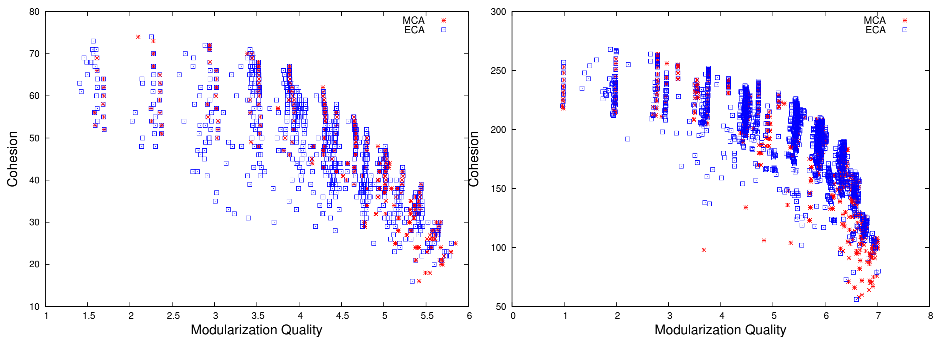
For the other quality metrics, MCA and ECA also presented similar results for both datasets, which suggests that these two different approaches did not find very different results for this case study. In fact, almost no statistical difference was detected between MCA and ECA, as can be visually seen in the plots of the solutions found in Figure 3.

RQ$_2$: What difference do omnipresent modules make?

For almost all systems, there is usually a subset of modules that have more dependencies than the average. These modules have been called omnipresent [11] because they belong to the whole system, rather than to a single cluster.

Table 1: Quality metrics results for the unweighted and weighted MDGs in comparison to Kate's original modularization

	Fitness	**Kate's Original**	**MCA**	**ECA**	**Effect Size**
Unweighted	Cohesion	51	59.30 ± 1.10	59.37 ± 1.08	-
	Coupling	10	1.70 ± 1.10	1.63 ± 1.08	-
	Number of Clusters	2	2.57 ± 0.92	2.37 ± 0.87	-
	MQ	1.308	1.42 ± 0.28	1.33 ± 0.36	-
	Isolated Clusters	0	0.53 ± 0.76	-	-
	Difference Modules	11	-	14.03 ± 7.79	-
Weighted	Cohesion	250	259.83 ± 4.62	258.73 ± 5.23	-
	Coupling	21	11.17 ± 4.62	12.27 ± 5.23	-
	Number of Clusters	2	5.90 ± 1.04	**6.97 ± 1.54**	0.22
	MQ	1.69	2.88 ± 0.46	2.71 ± 0.55	-
	Isolated Clusters	0	2.27 ± 1.26	-	-
	Difference Modules	19	-	21.23 ± 2.03	-



(a) Unweighted MDG (b) Weighted MDG

Fig. 3: MCA and ECA solutions location for the unweighted and weighted data

Based on previous works [11], omnipresent modules were handled using thresholds. By choosing an omnipresent threshold $o_t = 3$, for example, modules that have 3 times more dependencies than the average are considered omnipresent. Two different thresholds were used in this work, $o_t = 3$ and $o_t = 2$. After identified, the omnipresent modules are isolated from the MDG, and not considered during the optimization process. Because the results for both unweighted and weighted datasets when considering omnipresent modules are similar, only the unweighted results will be discussed. Table 2 presents such results.

For both $o_t = 3$ and $o_t = 2$, the improvements for cohesion and coupling were small. However, since the MQ metric had improvements of 151.5% and 182.8%, and the number of clusters was much bigger, a better overall modularization was achieved. Both approaches had almost the same performance for both thresholds, for almost no statistical difference was detected.

As an answer to RQ_2, there is nearly no difference in the behavior of the multi-objective approach when omnipresent modules are considered. It tends to improve all metrics, with both MCA and ECA presenting similar results. How-

Table 2: Quality metrics results for the unweighted dataset and different thresholds for omnipresent modules

	Fitness	**Kate's Original**	**MCA**	**ECA**	**Effect Size**
$o_t = 3$	Cohesion	34	35.60 ± 1.36	35.47 ± 1.54	-
	Coupling	5	3.40 ± 1.36	3.53 ± 1.54	-
	Number of Clusters	2	5.07 ± 1.44	4.77 ± 1.69	-
	MQ	1.32	3.32 ± 1.02	3.11 ± 1.18	-
	Isolated Clusters	0	0.27 ± 0.44	-	-
	Difference Modules	16	-	12.63 ± 4.03	-
$o_t = 2$	Cohesion	29	27.20 ± 0.95	27.67 ± 0.91	-
	Coupling	0	1.80 ± 0.95	1.33 ± 0.91	-
	Number of Clusters	2	**5.70 ± 1.04**	4.17 ± 1.75	0.73
	MQ	1.40	**3.96 ± 0.69**	2.93 ± 1.17	0.76
	Isolated Clusters	0	0.00 ± 0.00	-	-
	Difference Modules	17	-	6.03 ± 2.99	-

ever, the magnitude of the improvement is smaller. This might happen because the isolation of omnipresent modules reduces the search space, making the original solution closer to the optimal.

RQ_3: Can the multi-objective module clustering provide useful advice?

Figure 4 presents an example of solution found for the unweighted dataset and omnipresent threshold $o_t = 2$. Despite not being shown in this paper, the solutions for the other scenarios achieved similar modularization. We can see that this solution does capture intuitive clustering of functionality (even though it is computed structurally with no knowledge of purpose of intent). For instance, 'Tool', 'Plugins' and 'Tab' all appear to be related, and they were clustered together by the SBSE approach. Also the 'session' cluster, which appeared to make some sense in the original clustering, has been retained by the algorithm.

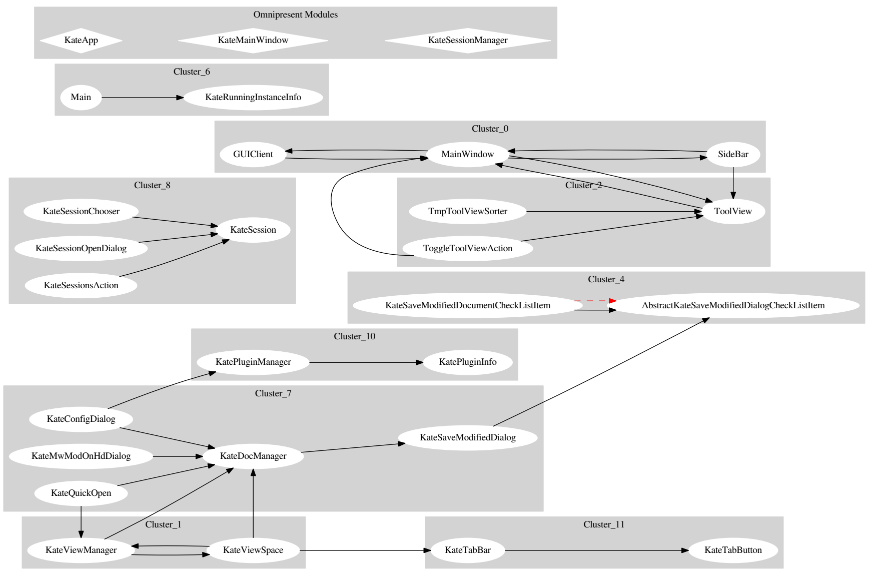


Fig. 4: Example of solution generated for the unweighted dataset and $o_t = 2$

3 Conclusion and Future Works

This paper demonstrated that, by applying a multi-objective module clustering approach, it was possible to improve Kate's original modularization for several quality metrics. The optimization technique had basically the same performance for both unweighted and weighted datasets, as well as considering omnipresent modules. The generated solutions were also able to provide useful advice about Kate's modularization. As future research directions, it is expected to apply the multi-objective module clustering approach to other systems.

References

1. Kate. `http://kate-editor.org/` (2015) Accessed in April, 2015.
2. Mancoridis, S., Mitchell, B.S., Rorres, C., Chen, Y.F., Gansner, E.R.: Using automatic clustering to produce high-level system organizations of source code. In: IWPC. Volume 98., Citeseer (1998) 45–52
3. Praditwong, K., Harman, M., Yao, X.: Software module clustering as a multi-objective search problem. Software Engineering, IEEE Transactions on **37**(2) (2011) 264–282
4. Mahdavi, K., Harman, M., Hierons, R.M.: A multiple hill climbing approach to software module clustering. In: Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on, IEEE (2003) 315–324
5. Doxygen. `http://www.stack.nl/~dimitri/doxygen/index.html` (2015) Accessed in April, 2015.
6. Praditwong, K., Yao, X.: A new multi-objective evolutionary optimisation algorithm: the two-archive algorithm. In: Computational Intelligence and Security, 2006 International Conference on. Volume 1., IEEE (2006) 286–291
7. Arcuri, A., Briand, L.: A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Software Testing, Verification and Reliability **24**(3) (2014) 219–250
8. Harman, M., McMinn, P., Souza, J.T., Yoo, S.: Search based software engineering: Techniques, taxonomy, tutorial. In: Empirical Software Engineering and Verification. Springer (2012) 1–59
9. Neumann, G., Swan, J., Harman, M., Clark, J.A.: The executable experimental template pattern for the systematic comparison of metaheuristics. In: Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion, ACM (2014) 1427–1430
10. Paixao, M., Harman, M., Zhang, Y.: Improving the module clustering of a c/c++ editor using a multi-objective genetic algorithm. RN **15**(02) (2015) 01
11. Mancoridis, S., Mitchell, B.S., Chen, Y., Gansner, E.R.: Bunch: A clustering tool for the recovery and maintenance of software system structures. In: Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on, IEEE (1999) 50–59