Research Note
RN/15/02

Improving the Module Clustering of a C/C++ Editor using a Multi-objective Genetic Algorithm

May 5, 2015

Matheus Paixao[1], Mark Harman[1], Yuanyuan Zhang[1]

Affiliation: University College London[1]

email: {matheus.paixao.14, mark.harman, yuanyuan.zhang}@ucl.ac.uk

Abstract

This Technical Report applies multi-objective search based software remodularization to a C/C++ editor called 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

Software systems are usually divided and organized in modules, such as packages, classes, functions etc. Software Module Clustering consists in organizing the system's modules into clusters, in order to achieve some desired system structure. According to several software engineering textbooks, well-modularized systems are easier to maintain, develop and evolve [15][14][17]. Usually, a good modularization presents a high degree of cohesion and a low degree of coupling between the modules [17], alongside with other metrics [2].

Although a good modularization is constantly desired throughout the whole system's life cycle, metrics of cohesion and coupling, as well as other quality metrics, tend to degrade as the software evolves [6]. Furthermore, regarding the specific domain of IDEs/Text Editors, which the system under study in this work is part of, the extensive empirical study in [16] provides evidence of quality degradation during system's evolution. Therefore, for most systems, there is a certain point in time (or more than one) where a re-modularization is needed in order to improve both the system's current comprehensibility and future maintainability and evolvability.

As one can expect, module clustering is not an easy task. This is mostly due to the high number of modules the system usually has at this refactoring phase, which results in a big set of possible solutions to consider. Even skillful developers may not be able to identify all the modules that should be included in a certain cluster, or not recognize quality trade-offs between clusters, for example.

Search-based approaches to module clustering were first introduced in the seminal paper by Mancoridis et al. [9], aiming to automatically re-organize the system's modules into existing or new clusters. A Module Dependency Graph (MDG) is used to represent the modules, and this graph can be either unweighted or weighted. In the unweighted MDG, an edge between two modules denotes a dependency between the respective modules. For a weighted graph, an edge between modules represents not only the existence of a dependency, but also the strength of this dependency, where the greater the edge weight, the greater the dependency [7]. By using the MDG, the optimization algorithm can then search for a partition of this graph that optimizes the considered quality metrics.

For almost all systems, there is usually a subset of modules that have more dependencies than the average. These modules have been called omnipresent [8] because it seems they do not belong to any particular cluster, but rather to the whole system. Because omnipresent modules have dependencies to a large number of modules in the system, they can affect the search-based module clustering.

The clustering process is based on the cohesion and coupling metrics, which are considered the fitness functions of the search algorithm. Most of the previous works in search-based module clustering, e.g. [9][8][7], employ a single objective approach, where the cohesion and coupling metrics are combined into one fitness function. Despite the single objective approach being able to find good results [10], it fails to present trade-off analyses between the different objectives.

A multi-objective module clustering approach was presented in [12], and it aims to support the developer decision making by providing a wide range of possible solutions, rather than only one. Since most of the metrics used in module clustering are not measured in a comparable unit, a multi-objective optimization process allows the developer to assess several trade-offs between quality metrics. The multi-objective approach handles the different quality metrics using the Pareto optimality concept [3], which tries to find not only one optimal solution, but a set of non-dominated solutions, i.e., solutions that are equally optimal. When compared to the single objective approach, the multi-objective one has found solutions that are better even for the combined fitness function used by the previous works.

This work adapts and applies the multi-objective module clustering approach presented in

[12] to Kate [5], a C/C++ editor for KDE platforms. Both unweighted and weighted datasets were extracted from Kate. Omnipresent modules were also considered in different ways. The objectives of this work are summarized in the following research questions:

- **RQ$_1$**: How much Kate's modularization can be improved regarding quality metrics previously used in the module clustering literature?
- **RQ$_2$**: What is the difference between the results for Kate's unweighted and weighted datasets?
- **RQ$_3$**: What is the difference in the results when omnipresent modules are considered?
- **RQ$_4$**: Can the multi-objective optimization process provide useful advice to the developer?

The rest of this technical report is organized as follows: Section 2 presents the multi-objective module clustering formulation used in this work, while Section 3 presents how the data used in the empirical evaluation was extracted. Section 4 presents the empirical study itself, showing the settings and analysis of results. Section 5 concludes and points out some future research directions.

2 Multi-objective Module Clustering Formulation

Consider the system's set of modules $M = \{m_1, m_2, \ldots, m_A\}$, where A is the number of modules in the system. The set of possible clusters is represented by $C = \{c_1, c_2, \ldots, c_B\}$, where B is the number of clusters, and each cluster has its unique number $1, 2, \ldots, B$. A possible solution for the module clustering problem is defined by the decision variables $X = \{x_1, x_2, \ldots, x_A\}$, where $x_i = c$ indicates that module m_i belongs to cluster c.

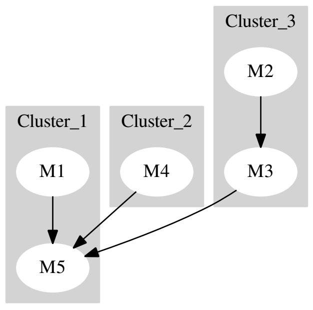


Figure 1: Modularization Example

A simple solution $X = \{1, 3, 3, 2, 1\}$, for example, denotes a modularization of 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 unweighted MDG for such example can be seen in Figure 1.

This work considers two different multi-objective module clustering approaches, named as Maximizing Cluster Approach (MCA) and Equal-size Cluster Approach (ECA) [12]. The set of fitness functions considered by the two approaches are presented next:

- **Maximizing Cluster Approach**
  - cohesion (max)
  - coupling (min)
  - number of clusters (max)
  - MQ (max)
  - number of isolated clusters (min)

- **Equal-size Cluster Approach**
  - 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 intra-edges, i.e., edges that start and finish in the same cluster. On the other hand, coupling is the sum of weights of all inter-edges, i.e., edges that start in a cluster and finish in another cluster. MQ stands for Modularization Quality [9], 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.*

3 Data Extraction

As stated earlier, this work adapts and applies the multi-objective module clustering formulation showed in the previous section to Kate [5], a C/C++ editor for KDE platforms. This section presents how Kate's modularization data was extracted.

Kate's source code is organized in only two folders, *src* and *session*, where each folder accommodates some classes. First, the call graph of each function of Kate was directly extracted from the source code using Doxygen [4]. Then, Doxygen was also used to extract the inheritance graph between classes. Finally, Kate's unweighted and weighted MDGs were created from the call and inheritance graphs, where each class is considered a module, and a function call or inheritance from one class to another represents a dependency between the respective modules. The weight of an edge in the weighted MDG is considered to be the number of functions calls from one class to another. For the unweighted MDG, all edges have the same weight of 1. The clusters are considered to be the folders the classes are in.

The original Kate's unweighted and weighted MDGs can be seen in Figures 2 and 3, respectively. Function calls are represented by black arrows and inheritance relationships are represented by red arrows. For the weighted MDG, the weight of the edge is represented by its thickness.

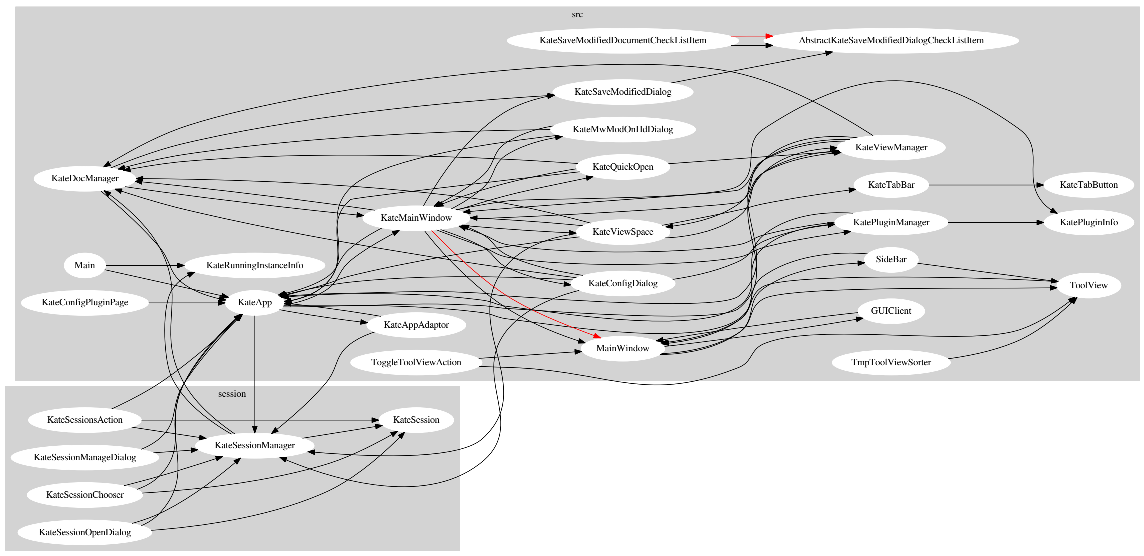


Figure 2: Kate's original unweighted modularization, where black arrows represent function calls and red arrows represent inheritance

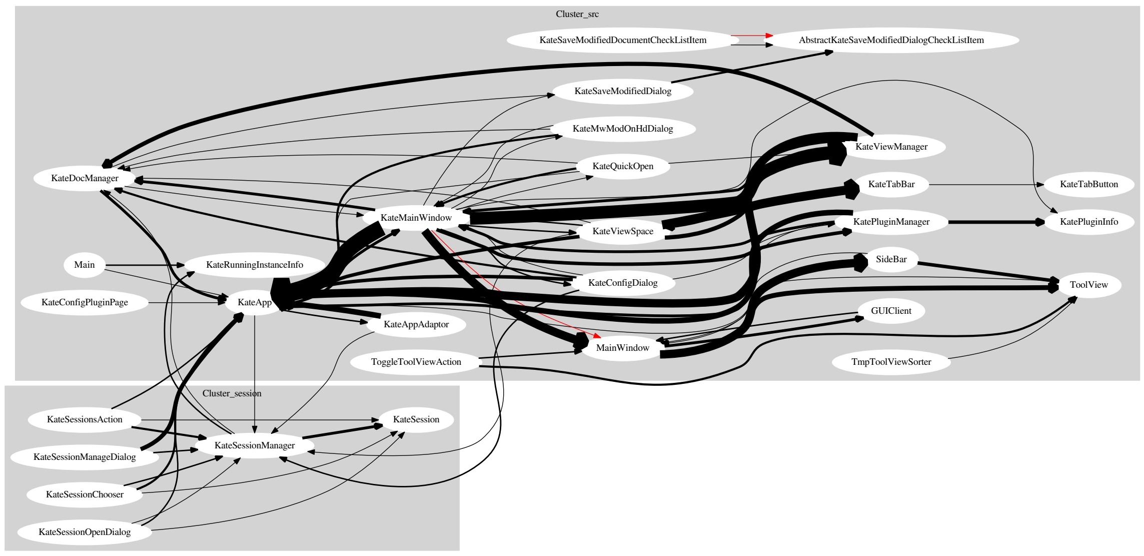


Figure 3: Kate's original weighted modularization, where black arrows represent function calls, red arrows represent inheritance and the thickness of an edge represents its weight

As one can see from the figures, Kate has only two clusters. The *session* cluster seems to be a cohesive one, where all modules are related to 'Session' functionalities. Furthermore, almost all external dependencies are related to one single module, KateApp. On the other hand, the *src* cluster presents a kind of 'god-class' structure, gathering all other modules. It is composed by modules related to different functionalities, varying from 'GUI' to 'Document Management' and 'Plugin Management'. Such organization suggests a not well modularized system as a whole, presenting room for optimization and improvement. The details and results of the empirical study are presented in the next section.

4 Empirical Study

The empirical study consisted in applying the multi-objective module clustering approach presented in Section 2 to Kate's modularization data presented in Section 3. This section presents the study settings and results.

4.1 Settings

The Two-Archive Genetic Algorithm [13] was employed. This algorithm presents good results for multi-objective problems with more than three fitness functions, which was the case in this work. The algorithm parameters were configured based both in [12] and the default algorithm settings. Crossover probability is 0.8, and mutation probability is $0.004 \log_2(M)$, where M is the number of modules. Population size is $10M$, and the algorithm is executed for 10000 generations.

Both MCA and ECA optimization approaches were executed 30 times, where each execution generates a set of non-dominated solutions. In order to compare the two approaches, one solution has to be chosen as a representative of each execution. This 'champion' solution can be selected in several ways, and for this work the solution with highest cohesion was selected. This set of representative solutions was then used to compute the average and standard deviations of each quality metric, as well as statistical comparisons. Statistical tests and effect size measurements were performed using the paired Wilcoxon and Vargha-Delaney tests, respectively, as suggested in [1]. Such tests were carried out using Astraiea, the tool for systematic comparison of metaheuristics described in [11].

Kate's modularization data is available in the supporting webpage `www0.cs.ucl.ac.uk/staff/m.paixao/kateMod/`. The tool implemented in this work for performing the multi-objective software module clustering will be available in the near future.

4.2 Results and Analysis

4.2.1 RQ_1: How much Kate's modularization can be improved regarding quality metrics previously used in the module clustering literature?

Table 1 presents the quality metrics results for both MCA and ECA approaches for the unweighted dataset 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. Regarding cohesion, coupling and MQ, MCA could improve such metrics in 16.3%, 83% and 7.88%, respectively. Considering ECA, these values are similar, 16.4%, 83.7% and 1.65%, respectively.

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

Fitness	Original Modularization	MCA	ECA	Effect Size
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	-

For the other quality metrics, both MCA and ECA presented similar results, which suggests that these two different approaches did not find very different results for this case study. In fact, none statistical difference was detected between MCA and ECA for all quality metrics.

Figure 4 presents the pareto fronts composed by all non-dominated solutions found in all 30 executions of both MCA and ECA for the unweighted dataset in the MQ × cohesion space. It is clear from the figure that the solutions found by the approaches are similar. Therefore, Figure 4 emphasizes the statistical tests, indicating almost no difference between MCA and ECA for this case study.

As an answer to the first research question RQ_1, the multi-objective module clustering approach can improve Kate's modularization for almost all considered quality metrics, reaching an improvement of 83% in a particular case. However, differently from previous works [12], it was not detected any statistical or solution location difference between MCA and ECA.

4.2.2 RQ_2: What is the difference between the results for Kate's unweighted and weighted datasets?

Table 2 presents the quality metrics results for both MCA and ECA approaches for the weighted dataset. Because of the weight in the dependencies edges, the value of the metrics tend to be

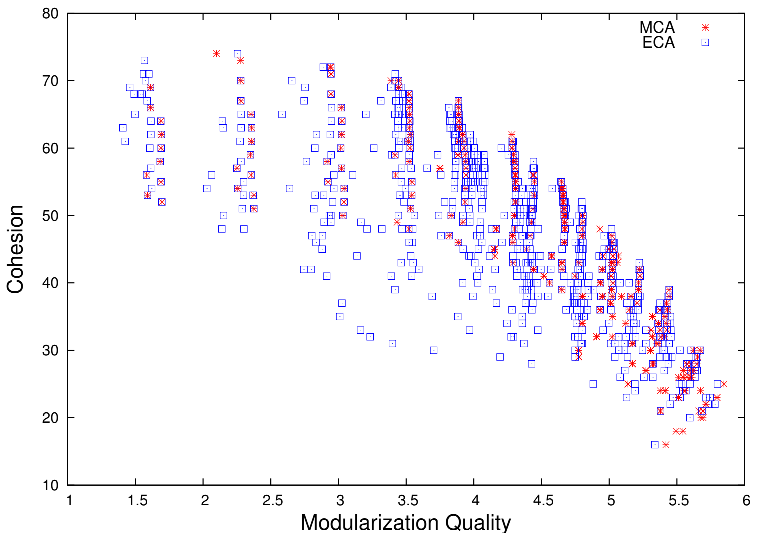


Figure 4: Solutions location for the unweighted dataset in the MQ $\times$ cohesion space

bigger. For the MCA, cohesion was improved in 3.93%, coupling in 46.8% and MQ in 70.41%. Regarding the ECA, the improvements were 3.49%, 41.57% and 60.35% for cohesion, coupling and MQ, respectively. The only statistical difference between MCA and ECA for the weighted dataset was regarding the number of clusters, but since the effect size is considerably small, the difference is not significant.

Table 2: Quality metrics results for the weighted dataset in comparison to Kate's original modularization

Fitness	Original Solution	MCA	ECA	Effect Size
Cohesion	250	259.83 $\pm$ 4.62	258.73 $\pm$ 5.23	-
Coupling	21	11.17 $\pm$ 4.62	12.27 $\pm$ 5.23	-
Number of Clusters	2	5.90 $\pm$ 1.04	**6.97** $\pm$ **1.54**	0.22
MQ	1.69	2.88 $\pm$ 0.46	2.71 $\pm$ 0.55	-
Isolated Clusters	0	2.27 $\pm$ 1.26	-	-
Difference Modules	19	-	21.23 $\pm$ 2.03	-

Similarly to the unweighted results, the multi-objective approach is able to improve almost all quality metrics for the weighted dataset. Furthermore, most of the results are not statistically different, as well as in the unweighted results. Although the quality metrics are generally improved for both unweighted and weighted datasets, the magnitude of the improvement for some metrics is different. Considering the unweighted dataset, cohesion had a big improvement and MQ had a small improvement. In the other hand, the results were the opposite for the weighted dataset, where cohesion had a small improvement and MQ had a big improvement. The improvement in coupling was considerably big for both datasets.

Figure 5 presents the pareto front of each multi-objective approach for the weighted dataset. Similarly to the unweighted pareto front, the solutions found by the two approaches are similar, emphasizing the nearly absence of statistical difference. In the case of the weighted dataset, as one can see from the figure, the number of solutions is bigger when compared to the unweighted dataset. This is due to the edges weighting, which creates a bigger search space, naturally increasing the number of non-dominated solutions.

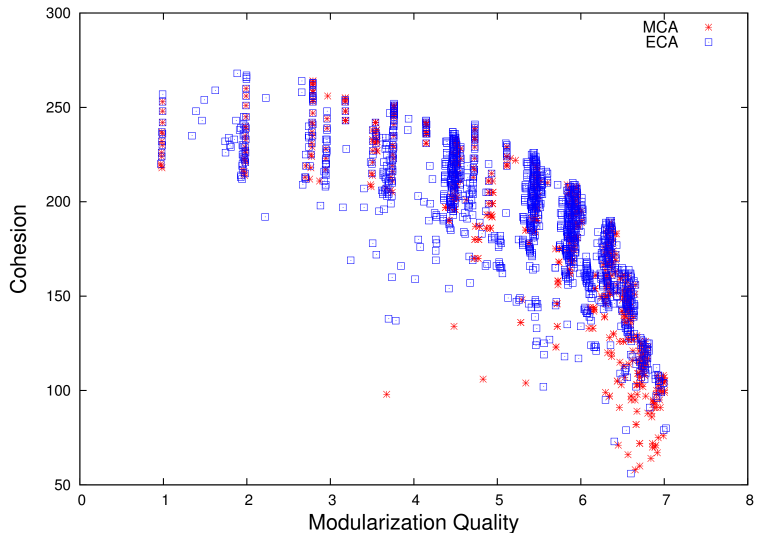


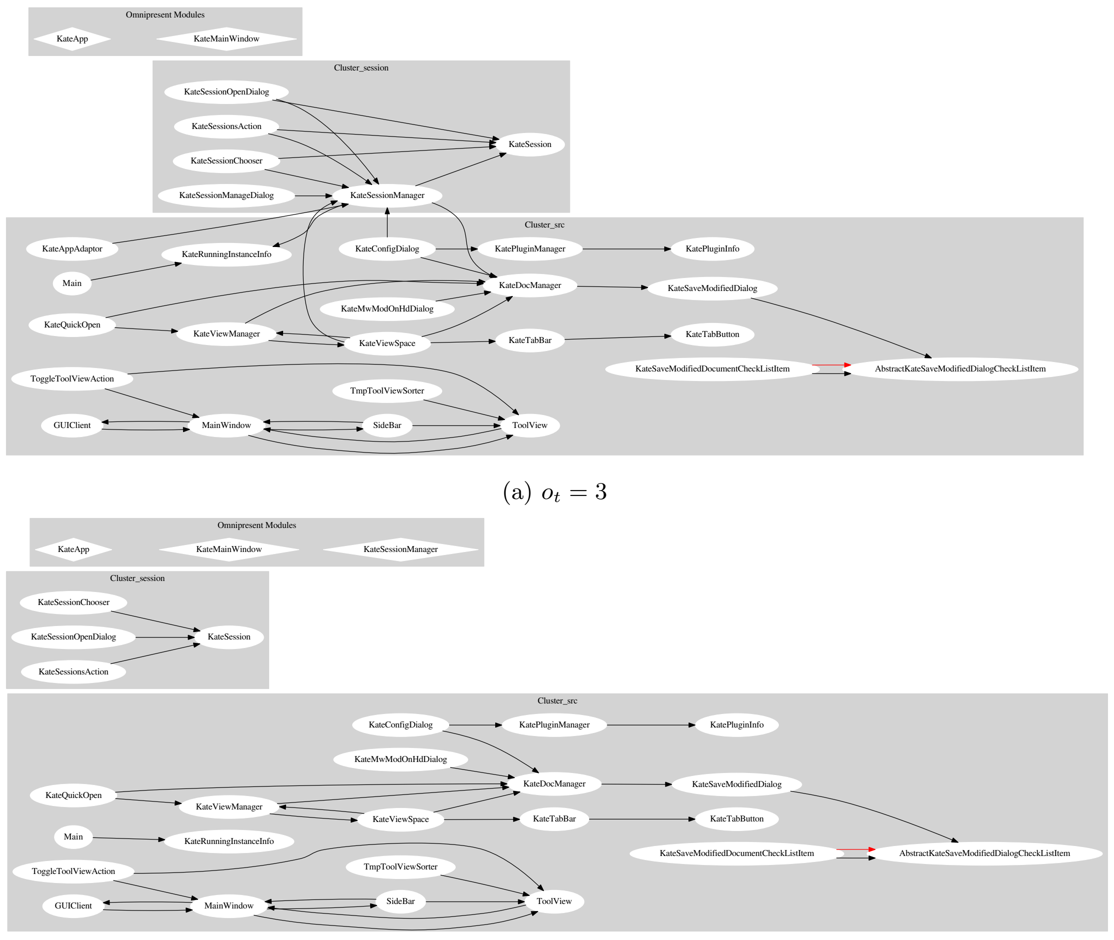
Figure 5: Solutions location for the weighted dataset in the MQ $\times$ cohesion space

As an answer to RQ_2, the general behavior of the multi-objective clustering approach when applied to both unweighted and weighted Kate's datasets is similar. Both MCA and ECA approaches can improve almost all Kate's original quality metrics for both unweighted and weighted datasets, with few statistical difference between results. In addition, the location of the non-dominated solutions found by the approaches are similar for both datasets. In contrast, the improvement of some metrics in the unweighted dataset was bigger than in the weighted dataset, and vice-versa. The weighted dataset also tends to present a bigger search space, leading to a bigger number of solutions in the pareto front.

4.2.3 RQ_3: What is the difference in the results when omnipresent modules are considered?

As stated earlier, there are usually some modules that have more dependencies than the average. Such modules are called omnipresent because they do not seem to belong to any particular cluster, but to the system as whole. Based on previous works [8], the omnipresent modules were identified using thresholds. By choosing an omnipresent threshold $o_t = 3$, for example, all modules that have 3 times more dependencies than the average are considered to be omnipresent. As smaller the threshold, more modules will be identified as omnipresent.

Two different thresholds were used in this work, $o_t = 3$ and $o_t = 2$. A threshold $o_t = 4$ did not identified any omnipresent module. After identified, the omnipresent modules are then isolated from the MDG, and the search algorithm will not consider them during the optimization process. Figure 6 presents the original unweighted Kate's modularization for the different thresholds and Table 3 presents the results for such datasets.



(a) $o_t = 3$

(b) $o_t = 2$

Figure 6: Kate's Original unweighted modularization with different thresholds

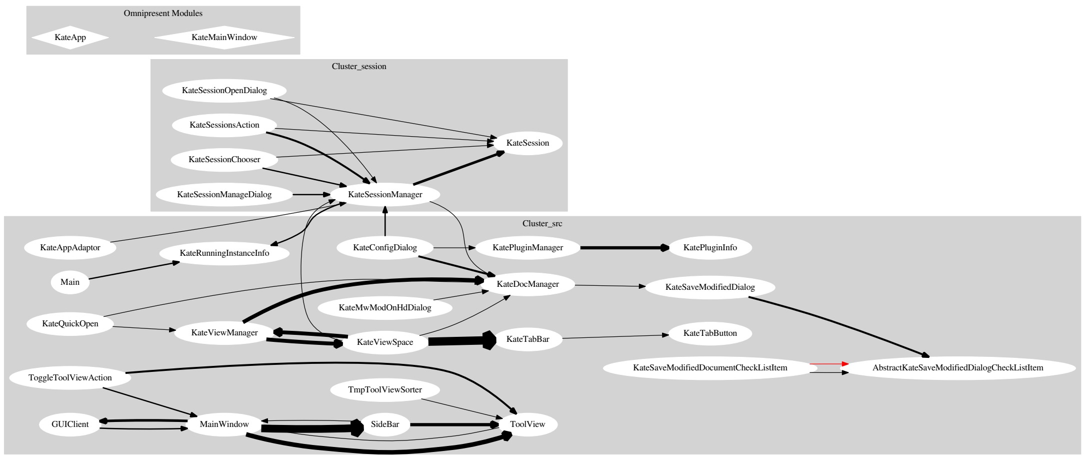
Table 3: Quality metrics results for the unweighted dataset and different thresholds for omnipresent modules

	Fitness	**Original Solution**	**MCA**	**ECA**	**Effect Size**
$\mathbf{o_t = 3}$	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	-
$\mathbf{o_t = 2}$	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	-

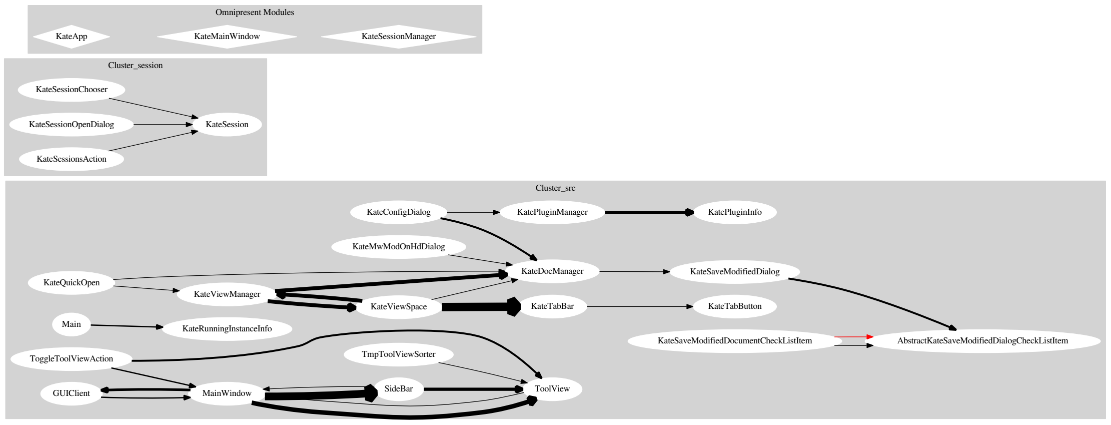
The number of overall dependencies is reduced when the omnipresent modules are isolated, so the fitness values tend to decrease. For an omnipresent threshold of $o_t = 3$, apart from the MQ metric, which had a improvement of 151.5%, the improvement for the other metrics are small. However, even cohesion having an improvement of only 4.7%, the number of clusters is much bigger, which suggests a better modularization. Both approaches had almost the same performance for this particular threshold, for no statistical difference was detected.

Considering the threshold $o_t = 2$, the metrics of cohesion and coupling were slightly decreased instead of improved, but since MQ had an improvement of 182.8% and the number of clusters is much bigger, the overall modularization is improved. The MCA approach achieved some statistically better results for some metrics with a considerable effect size.

Figure 7 presents the weighted Kate's modularization for the different thresholds, and Table 4 presents the results. Similarly to the unweighted results, for $o_t = 3$, almost all quality metrics are improved, with a small improvement for cohesion and coupling and a big improvement for MQ. The ECA approach found statistically different results for cohesion and coupling, however the effect size is not significant.



(a) $o_t = 3$

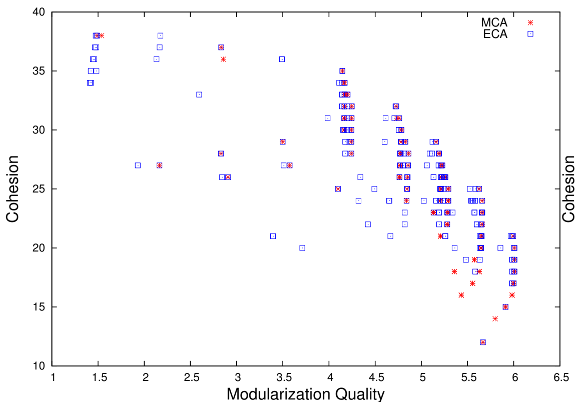


(b) $o_t = 2$

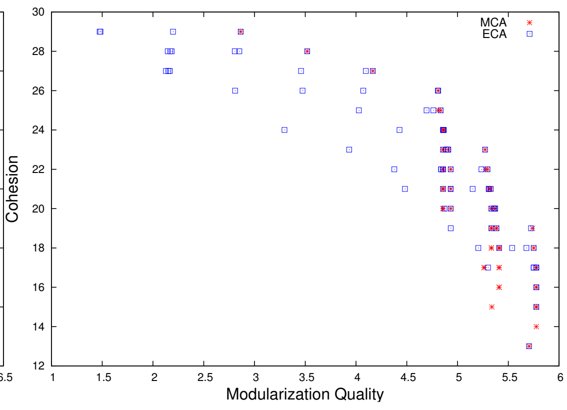
Figure 7: Kate's Original weighted modularization with different thresholds

Table 4: Quality metrics results for the weighted dataset and different thresholds for omnipresent modules

	Fitness	Original Solution	MCA	ECA	Effect Size
	Cohesion	105	106.13 ± 1.73	**107.07 ± 2.14**	0.37
	Coupling	7	5.87 ± 1.73	**4.93 ± 2.14**	0.62
$\mathbf{o_t} = \mathbf{3}$	Number of Clusters	2	5.37 ± 1.40	5.37 ± 1.62	-
	MQ	1.88	4.08 ± 1.07	4.12 ± 1.22	-
	Isolated Clusters	0	0.83 ± 0.82	-	-
	Difference Modules	16	-	11.93 ± 3.54	-
	Cohesion	93	90.90 ± 1.08	91.33 ± 0.70	-
	Coupling	0	2.10 ± 1.08	1.67 ± 0.70	-
$\mathbf{o_t} = \mathbf{2}$	Number of Clusters	2	**5.93 ± 1.18**	5.03 ± 1.52	0.66
	MQ	2	5.51 ± 0.80	4.95 ± 1.44	-
	Isolated Clusters	0	0.37 ± 0.60	-	-
	Difference Modules	17	-	6.83 ± 2.13	-



(a) $o_t = 3$



(b) $o_t = 2$

Figure 8: Solutions location for the unweighted dataset with different omnipresent thresholds

Regarding $o_t = 2$, the weighted results are also similar to the unweighted results, where cohesion and coupling metrics are slightly decreased. However, other metrics like MQ and number of clusters are highly improved, which generates a better modularization. In addition, MCA also found some statistically significant results when compared to ECA, and again with considerable effect size.

Figure 8 presents the location of the solutions in the pareto front for the unweighted dataset with different omnipresent thresholds. As one can see from the figure, the solutions found by the two approaches for both thresholds are similar, as expected from the almost absence of statistical difference in the results. The number of solutions when considering $o_t = 2$ is smaller than considering $o_t = 3$, which is itself smaller than not considering any omnipresent module at all. Such observation confirms the claim that isolating omnipresent modules can considerably reduce the search space [8].

Figure 9 presents the solution location for the weighted dataset and different omnipresent thresholds. For both thresholds, the solutions in the MCA and ECA pareto fronts are exactly

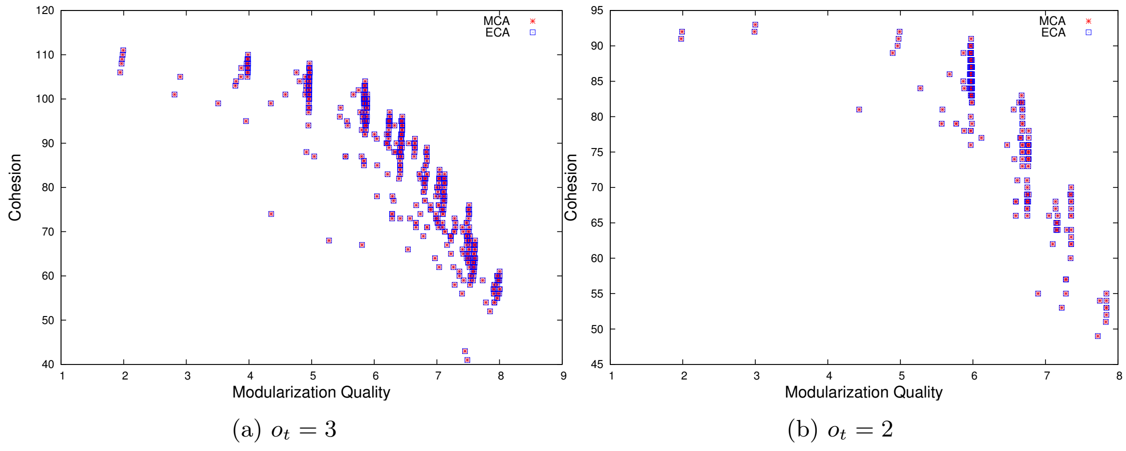


Figure 9: Solutions location for the weighted MDG with different omnipresent thresholds

the same, which accentuates even more the equality of the approaches for this particular system. The number of solutions is bigger when compared to the unweighted results because of the increase in the search space due to edges weighting, as previously stated.

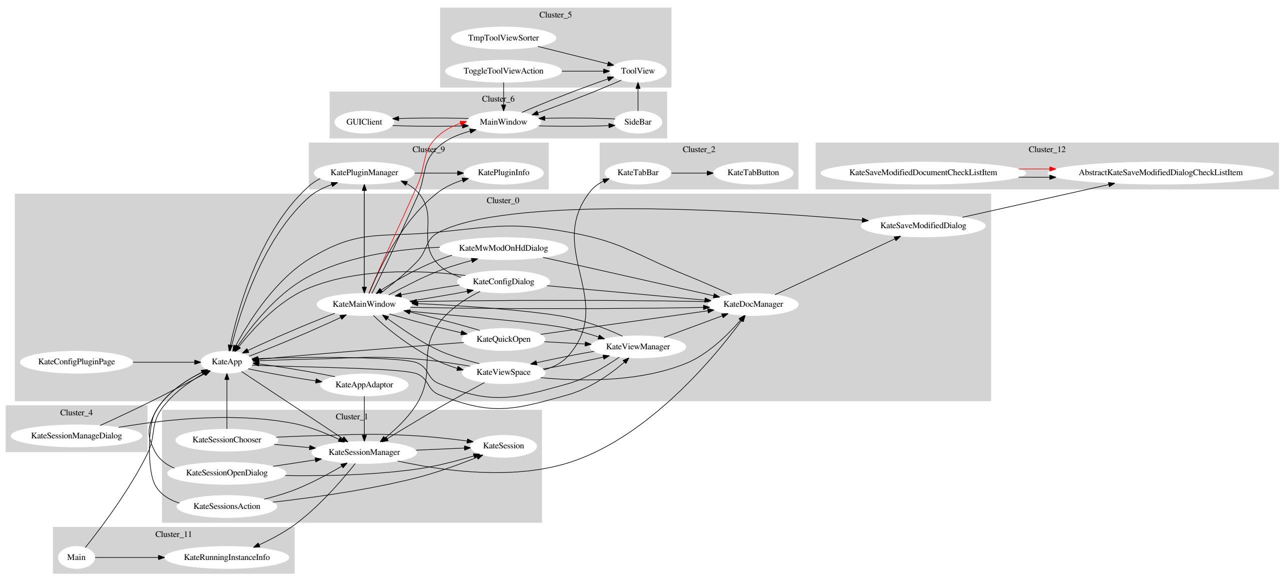
As an answer to RQ_3, the general behavior of the multi-objective approaches when considering omnipresent modules is basically the same as not considering omnipresent modules. It tends to improve all metrics, with both MCA and ECA presenting almost the same performance. The difference is in the magnitude of the improvement, which decays as more omnipresent modules are isolated. This might happen because the isolation of omnipresent modules reduces the search space, making the original solution closer to the optimal. The location of solutions found by MCA and ECA when considering omnipresent modules is similar, being even exactly the same for the weighted dataset and $o_t = 2$.

4.2.4 RQ_4: Can the multi-objective optimization process provide useful advice to the developer?

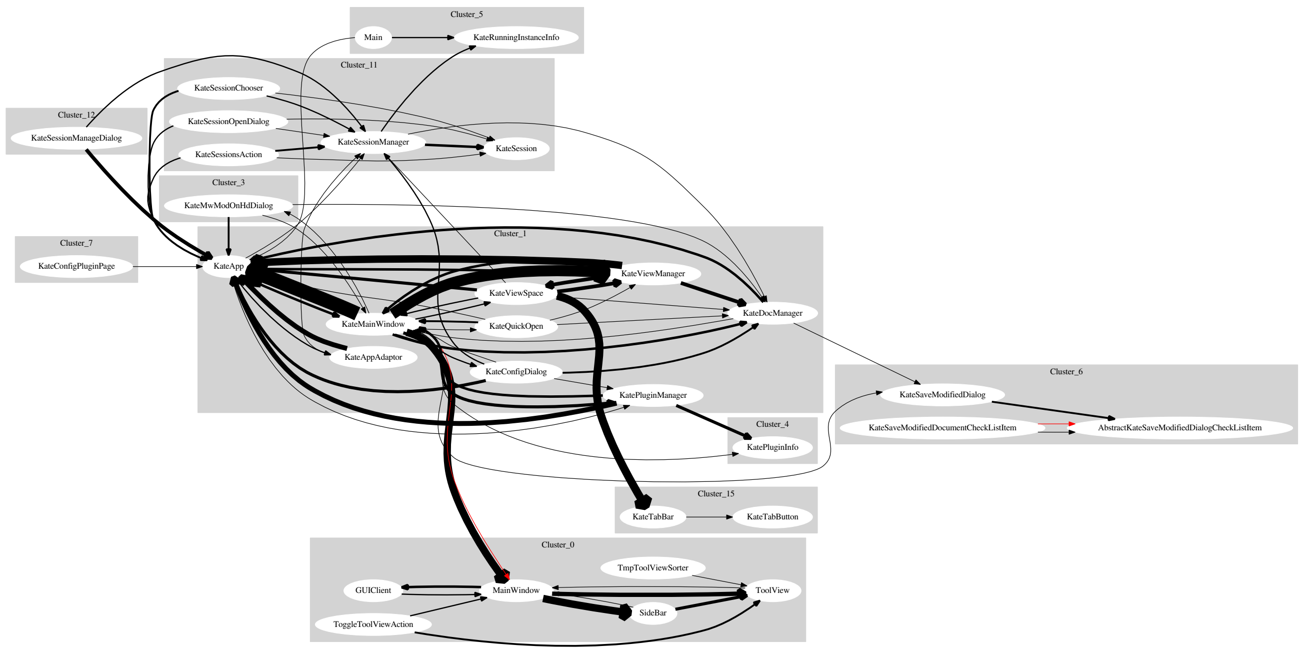
Although both MCA and ECA are able to improve Kate's modularization for almost all considered quality metrics in almost all scenarios, one may wonder whether the generated solutions are useful to the developer or not. In order to answer this question, examples of solutions found by the multi-objective approaches will be presented.

Figure 10 presents examples of solutions found for the unweighted and weighted datasets without considering omnipresent modules. In both cases, the solution does not appear good at a first sight, but a closer inspection can reveal some interesting insights. The 'session' cluster was kept almost the same in both examples, which suggests this cluster was already cohesive. All classes related to 'Tools' and 'Tabs' were also clustered together. Although some classes are correctly clustered, the presence of the omnipresent modules affects the optimization process, creating a big cluster with classes that are not related to each other.

Figure 11 presents examples of solutions found when considering an omnipresent threshold of $o_t = 3$. When the omnipresent modules are isolated from the clustering process, the solutions are much 'cleaner' than the previous ones. Several related classes were clustered together, both for the unweighted and weighted datasets. One can notice that the modularization is not 'optimal' in the sense that some classes are wrongly clustered, but because the modularization is more simple, the developer can easily identify these outliers and allocate them in the right cluster.

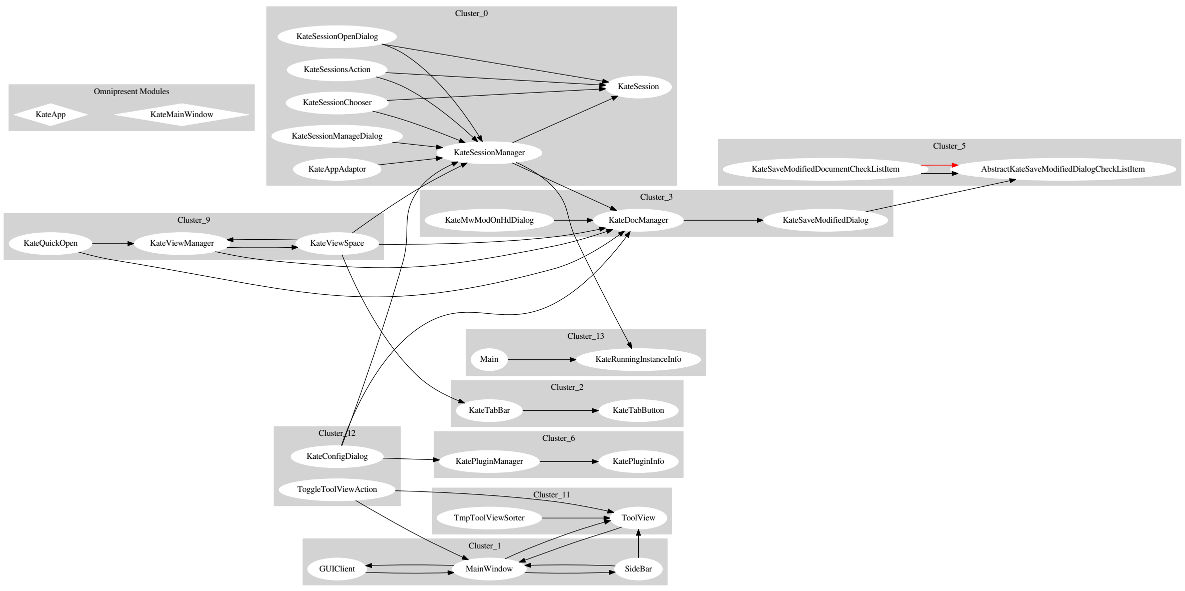


(a) Unweighted solution example and no omnipresent modules

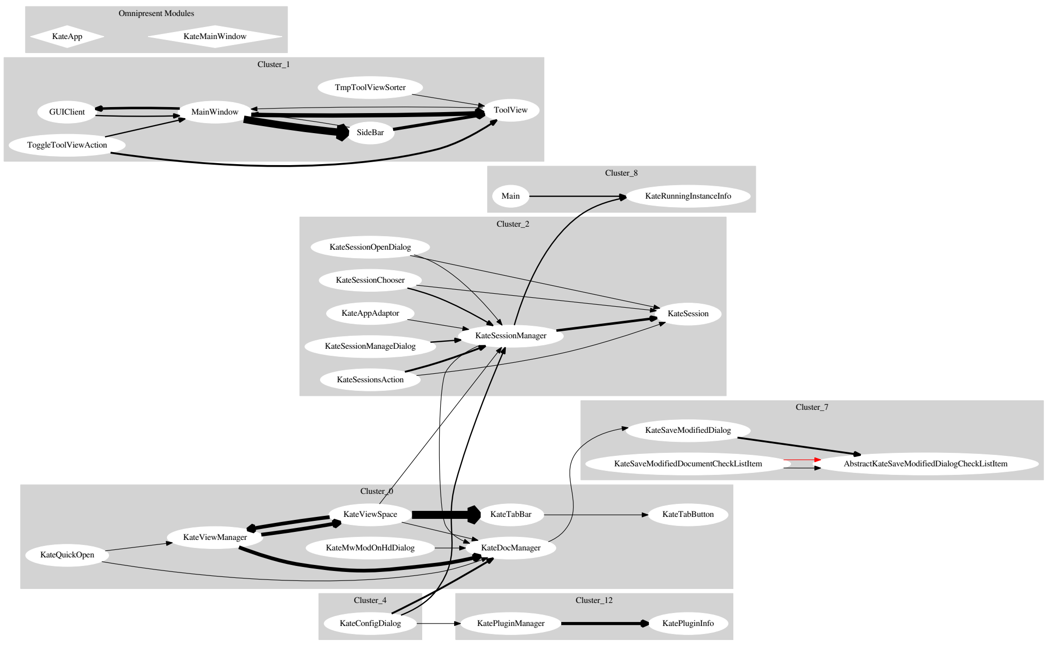


(b) Weighted solution example and no omnipresent modules

Figure 10: Example of solutions found for the unweighted and weighted datasets



(a) Unweighted solution example and $o_t = 3$



(b) Weighted solution example and $o_t = 3$

Figure 11: Example of solutions found when considering an omnipresent threshold of $o_t = 3$

Figure 12 now presents the solutions found for the threshold $o_t = 2$. Since more omnipresent modules were identified, and consequently isolated, the generated clusters are even 'cleaner', both for the unweighted and weighted datasets. For these particular solutions, the clusters were almost completely independent. One should notice the 'session' cluster was kept almost the same for all presented solutions, suggesting that the multi-objective clustering process tends to keep cluster that are already cohesive. Once again the solutions are not perfect, but they are easily adjustable by the developer.

Therefore, as an answer for RQ_4, the multi-objective optimization approach is able to provide useful advice to the developer. If the omnipresent modules are not isolated, the solutions will have some insights, but they will be hard to identify. By isolating the omnipresent modules, the multi-objective approach can cluster several related classes together, and as more omnipresent modules are identified, more independent the clusters tend to become. Furthermore, the optimization process tend to keep clusters that are already cohesive in the original solution.

5 Conclusion and Future Works

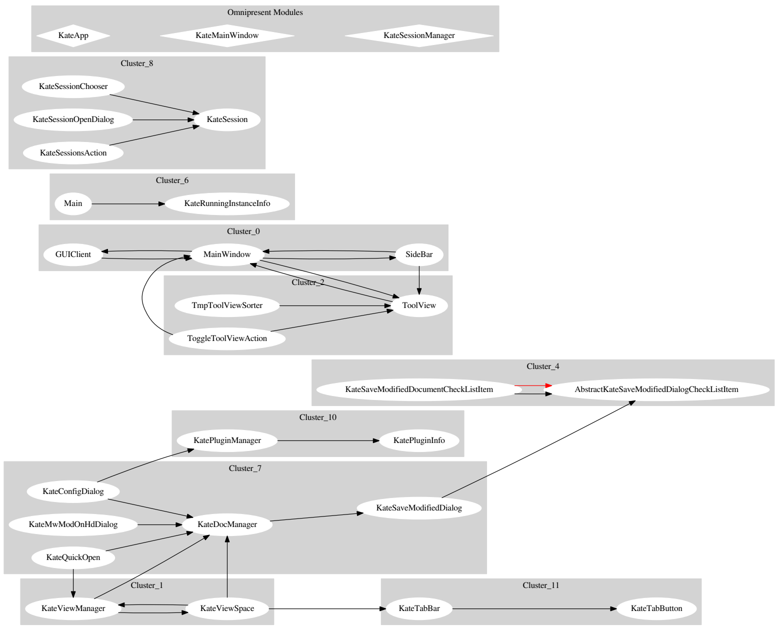
A good modularization of a software system throughout its whole life cycle is believed to make the system easier to comprehend, maintain and evolve. Sadly, modularization quality tends to degrade during systems' evolution, which creates space for the application of search based approaches to module clustering. Although single objective optimization approaches are able to achieve good results, they fail in provide a wide range of different solutions to the developer. Multi-objective approaches, by using pareto optimality, are able to present several different solutions to the developer, enabling trade-off analysis between the different quality metrics.

This work applied a multi-objective module clustering approach to a C/C++ editor called Kate. It's original modularization did not look cohesive, with only two clusters, and several classes with different purposes together in the same cluster. Different scenarios were considered, including unweighted and weighted datasets, as well as different thresholds for identifying omnipresent modules.

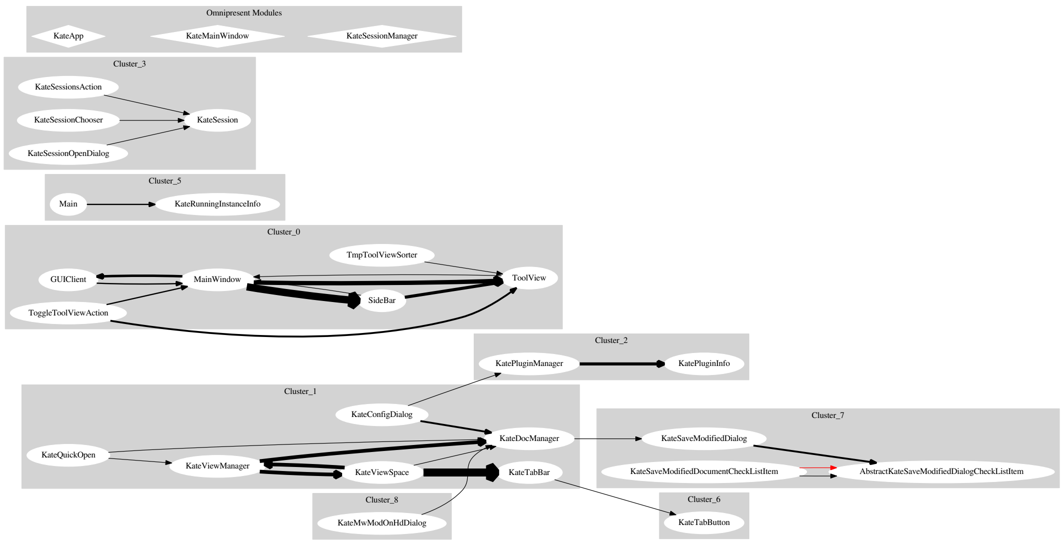
The multi-objective optimization process was able to improve Kate's modularization for almost all considered quality metrics and scenarios, including improvements of more than 200% in some cases. The behavior of the clustering techinique of improving the original metrics remained regular for all scenarios, but the magnitude of the improvements decreased as more omnipresent modules were considered.

By looking at the solutios found by the search algorithm, it was possible to notice that several related classes were put together in the same cluster, especially when the omnipresent modules were identified and isolated. In addition, the optimization algorithm tends to keep in the final solutions clusters that are already cohesive. Such solutions can provide insights to the developer, helping him/her to refactor the system in order to improve its modularization.

As future research directions, it is expected to apply the same optimization approach to other systems in order to assess for similarity in the results. Another future work is the usage of different multi-objective evolutionary algorithms in order to see if a better search algorithm would lead to better results.



(a) Unweighted solution example and $o_t = 2$



(b) Weighted solution example and $o_t = 2$

Figure 12: Example of solutions found when considering an omnipresent threshold of $o_t = 2$

References

[1] Andrea Arcuri and Lionel Briand. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability*, 24(3):219–250, 2014.

[2] Lionel C Briand, Sandro Morasca, and Victor R Basili. Property-based software engineering measurement. *Software Engineering, IEEE Transactions on*, 22(1):68–86, 1996.

[3] Kalyanmoy Deb. Multi-objective optimization. In *Search methodologies*, pages 403–449. Springer, 2014.

[4] Doxygen. `http://www.stack.nl/~dimitri/doxygen/index.html`, 2015. Accessed in April, 2015.

[5] Kate. `http://kate-editor.org/`, 2015. Accessed in April, 2015.

[6] Meir M Lehman. On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software*, 1:213–221, 1980.

[7] Kiarash Mahdavi, Mark Harman, and Robert M Hierons. A multiple hill climbing approach to software module clustering. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 315–324. IEEE, 2003.

[8] Spiros Mancoridis, Brian S Mitchell, Yihfarn Chen, and Emden R Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on*, pages 50–59. IEEE, 1999.

[9] Spiros Mancoridis, Brian S Mitchell, Chris Rorres, Yih-Farn Chen, and Emden R Gansner. Using automatic clustering to produce high-level system organizations of source code. In *IWPC*, volume 98, pages 45–52. Citeseer, 1998.

[10] Brian S Mitchell and Spiros Mancoridis. On the automatic modularization of software systems using the bunch tool. *Software Engineering, IEEE Transactions on*, 32(3):193–208, 2006.

[11] Geoffrey Neumann, Jerry Swan, Mark Harman, and John A Clark. The executable experimental template pattern for the systematic comparison of metaheuristics. In *Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion*, pages 1427–1430. ACM, 2014.

[12] Kata Praditwong, Mark Harman, and Xin Yao. Software module clustering as a multi-objective search problem. *Software Engineering, IEEE Transactions on*, 37(2):264–282, 2011.

[13] Kata Praditwong and Xin Yao. A new multi-objective evolutionary optimisation algorithm: the two-archive algorithm. In *Computational Intelligence and Security, 2006 International Conference on*, volume 1, pages 286–291. IEEE, 2006.

[14] Roger S Pressman. *Software engineering: a practitioner's approach.* Palgrave Macmillan, 2005.

[15] I. Sommerville. *Software Engineering.* Addison Wesley, 2011.

[16] Michel Wermelinger, Yijun Yu, Angela Lozano, and Andrea Capiluppi. Assessing architectural evolution: a case study. *Empirical Software Engineering*, 16(5):623–666, 2011.

[17] Edward Yourdon and Larry L Constantine. *Structured design: Fundamentals of a discipline of computer program and systems design*, volume 5. Prentice-Hall Englewood Cliffs, NJ, 1979.