Conceptual Level Comprehension Support of the Object-Oriented Programming Source-Code Using Kit-Build Concept Map

Nawras KHUDHUR[a*], Pedro Gabriel Fonteles FURTADO[a], Aryo PINANDITO[a], Shimpei MATSUMOTO[b], Yusuke HAYASHI[a] & Tsukasa HIRASHIMA[a]
[a]*Graduate School of Advanced Science and Engineering, Hiroshima University, Japan*
[b]*Faculty of Applied Information Science, Hiroshima Institute of Technology, Japan*
[*]nawras@lel.hiroshima-u.ac.jp

Abstract: Object-oriented programming (OOP) is a modern model of programming languages and an important module for many programming courses in academics. Not only do educators have trouble teaching OOP concepts but students are also reported to having trouble comprehending those concepts. The difficulty lies in dealing with abstract concepts and finding a relationship between the textbook explanations and the application of these concepts. Several works try to approach this problem, but they lack connecting the OOP concepts with its implementation in the source-code. In this research, we propose a new visualization form using concept maps to combine the OOP concepts with its' source-code to promote OOP concept comprehension. The proposed visualization is called the conceptual representation of the source-code (CRS). CRS unites the source-code statements and the OOP concepts into one comprehensible diagram. A concept map recomposition activity with Kit-Build is used to implement the CRS. We have conducted an experiment on university students to verify the learning effects and use of the proposed method. The results show a significant improvement in immediate learning by comparing before/after activity test-scores. In addition, students showed a positive impression and intention about using the tool during their studying of OOP by answering a questionnaire. The research findings shed light on a promising aspect of teaching OOP concepts in programming courses.

Keywords: Concept comprehension, conceptual level representation, concept map, object oriented programming, OOP concepts, kit build, concept visualizer.

1. Introduction

Computer programming is the main subject of study curriculum in computer-science related fields and even some high-schools that provide programming classes. Being a major part of computer programming, object-oriented programming (OOP) is a recent paradigm of programming languages. In OOP, programs consist of classes and objects. This structure is beneficial to divide problems into smaller pieces thus making the problem-solving more natural and the code reusable. OOP consists of several strongly interrelated concepts. Armstrong (2006) lists the concepts as object, class, method, message passing, inheritance, polymorphism, encapsulation, abstraction, instantiation, and modeling. Teaching these concepts and its comprehension in terms of actual coding is shown to be a difficult task for both educators and students.

The difficulty of OOP concept comprehension is identified in the literature (Kaczmarczyk et al., 2010; Sorva, 2018). Relational problems are also described where a study shows students find it difficult to comprehend the relationship among different concepts (Sajaniemi et al., 2008). Lack of active practice and suitable teaching tools are one of the reasons why it is hard to teach students about OOP concepts (Sarpong et al., 2013).

One familiar way to represent some of the OOP concepts is to use UML class diagrams. But in UML, the general explanations of the OOP concepts and how it is related to the source-code is not represented. Students are left with a code structure which is helpful but not enough to comprehend OOP concepts especially when students are less experienced with UML class diagrams (Gravino et al., 2015). In some courses, program visualizers (PV) are implemented as a tool to support OOP concept comprehension. PVs are tools that show the run-time behavior of a program when executed. Despite its usefulness, PVs could not fill the gap of OOP concept comprehension, mainly because of the low engagement structure of the PV and its limitation in visualizing OOP concepts and its' relationships (Sorva et al., 2013). Thus, educators need a tool to create conceptual level activities that can correlate the general OOP concept explanations with the actual source-code implementation effectively.

To approach this goal, we investigate a way to combine the OOP concepts and the source-code into one diagram using concept map (CMAP) (Novak, 2005). We call this combination representation *Conceptual Representation of The Source-code (CRS)*. Creating such a relational view between OOP concepts and its' actual use in source-code is not proposed before to the extent of our knowledge. By this combination, we aim to expose the learner to a productive view of the theory and practice of OOP concepts and encourage learners to interact with it actively. One issue with conventional CMAP is that each learner tends to construct the map for the targeted knowledge differently since the map depends on learners' conceptual understanding which can vary from learner to learner. One way to transform concept mapping into a more manageable and controllable yet effective activity for educators is using Kit-Build (KB) recomposition (Hirashima et al., 2015). KB is one type of CMAP that focuses on expert map recomposition instead of free map creation. In KB the learners are provided with a kit of concepts and links. Learners' goal is to recompose the CMAP in the same way the expert built it. This activity is called *concept map recomposition*.

In this paper, we investigate the possibility of using KB concept map to create an activity that unifies OOP source-code with its concepts i.e., implementing CRS. In addition, we aim to consider its impact on OOP concept comprehension.

2. Kit-Build Concept Map Recomposition

Concept map was first introduced by Novak (Novak, 2005) to evaluate students' conceptual learning and progress. In CMAP, concepts are expressed as nodes. These concepts are then connected to each other using labels to form a meaningful proposition. It is confirmed by many studies that it can promote the learning process in a variety of subjects and specialties (Wang and Chen, 2018; Balim, 2013).

CMAP comes in various forms such as scratch map (SM) and closed concept map (CCM) (Furtado et al., 2019). SM is a traditional concept mapping where learners start from an empty layout and build the concept map gradually. SM has unconstrained variability as it reflects each individual. In contrast, CCM provides a limited map building environment where learners are given a selected set of concepts and labels to choose from while building the map. However, the learners are free to make any proposition that they assume is valid using the provided set.

A more restricted type of CCM introduced by Hirashima et al. (2015) called Kit-Build (KB). KB asks learners to recompose the concept map instead of building it. By recompose, it means to re-connect a concept map from a kit of concepts and links of a pre-built concept map (expert map). The steps of a simple KB activity are as follows: 1) The expert creates a concept map for a material. 2) The expert concept map is then decomposed to its basic parts by removing the connections, thus creating a kit of concepts and links. 3) The kit is given to the learners to recompose it to the expert map. In KB, it is possible to have an exact map comparison between learners' map and the expert map since the same map pieces are used to make learner map and expert map is used as a reference. This comparison allows instructors to pinpoint the difficult parts of the lecture and give more accurate feedback to the learners (Sugihara et al., 2012). The validity and reliability of the KB diagnosis tool compared to the traditional map evaluation have been verified by past research (Wunnasri et al., 2018, 2017). KB also makes it possible to give automatic feedback to the learners while recomposing the map such as highlighting different propositions compared to the expert map. Another study showed that using expert map recomposition let the learners get a broader and deeper knowledge comprehension compared to scratch map building (Prasetya et al., 2021).

Despite of these many studies about concept maps and particularly KB, there are no investigations about using it in technical comprehension tasks such as to represent source-code and its concepts up to the authors' knowledge.

In this research, we used KB to implement CRS. The screenshot of the KB is shown in Figure 1. In KB, labels have two connectors colored red and blue, which appear only when the label is selected. The red connector refers to the source of the relationship, while the blue connector means the target of the relationship. Porpositions can be made by connecting these connectors to the concepts. Labels can make one-to-many relationships with concepts. Hence, the number of connectible targets is shown inside the blue connectors circle.

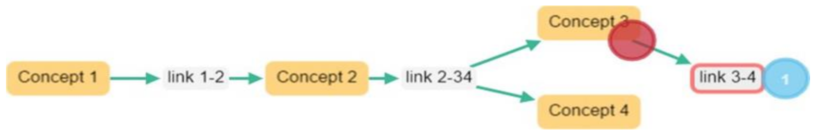

Figure 1. Kit-Build Example.

3. CRS Concept Map

The structure of an object-oriented (OO) source-code can be divided into two sections, internal and external structure. The internal structure refers to the statements of the source-code such as method/variable definitions and so on. It is explicitly visible to the learner i.e. the learner can just read through the source-code. Another feature of the internal structure is that it can go differently compared to another OO source-code, since the structure is the written text itself. In contrast, the external structure describes the OOP concepts such as inheritance, polymorphism, etc. These concepts are not directly visible in the OO source-code but its' implementation is realized in it. Moreover, the "**fact"**s of these OOP concepts are not dependent on the written text itself. In this sense, two different OO source-code can implement these OOP concepts in the same manner.

The objective of CRS concept map is to visualize both structures in one diagram and act as an intermediary between the two structures. Bridging these two structures allows the learner to connect the concepts of OOP to its' actual implementation in the source-code. Consequently, it can promote the conceptual interrelationships and how they affect each other. Another use of CRS is to use it as an evaluation map to measure the quality of an OO source-code by representing a given source-code in CRS and focus on what OOP concepts cannot be represented. This concludes that learners code does not implement these OOP concepts.

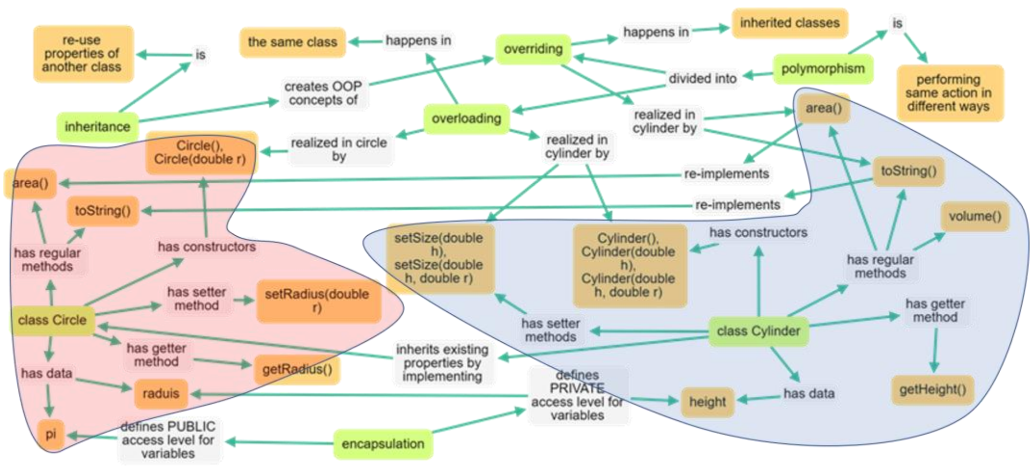

Figure 2. The Expert Goal-map for Learning OOP Concepts in a Source-code.

Figure 2 shows the proposed implementation of CRS concept map. The concept map is based on a source-code that implements a set of the OOP concepts[1]. The source-code consists of two classes Circle and Cylinder. Both classes contain multiple constructors. The Cylinder class inherits the Circle class and overrides two methods of Circle, namely area and toString. Concepts included in the source-code are inheritance, polymorphism, encapsulation, class, and composition. The red and blue

[1] https://git.io/JtKt6

regions in Figure 2 represents the internal structure of the class Circle and Cylinder respectively. These basic elements are the parts of the source-code that can be noticed easily by any learner, but it is not self-explainable. Several propositions are used to create the internal structure. These propositions act like annotations to support the fundamental source-code comprehension.

After grasping the fundamentals, the next target would be bridging it to the external structure of the source-code. The propositions outside the colored regions are the key propositions in CRS that act as a bridge to connect the major sections of the source-code to the OOP concepts. Hence, it enables the learner to foster the OOP concepts in the practical environment. The bridging propositions indicate what parts in the source-code represent the corresponding OOP concept. Moreover, CRS wraps up and provides the big picture of the implemented OOP concepts enticing the learner to track the interrelationship among different OOP concepts. To give an example, the CRS can tell why "overriding" can occur when inheritance is implemented, but it is not possible with a standalone class.

The CRS can be extended to target different goals. For instance, an "object" of type Cylinder can be added to CRS to reveal the access restrictions of an object toward different class variables and methods. It can also be modified to meet the understanding level of the students by adding more details of the OOP concepts when it is the first time to introduce OOP concepts.

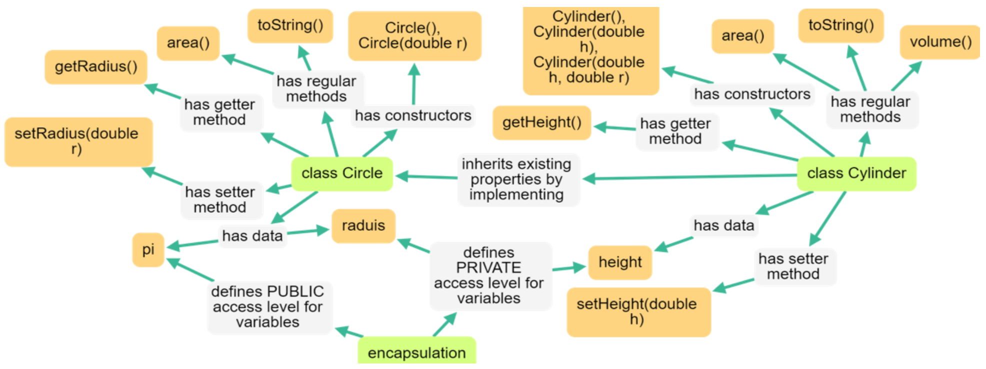

Figure 3. The Expert Goal-map Adopted in The Experiment.

4. Research Methodology

To investigate the effectiveness of CRS, a quasi-experimental design was utilized. Students performed a pre-test, recomposed the map, performed the post-test. At the end of the activity,students were asked to fill in a questionnaire about the activity.

4.1 Participants

The participants were 49 undergraduate third-year university students, majored in computer science. The experiment was conducted during their regular class and KB is used as a part of class-teaching material. Thus, we could not prepare a control group. Students were free to discontinue the experiment at any stage. Particularly, out of 49 students, 31 students completed the experiment and only the data for those 31 students were included in the analysis.

4.2 Materials

For this experiment, two materials were prepared. The first material was online lecture notes OOP concepts. The second material was a source-code explained in Section 3. An expert map implementing CRS was prepared which was about the realized OOP concepts in the given source-code. However, the map was simplified for this experiment to include fewer concepts and details, since the class time was very limited. The used expert map in the experiment is shown in Figure 3.

4.3 Procedure

The class instructor started the experiment by explaining the KB to the students and let them to build the training map to get familiar with the tool and its features. After that, students tested for their basic knowledge about the concepts of OOP given a source-code as a reference by answering multiple-choice questions about the concepts that were applied in the given source-code. This pre-test session lasted for 5 minutes. Afterward, material about the concepts of OOP given to the students to read and briefly explained by the instructor in 10 minutes. Then students were asked to recompose the kit using KB in 25 minutes. During concept map recomposition, students were allowed to look at the source-code. The KB tool had feedback feature to evaluate learners' map. The feedback reports the wrong propositions made by the learner that does not exist in the expert map. Students did a post-test afterward.

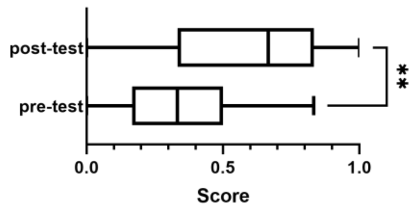

Figure 4. Comparison of Pre-test Score with Post-test Score.

4.4 Learning Effect of CRS Recomposition on OOP Concept Comprehension

We run the Fligner-Killeen test of homogeneity of variances to make sure there is no selection bias for students who decided to finish the experiment successfully. The result shows that the students were homogeneous with a p-value of 0.09639. To measure the learning outcome of using the CRS, we have compared the post-test scores against pre-test scores. The scores failed the Shapiro-Wilk normality test due to small number of participants. Thus, we used the Wilcoxon matched-pairs signed-rank test to measure the difference. Figure 4 shows the comparison results. The medians of pre-test score and post-test score were 0.33 and 0.66, respectively. A Wilcoxon Signed-rank test showed a significant difference between the post-test and pre-test scores (W=229, Z=-2.6754, p-value=0.006, r =0.346). The result suggests that using CRS recomposition can promote learning OOP concepts adequately.

4.5 Students' Feedback on Applying CRS in The Class

A questionnaire was given consisted of six 6-Likert scale questions to measure the students' likeness and expectation regarding the use of the CRS in learning OOP. Students showed a positive impression in using CRS, showing the average score for likeness and expectation 4.2 and 4.4 respectively. We can estimate that this activity was a little odd for students since it was different from their usual learning methods but still useful. The students' expectation for the CRS recomposition is high and positive. The students mostly agree that this activity will help them in learning OOP concepts effectively.

5. Concluding Remarks

In this study, we have presented a novel way to visualize OOP concepts and combine it to the OOP-based source-code. The post-test score results showed a significant improvement in students' OOP concept comprehension after the recomposition activity of the proposed concept map. Additionally, the questionnaire analysis of students' feedback shows that learning with the proposed activity is memorable and friendly as well as fun to some extent.

This study raises a new insight toward the perception of researchers and educators on OOP concept comprehension solutions coupled with source-code. The proposed approach has potential in various teaching aspects. It can be used in creating OOP-based activities to promote OOP comprehension. It also allows educators to create conceptual-based activities when teaching OOP.

One future work is to set a control group to compare it to other similar methods and to generalize the findings in this study. Another essential future work is considering the CRS as a source-code qualifier in terms of OOP concepts. It can be used by the learners to self-evaluate their source-code or by the teacher to group-evaluate the learners' source-code.

Acknowledgments

This research is partially supported by JSPS KAKENHI Grant Numbers 19H04227.

References

Armstrong, D. J. (2006). The quarks of object-oriented development. *Communications of the ACM*, 49(2), 123-128.

Balim, A. G. (2013). Use of technology-assisted techniques of mind mapping and concept mapping in science education: a constructivist study. *Irish Educational Studies*, 32(4), 437-456.

Furtado, P. G. F., Hirashima, T., & Hayashi, Y. (2019). Reducing cognitive load during closed concept map construction and consequences on reading comprehension and retention. *IEEE Transactions on Learning Technologies*, 12(3):402-412.

Gravino, C., Scanniello, G., & Tortora, G. (2015). Source-code comprehension tasks supported by UML design models: Results from a controlled experiment and a differentiated replication. *Journal of Visual Languages and Computing*, 28, 23-38.

Hirashima, T., Yamasaki, K., Fukuda, H., & Funaoi, H. (2015). Framework of kit-build concept map for automatic diagnosis and its preliminary use. *Research and Practice in Technology Enhanced Learning*, 10(1), 17.

Kaczmarczyk, L. C., Petrick, E. R., East, J. P., & Herman, G. L. (2010). Identifying student misconceptions of programming. *SIGCSE'10 - Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 107-111.

Novak, J. D. (2005). Results and implications of a 12-year longitudinal study of science concept learning. *Research in Science Education*, 35(1), 23-40.

Prasetya, D. D., Hirashima, T., And Hayashi, Y. (2021). Comparing two extended concept mapping approaches to investigate the distribution of students' achievements. *IEICE Transactions on Information and Systems*, E104.D(2):337-340.

Sajaniemi, J., Kuittinen, M., & Tikansalo, T. (2008). A study of the development of students' visualizations of program state during an elementary object-oriented programming course. *ACM Journal on Educational Resources in Computing*, 7(4).

Sarpong, K. A.-M., Arthur, J. K., and Amoako, P. Y. O. (2013). Causes of failure of students in computer programming courses: The teacher-learner perspective. *International Journal of Computer Applications*, 77(12).

Sorva, J. (2018). Misconceptions and the beginner programmer. *Computer Science Education: Perspectives on Teaching and Learning in School* (pp. 171-187). Bloomsbury Publishing.

Sorva, J., Karavirta, V., & Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education*, 13(4).

Sugihara, K., Osada, T., Hirashima, T., Funaoi, H., & Nakata, S. (2012). Experimental evaluation of kit-build concept map for science classes in an elementary school. *Proceedings of the 20th International Conference on Computers in Education*, ICCE 2012, 17-24.

Wang, S. P., & Chen, Y. L. (2018). Effects of multimodal learning analytics with concept maps on college students' vocabulary and reading performance. *Educational Technology and Society*, 21(4), 12-25.

Wunnasri, W., Pailai, J., Hayashi, Y., & Hirashima, T. (2017). Reliability investigation of automatic assessment of learner-build concept map with kit-build method by comparing with manual methods. *International Conference on Artificial Intelligence in Education*, 418-429.

Wunnasri, W., Pailai, J., Hayashi, Y., and Hirashima, T. (2018). Validity of kit-build method for assessment of learner-build map by comparing with manual methods. *IEICE Transactions on Information and Systems*, E101.D(4):1141-1150.

Wunnasri, W., Pailai, J., Hayashi, Y., & Hirashima, T. (2018). Validity of kit-build method for assessment of learner-build map by comparing with manual methods. *IEICE Transactions on Information and Systems*, E101D(4), 1141-1150.