On the Use of Concept Maps to Improve Student Skills in an Introductory Object-Oriented Analysis and Design Course

José F. Vélez
Depto. Informática y Estadística, Universidad Rey Juan Carlos, Madrid, Spain

A. Belén Moreno
Depto. Informática y Estadística, Universidad Rey Juan Carlos, Madrid, Spain

Victoria Ruiz-Parrado
Depto. Informática y Estadística, Universidad Rey Juan Carlos, Madrid, Spain

Ángel Sánchez
Depto. Informática y Estadística, Universidad Rey Juan Carlos, Madrid, Spain

Abstract

This paper presents an ongoing work on the application of concept maps to teaching an introductory Object-Oriented Analysis and Design course for Computer Science students. There exist previous works that introduce the concept map model in these object-oriented courses. However, these works do not usually go deeply enough into the transition from concept maps to static class diagrams. Although concept maps present some clear advantages when defining the abstractions present in object-oriented software modeling, some drawbacks may also appear if the transformation from these maps to class diagrams when the task is carried out through simplistic rules. In this paper we propose an approach, which is illustrated through a use case, to transition from a concept map to a static class diagram in a more realistic way.

2012 ACM Subject Classification Software and its engineering; Software and its engineering → Object oriented frameworks; Applied computing → Education

Keywords and phrases Object-Oriented Programming, Concept Map, Abstraction, Unified Modeling Language (UML), Static Class Diagram

Digital Object Identifier 10.4230/OASIcs.ICPEC.2025.3

Funding *Ángel Sánchez*: Gratefully acknowledge the financial support given by the Spanish MICINN Project: PID2021-124064OB-I00, and the URJC Educational Computing Project: PIE24_066.

1 Introduction

Courses on Object-Oriented (OO) Programming [1] are considered key components in most Computer Science curricula. As a part of an OO paradigm course, learning Object-Oriented Analysis and Design (OOAD) can be difficult for students, because it requires some abstract thinking skills and (usually) a shift from the (initially learned) procedural programming paradigm to the object-oriented one. Students must grasp and apply non-trivial concepts like encapsulation, inheritance, and polymorphism, and also learn to model real-world problems using objects/classes and their interactions. Moreover, OOAD involves creative problem-solving, where there is often no single correct solution, making it harder to assess and improve design choices without experience and practice. Beginners often struggle with complex terminology, connecting modeling diagrams (usually, the ones of the *Unified Modeling Language*), and object-oriented programming code. For all of this, many students fail when applying the key Object-Oriented principles correctly.

The Unified Modeling Language (UML) diagrams [6] are visual tools used to model software systems. They help software developers to analyze, design, program object-oriented solutions to computing problems, and also communicate how a system works. UML diagrams can be *static* (i.e., structure-focused) that represent the structural parts of an OO system, such as the classes and relationships, and *dynamic* diagrams (i.e., behavior-focused) that represent the behavior of an OO system over time, such as the modeling of user interactions and object workflows. These diagrams are useful for planning, documenting, and maintaining OO software. Among the static diagrams, class diagrams show the structure of a system without modeling its behavior. They represent hierarchies of classes and relationships between them (like inheritance, association or composition) which are shown using connecting lines, that can include cardinalities defining how many instances of one class can be associated with instances of another class in a relationship. Each class in a UML diagram contains its attributes and attached methods. In summary, these class diagrams focus on how classes are related, not how they interact during execution.

Concept maps [9][10] are visual tools used to organize and represent knowledge. They consist of nodes (representing concepts) and links (showing relationships between those concepts). These links are often labeled to describe the nature of the relationships, forming network-like structures. Concept maps are widely used in education to help students visualize and understand complex topics. Teachers apply them for lesson planning or for organizing curriculum content. In some workplaces, they are valuable for knowledge management and employee training. Researchers and professionals use them for problem-solving, brainstorming, and structuring information. These maps also useful in software design to model systems and analyze requirements.

This paper describes our work in progress on a proposed approach to integrate the use of concept maps as a tool to help Computer Science students in effectively developing static class UML diagrams.

2 Related work

Concept maps were introduced by Joseph D. Novak and his research team [11] as a mean of representing the emerging science knowledge of the students. Since then, many works have appeared to demonstrate the advantages of this graphical tool used to organize and represent knowledge in different higher education domains [13]. The psychological foundations of concept maps are rooted in cognitive psychology, particularly in how people organize, represent, and recall information. Concept maps are designed to represent the structure of human knowledge, and understanding their psychological basis is crucial for their effective use in learning and problem-solving [10].

More specifically, the effective use of concept maps in Engineering and Computer Science education has been widely researched [13] [15]. In the domain of Computer Programming education, Keppens and Hay [7] suggest that the introduction of concept maps help with the development and incremental refinement of mental models of programming concepts and promotes meaningful programming learning. We can also cite the work by Perin and collaborators [12], who performed an experimental study that demonstrated the efficacy of introducing concept maps as a metacognitive tool for enhancing students to learn of Entity and Relationship Diagrams (ERDs) in database modeling.

Different software tools have been developed to create and handle concept maps in an automatic form. One of the most well-known applications is *CMapTools* [2]. This is a software for creating, sharing, and collaborating on concept maps. It allows users to easily draw concept maps by dragging and dropping nodes, adding linking phrases, and other additional components. The tool supports real-time collaboration through shared servers and offers export options such as PDF, image files, and web pages. CmapTools is widely used in education, research, and knowledge management to visually represent knowledge structures. Dogan and Dikbıyık [3] have developed an adaptive intelligent web-based learning system, called *OPCOMITS*, that has a free domain model which can be regulated by an expert for any course. It uses a concept map model to regulate the topic hierarchy, to measure the learning of the students about course topics and to stimulate learning. Very recently, Diaz and Garmendia [4] have proposed incorporating annotation practices into concept mapping. Students could highlight text and link these highlights to existing or newly created concepts in their concept maps. This way, instructors can access both the concept map and the relevant readings for better feedback. This approach was implemented through the software *Concept-Go*, which is a plug-in for *CMaptools*.

One of the Computer Science disciplines where concept mapping has been more applied is Object-Oriented Analysis and Design (OOAD). Ferguson [5] studied how to create concept-map class diagram using UML for a better understanding of OO concepts. The work by Sanders and others [14] continues this approach and presented the results of an experiment to evaluate the OO concept understanding by introducing concept maps in their course. Sien [17] observed that the abstraction of problem domain into specific "objects" was an important difficulty for students when following an OOAD course. He describes a set of basic rules to transform a concept map into class (static) and sequence (dynamic) diagrams. In a later work, this same author [18] proposed teaching OO modeling as a set of threshold concepts and adopted concept maps as a mechanism to facilitate the development of class and sequence diagrams. Later, Shin [16] observed that many students attending OOAD courses encountered difficulties when transitioning from requirements analysis to OO design, and then to code implementation. This author performed a study demonstrating that integrating OOAD with concept maps can help the learners to understand the map-to-diagram transitions more clearly.

3 Proposed Approach

We describe our proposed approach to introduce concept maps in an OOAD introductory course. For this description we use a concept map example that was created from the statement of the following class exercise:

"A university library management application is to be designed. The application should allow to handle books, study rooms and users. Users can register and unregister, study rooms can be reserved according to a schedule, and books can be searched and lent according to some rules. Both study rooms and books are identified by a unique number. Each room can have different capacity. Additionally, for each book, some common data such as its author, title and ISBN, must be stored. The library has two types of users: teachers and students, respectively, who are also identified by a unique numeric code, and the system also stores their name and email. The main difference between these two user categories is the allowed loan period. There should also be an option to generate a listing of books that have not been returned and have an overdue loan date."

Fig. 1 illustrates the concept map created from this statement. We begin by identifying the main topic or theme of the exercise, and break it into key concepts, terms, and ideas related to the topic. We determine how these concepts are related (e.g., cause-effect, sequence, or hierarchy). Next, we aim to represent each concept, and connect them to other concepts

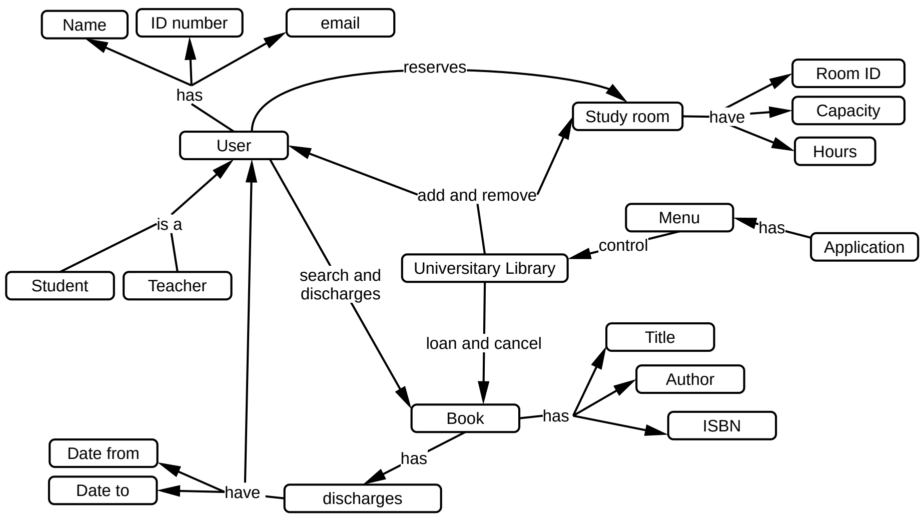

Figure 1 Concept Map for the University Library Application problem.

using labeled arrows to show their relationships. This map is arranged to flow logically, often from general to specific concepts. Finally, we review the map to ensure it accurately represents the content of the class exercise.

Next, we make some remarks on some aspects related to teaching our object-oriented course. When translating a concept map (created from the problem statement explained above) to a static class UML diagram, some considerations must be taken into account.

In particular, the nouns in a sentence correspond to concepts (or objects) in the map, that have types associated to them. These types may correspond either to classes predefined by the programming language (such as String or Integer) or to classes that must be ad-hoc defined. Concepts that have no connection with another concept usually correspond to classes predefined by the language and are therefore represented as attributes. For example, in the sentence: "The book has an author", the concept *book* is associated with a class *Book*, while the concept *author* is associated with the class *String* and we say that "author is an attribute of book".

On the other hand, the verbs in a statement can correspond to methods or to classes. When a verb has connections to several concepts it is usually represented as a class. If the verb is a "leaf" (in the concept map), it is usually a method. Also, when there are several verbs that apply to the same concept, it can be interesting to manage them as classes in a polymorphic scheme defined by a superior interface. These verbs can always be substantivized (e.g., "registration" of a book instead of "registering" a book).

Next, we describe the types of linguistic connectors in maps. First, the connectors of type "is a" or "is of type" usually give rise to *inheritance* relations. Connectors that imply "belonging to" or "control" of a concept over other ones in a permanent way, usually give rise to *association* relations or to *object attributes*. Examples of these connectors are: "has", "manages", ... Note that the cardinalities of the association relations are not specific to the concept map and they are added later in the class diagram. In addition, connectors involving "short temporal relationships" between concepts often give rise to *usage* relationships. Some examples of connectors are: "uses", "needs", "accesses", ...

The previous correspondences may be valid for constructing a class diagram of an initial OOAD solution, but this has many nuances that will be modified when constructing the class diagram during the implementation of a solution. In particular, we detected the following problems:

- The problem of *distribution of methods*, which is related to the principles of sole responsibility (SOLID) and the principle of the expert (GRASP) [8].
- The problem of the *introduction of polymorphism*, related to the principle of preferring polymorphism to type-based conditions.
- The problem of *unnecessary inheritance.*
- The problem of *multiplicity assignment.*

Next, we describe each of these mentioned problems.

3.1 The problem of distribution of methods

Students tend to place methods in classes that have a semantic relationship with the objects. However, they should take two other things into account with more relevance:

- The methods must be in classes that have sufficient information to implement them. This is clearly expressed in one of the GRASP principles: "A responsibility should only be assigned to a class if that class has the necessary information to fulfill it". For example, according to the concept diagram, the *user* searches for *books* in the *library.* It seems that the *search* method belongs to the *User* class. However, it must be placed in the *UniversityLibrary* class, since it is the one that has the information about the *books.* On the other hand, the *library* registers *books*, in this case it is correct, because the *UniversityLibrary* class has the necessary information to allow the method implementation.
- The classes should not have multiple responsibilities. For example, the *UniversityLibrary* class (in Fig. 2) is heavily loaded with methods. This will make the implementation of this class more complex and difficult to maintain.

According to the rules proposed previously, Fig. 2 shows the corresponding UML static class diagram corresponding to the concept map presented in Fig. 1. Note that the cardinalities in the class diagram are not obtained from the concept map but from the statement and context of the problem.

3.2 The problem of multiplicity assignment

A concept map does not address the problem of multiplicities. Perhaps one could force all concept map links to refer to such a concept using labels such as "many", "some", "at least one", ... However, this solution would complicate the diagram. It is necessary, when transforming a concept map into an equivalent class diagram, to carry out a detailed study of the multiplicity in each association relation that appears.

3.3 The problem of introduction of polymorphism

In general, students struggle to understand the concept of polymorphism. Furthermore, using a conceptual diagram does not clearly highlight the operations that should be grouped polymorphically.

In the library example, all operations in the class *UniversityLibrary* could be placed in different classes that implement an interface called *LibraryOperation.* Fig. 3 illustrates this scenario. In addition, this approach solves the single responsibility problem previously discussed.

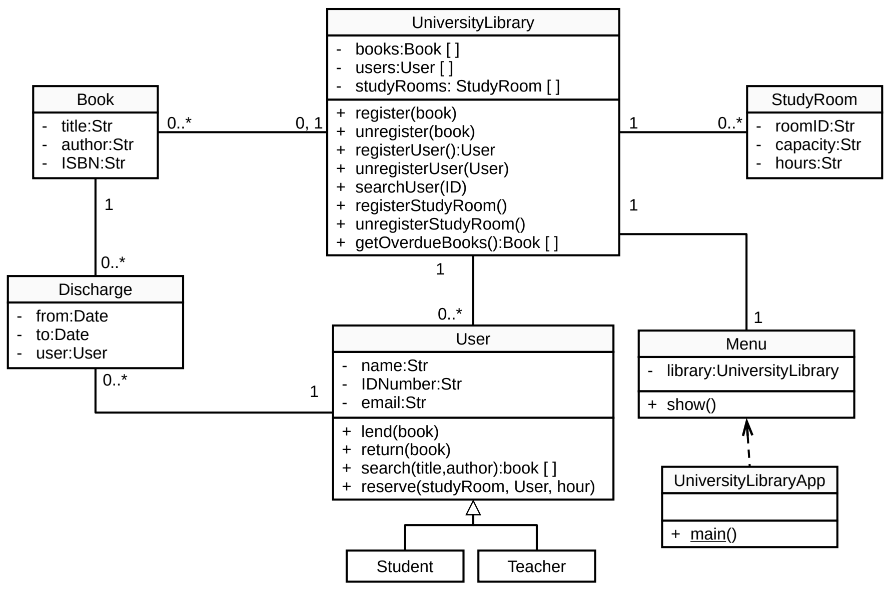

Figure 2 Corresponding UML Static Class Diagram.

3.4 The problem of unnecessary inheritance

Students tend to create unnecessary inheritance relationships. The Static Class Diagram can help to discard such relationships. When classes do not provide new methods or properties, they are often unnecessary. For example, *Student* and *Teacher* appear (in Fig. 2) as classes without attributes or methods. In reality, in the implementation these classes are not necessary, we only need from an attribute in the *User* class that indicates the type of user. Fig. 3 shows this solution. It could be thought that such classes would be necessary if the treatment of a *teacher user* was different from that of a *student user* (e.g., allowing for a larger number of books to be borrowed and longer borrowing times for the case of *teachers*). However, this can be solved in a more versatile way by creating a *UserType* class that would capture such differences between these two user types. In addition, this solution would allow the creation of new user types without modifying the implementation.

4 Discussion

The idea of teaching OOAD using concept maps (as it was proposed by previous authors [17]) is appropriate and powerful to easily recognize object abstractions, because these maps can help to discover objects and their relationships (e.g., inheritance) in an intuitive and visual form from the textual statement of a problem. This method encourages a deeper understanding of abstraction mechanism by students, since they must think more critically about the structure and behavior of their OO solutions to problems.

However, not using properly concept maps as a tool in OOAD courses can also have some undesired effects. First, students might focus too much on building a map that look neat rather than truly understanding the OO design principles. Concept maps can oversimplify complex relationships, leading to misunderstandings about how real-world systems behave. They can also be difficult to represent dynamic object behaviors (like polymorphism). Finally, assessing concept maps "objectively" can be challenging, since different students might organize the same ideas very differently.

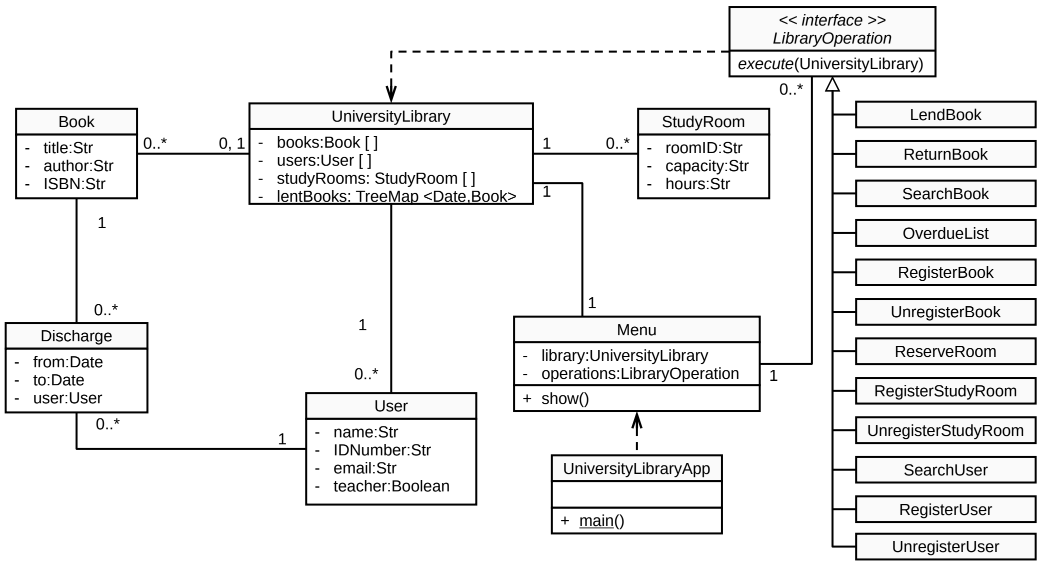

Figure 3 Corresponding UML Static Class Diagram, after corrections (omitting constructors, getters and setters).

Although the conversion from a concept map to a static UML class diagram is neither automatic nor immediate, the use of these maps as an intermediate model previous to the creation of a static class diagram, provides the following pedagogical advantages:

- It facilitates the understanding of the OOAD problem for instructors and students (or for the system analyst and the clients in a business approach). By being concept maps more readable and easy to understand, and having them only two types of components (i.e., nodes and connecting elements), students (and customers) can better understand the model derived from the problem statement.
- Concept maps avoid missing elements when translating the requirements to the implementation. These conceptual diagrams make it easy to check that all the elements present in the statement (or in the requirements) appear in the derived UML static class diagram.

5 Conclusion

We describe a work in progress about the application of concept maps to teaching an introductory OOAD course for Computer Science students. The proposal was illustrated using a sample class exercise. As mentioned, there are some previous works (in particular, the one by [17]) that propose a method to translate a concept map into a UML class diagram. However, in our opinion, these solutions are incomplete and require from the additional modifications such as the ones presented in Section 3. This way, we are able to produce more realistic solutions when using concept maps in introductory OOAD courses.

As future work, we propose to validate our approach with more complex class exercises. Moreover, we also will perform an evaluation study to measure the pedagogical effectiveness of adopting concept maps as a tool to help students when designing their UML static class diagrams.

References

1 Grady Booch. Object-oriented development. *IEEE Transactions on Software Engineering*, SE-12(2):211–221, 1986. `doi:10.1109/TSE.1986.6312937`.

2 Alberto J. Cañas and et al. Cmaptools: A knowledge modeling and sharing environment. In *Concept Maps: Theory, Methodology, Technology. Proceedings of the First International Conference on Concept Mapping*, 2004.

3 Buket Dogan and Emrah Dikbıyık. Opcomits: Developing an adaptive and intelligent web based educational system based on concept map model: An educational system based on concept map model. *Computer Applications in Engineering Education*, 24(5):676–691, May 2016. `doi:10.1002/cae.21740`.

4 Oscar Díaz and Xabier Garmendia. Bridging reading and mapping: The role of reading annotations in facilitating feedback while concept mapping. *Information Systems*, 127:102458, 2025. `doi:10.1016/j.is.2024.102458`.

5 Ernest Ferguson. Object-oriented concept mapping using uml class diagrams. *Journal of Computing Sciences in Colleges*, 18(4):344–354, 2003.

6 Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., USA, 3 edition, 2003. `doi:10.5555/861282`.

7 Jeroen Keppens and David Hay. Concept map assessment for teaching computer programming. *Computer Science Education*, 18(1):31–42, 2008. `doi:10.1080/08993400701864880`.

8 Robert C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, USA, 2003. `doi:10.5555/515230`.

9 Joseph D. Novak. *Learning, Creating, and Using Knowledge: Concept Maps as Facilitative Tools in Schools and Corporations*. Routledge, USA, 2 edition, 2010. `doi:10.4324/9780203862001`.

10 Joseph D. Novak and Alberto J. Cañas. The theory underlying concept maps and how to construct and use them. *Technical Report IHMC CmapTools 2006-01 Rev 01-2008, Florida Institute for Human and Machine Cognition, Pensacola.*, 2008.

11 Joseph D. Novak and D. Bob Gowin. *Learning How to Learn*. Cambridge University Press, Cambridge, 1984. `doi:10.1017/CBO9781139173469`.

12 Wagner A. Perin, Davidson Cury, Crediné S. de Menezes, and Camila Z. de Aguiar. Active learning with concept maps: Enhancing understanding of entity-relationship diagrams in database modeling. In *2023 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, 2023. `doi:10.1109/FIE58773.2023.10343471`.

13 Amparo B. Pia, Encarna Blasco-Tamarit, and Maria-Jose Muñoz-Portero. Different applications of concept maps in higher education. *Journal of Industrial Engineering and Management*, 4(1):81–102, 2011. `doi:10.3926/jiem.2011.v4n1`.

14 Kate Sanders, Jonas Boustedt, Anna Eckerdal, Robert McCartney, Jan E. Moström, Lynda Thomas, and Carol Zander. Student understanding of object-orientd programming as expressed in concept maps. In *39th ACM Technical Symposium on Computer Science Education (SIGCSE '08), Portland, Oregon, USA, March 12-15, 2008*, pages 332–336, 2008. `doi:10.1145/1352322.1352251`.

15 Vinicius dos Santos, Érica F. de Souza, Katia R. Felizardo, and Nandamudi L. Vijaykumar. Analyzing the use of concept maps in computer science: A systematic mapping study. *Informatics in Education*, 16(2):257–288, 2017. `doi:10.15388/infedu.2017.13`.

16 Shin-Shing Shin. Concept maps as instructional tools for improving learning of phase transitions in object-oriented analysis and design. *IEEE Transactions on Education*, 59(1):8–16, 2016. `doi:10.1109/TE.2015.2418176`.

17 Ven Yu Sien. Teaching object-oriented modelling using concept maps. *6th Educators' Symposium: Software Modeling in Education at MODELS 2010 (EduSymp '10), Electronic Communications of the EASST 34*, 2010. `doi:10.14279/tuj.eceasst.34.590.619`.

18 Ven Yu Sien and David Weng Kwai Chong. Threshold concepts in object-oriented modelling. *7th Educators' Symposium: Software Modeling in Education at MODELS 2011 (EduSymp '10), Electronic Communications of the EASST 52*, 2011. `doi:10.14279/tuj.eceasst.52.763`.