Using a Concept Map to Represent the Composition of Knowledge in an Introductory Programming Course

Pratibha Menon
menon@calu.edu

Lisa Kovalchick
kovalchick@calu.edu

Department of Mathematics,
Computers and Information Systems
California University of Pennsylvania
California, PA 15419

Abstract

Concept mapping, a tool originally developed to facilitate student learning by organizing and visualizing key concepts and their relationships, can also be used to represent the composition of the knowledge contained in a course. In this paper, the authors describe a specific application of concept mapping to help instructors and students visualize the knowledge contained in an introductory programming course of an undergraduate Computer Information Systems program. The authors show how representing the knowledge structure of a course using a concept map can enable the faculty to assess the breadth and depth of the knowledge imparted through the course. The authors discuss how a concept map that depicts the composition of a course can function as a useful instructional tool to assess and improve the quality of instruction that may enable meaningful learning among students.
Keywords: Concept-Map, Knowledge-Representation, Knowledge-Model, Programming, Course, Nodes, Links

1. INTRODUCTION

Any person who wishes to reason about his/her world comes across an inescapable fact that reasoning is a process that goes on in the mind of the person, while the very thing she/he wishes to reason exists outside the mind. This unavoidable dichotomy is the fundamental reason as to why we need some form of representation, or model, of the world about which we need to reason. This representation exists as a substitute for the real thing about which we wish to reason. Any operation that we wish to perform on the real thing can be performed on the representation and reasoning itself will be the surrogate for the action that we want to perform on the real thing.

In this paper, our focus is on creating a representation model for the knowledge contained in an introductory programming course and to use this representation to reason and draw inferences about various structural attributes of the knowledge contained in a course.

In order to be able to use a knowledge model, one needs to clearly know the intended purpose of this model, and the attributes of the real world that this model incorporates. In this paper, the proposed knowledge representation model intends to model the knowledge contained in the course as a network of concepts that the learner should master to result in a meaningful learning experience.

By selecting a knowledge representation model - which models a course as a network of concepts, we make an 'ontological commitment' that brings to focus a certain aspect of the course and blurs several other facets of the course. For example, the knowledge model does not model all the course contents explicitly, nor does it model the time required to learn concepts, all of which are important facets in the design of a course and will depend on the specific teaching context. The proposed knowledge representation model purely focuses on the skeletal structure of the knowledge base of the course.

Different forms of knowledge representations, known as 'ontologies', have been extensively used to model domain knowledge for teaching and learning purposes, primarily in the area of Personalized Learning Systems (Brusilovsky et al., 2004(a); Brusilovsky et al., 2004(b); Lee & Segev, 2012). According to one of the definitions "an ontology is a hierarchically structured set of terms for describing a domain that can be used as a skeletal foundation for a knowledge base" (Swartout et al., 1997). Unlike a taxonomy that classifies and organizes knowledge components, an ontology specifies the knowledge components and their relationships in greater detail.

Ontologies have been used to provide common vocabulary for query retrieval in a case-based recommendation strategy for personalized access to learning objects (LOs) in a learning system (Gomez & Diaz, 2009; Brusilovsky et al., 2004(a); Brusilovsky et al., 2004(b); Lee & Segev, 2012; Wang & Mendori, 2012). In such learning systems, we observe a knowledge model that depicts a tree-like organization of the concepts and/or learning objects. For example, Appendix A, shows a hierarchical tree-like structure used to create an ontology for the C Programming language (Sosonovsky & Gavrilova, 2006). A five-step algorithm has been proposed by Gavrilova et. al, to develop teaching Ontologies (Gavrilova et al., 2005). This algorithm has also been used to develop teaching Ontologies for the Java programming language (Ganapati et al., 2011). In all these examples, the Ontology categorizes the units of knowledge in the form of a tree-like hierarchy. However, our goal of creating the knowledge model is to organize and inter-relate the concepts, in way that will make it possible for students to learn how to write computer programs. Such a knowledge model may contain cross-links between the concepts and may result in a non-hierarchical structure.
In the proposed knowledge model, we depict such cross-links, and in this way, create a generalized network-like structure of organizing concepts. To organize and represent the key concepts of a course in a network-like manner, we propose to use a concept map based approach.

The design methodology of concept maps was first introduced by Joseph Novak by re-examining Ausubel's learning theory that differentiates rote learning from meaningful learning (Ausubel, 1963; Ausubel, 1968). The fundamental idea in Ausubel's cognitive psychology is that meaningful learning occurs by the assimilation of new concepts and propositions into the existing conceptual frameworks held by the learner (Ausubel, et.al, 1978). Novak argues that knowledge construction is nothing other than a relatively high level of meaningful learning, and that concepts and propositions are the building blocks for knowledge in any domain (Novak & Gowin, 1984; Novak, 2002). Novak compares concepts to atoms, and propositions to molecules. Just as molecules are formed by atoms and the valid relationships that bond them, propositions are formed by valid relationships among concepts. For example, consider the two concepts car and engine. The proposition can be an assertive statement such as, car has engine (see Figure 1). Here the linking word 'has' relates the two concepts, car and engine (Novak & Canas, 2008).

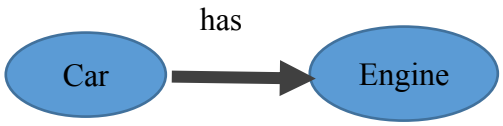

Figure 1. Example of depicting a proposition using nodes and link

Concept maps have been used as a scaffold for cognitive processing of knowledge in a given subject area (O'Donnell, et.al, 2002). Concept maps have found a wide variety of applications in the development of curriculum and instruction (Allen et al., 1993; Edmondson, 1993). Concept maps may be linked to high level learning objectives to create a 'finer-grained' learner model in an introductory programming course (Kumar, 2006). Concept maps have been used to represent the domain model, and also as an overlay student model in the design of tutoring systems (Mabbott & Bull, 2004). Concept mapping exercises and their scoring methods have been developed for testing students'

knowledge in an introductory programming course (Keppens & Hays, 2008).

The scope of this paper does not address the use of a concept map as a way to measure the learner's ability to create a computer program. Neither does the paper address the ability of students to construct their own concept maps. In this paper, a concept map is an instructional tool that will help instructors to represent the knowledge required to write procedural style computer programs in an introductory programming course for an undergraduate Computer Information Systems (CIS) program.

The figure shown in Appendix B, depicts a complete concept map developed by the authors. We explain the key design features of this concept map that can help the course instructors and students to understand the inter-relatedness of key concepts covered by course. We introduce some of the course quality attributes that can be inferred from the concept map. We also provide a sample implementation of a concept map based instructional strategy in a programming course. Finally, we provide a critique on the utility of the proposed concept map as an instructional tool to analyze the composition of the course contents to improve the quality of the course.

2. CONCEPT MAP OF AN INTRODUCTRY PROGRAMMING COURSE

The figure shown in Appendix B depicts the concept map representation created by the authors, for an introductory procedural style, Java programming course of an undergraduate CIS program. The concepts (i.e., nodes depicted in the figure) are extracted from the syllabus prescribed by the university approved course curriculum. The links that represent the relationship between the concepts were created by the authors to show the relationships between the key concepts.

The figure shown in Appendix C, depicts a partial view of the course syllabus. It can be inferred from the syllabus that there is a great emphasis on using programming concepts to solve problems. Additionally, the syllabus focuses on using case studies to outline the pseudocode problems, for which students are expected to create a programming solution. To represent the relationship that exists between problem solving and program structure, the concept map is divided into two clusters − the concepts that are part of the Problem Structure, and the concepts that describe the Programming Structure. This high level division follows the common teaching practice of many programs, where students are first made to analyze the problem, write pseudocode, and only then, write the computer program.

2.1 Mapping the Course Composition

As previously mentioned, the intent of the knowledge representation scheme is to map the composition of the course's subject area as a network of concepts and their interrelatedness. The key concepts to be covered by the course are identified and represented as distinct nodes in the concept map. The next step is to connect these concepts using proper linking phrases that can meaningfully convey the relationship between concepts that the instructor must convey through course contents and the student must learn, to successfully meet the course objectives.

In the concept map shown in the figure in Appendix B , the links between the concepts are primarily labeled using the phrases – 'is a type of ', and 'has'. It is typical for course domain ontologies to focus on 'is-a-type' and 'has'/'part-of' relationships between concepts (Omez-Albarran & Jimenez-Diaz, 2009; Sosnovsky & Gavrilova, 2006) . However, it is to be noted that concepts maps have no restriction on what kind of meaningful phrases that can be used to label the links.

The relationships indicated in the concept map are read along the direction of the arrows. For example, the relationship between Program and Method(s) is read as, "A Program has (one or more) Method(s)". The relationships can be of the 'one to one' type, or of the 'one to many' type. In a 'one to many' relationship, the concept on the many side is written in plural tense such as Method(s), or Expression(s). The non-arrow side of the relationship is always singular, and the arrow side of the relationship can be singular or plural. If the arrow side of the relationship is a plural, then the concept at the arrow side will be indicated in plural tense.

The selection of meaningful phrases used to label the relationships depends on the intent of the concept map. Since the primary goal of the concept map is to represent the course composition, the words 'has' and 'is a type of' convey the composition, and choices of composition, respectively.

For example, the concept map fragment in Figure 2, shows how programs are composed of some of the concepts. In this concept map

fragment, a "program *has* (one or more) method(s). A method is *of type* main method. A method *has* (one or more) statement(s), and a statement *has/is of type* expression(s)." This chain of reasoning indicates a program is *composed* of one or more methods, and a method is, in-turn, *composed* of one or more statement(s), and a statement is *composed* of one or more expression(s). The 'has' relationship indicates that one concept is composed of another concept. There are two types of relationships between statement(s) and expression(s). A statement is always composed of an expression and some statements only have expressions (i.e., no loops, or branches, for example).

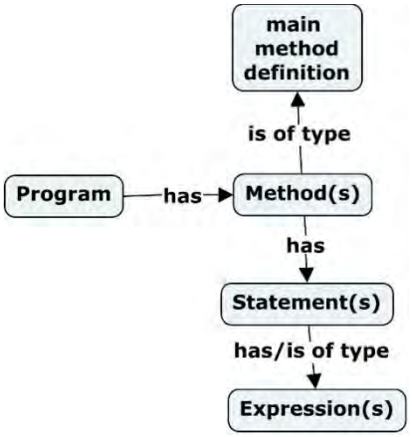

Figure 2. A small subset of the course concept map showing program composition

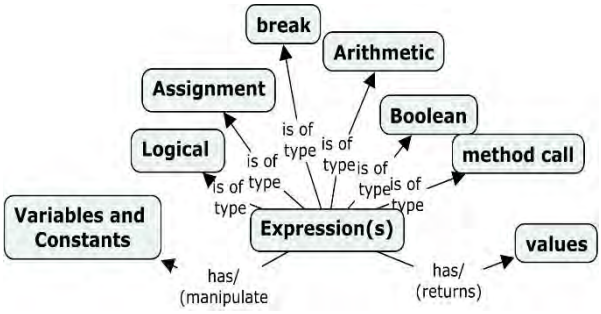

Figure 3. A Small subset of the course **concept map showing 'is a type of'** inheritance relationship

The 'is of type' linking phrase indicates 'choices of composition' that the learner must be able to discern for a given problem. This kind of a relationship indicates the need for the learner to discern among several permissible options of program or problem composition. For example, the concept map fragment in Figure 3, shows that every Statement *has* Expression(s), and an Expression can be *of many types* (such as Assignment, Arithmetic, Logical, Method call, Boolean, and Break). From this one can infer that to compose a statement, one will need to discern among, and choose from, a set of permissible expressions.

2.2 Mapping concept hierarchy

Even though a concept map has an overall network-like structure, sections of the concept map can define several types of tree-like, hierarchical relationships between concepts. Appropriate linking phrases can be used to depict these hierarchical relationships. One such hierarchical relationship, that we observe in the concept map depicted in Figure 1, is that of the concept of inheritance between what can be called the 'child' and 'parent'. The linking phrase, 'is a type of' relates a 'child' concept, to a 'parent' concept, as shown in Figure 4. For example, the concepts named Selection and Iteration are the child concepts of the parent concept called Statement(s). A single parent may have one or more child concepts and a child concept may, in turn, be a parent to its own child concept(s). For example, in addition to being a child concept of Statement(s), the Selection concept is a parent of the concepts named if..else and Switch.

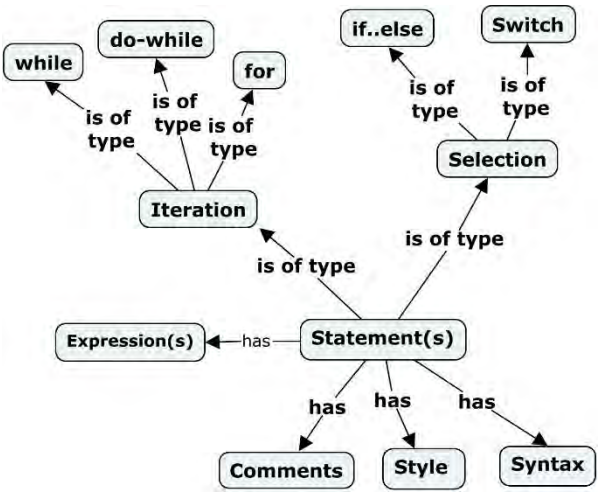

Figure 4: A small fragment of the course concept map showing concept hierarchy and inheritance

The 'is a type of' relationship indicates 'inheritance' of pre-requisite topics from the parent node. For example, in Figure 4, the concept called Statement(s) is composed of several concepts such as Expression(s), Style, Syntax, and Comments. All these pre-requisite concepts are inherited by the child nodes –

Selection and Iteration. This implies both Selection and Iteration lessons require the lesson on Expressions to be covered a-priori.

It can be inferred that the parent concepts are relatively more abstract, as compared to a child concept. This means the child concept provides more concrete course contents and learning implementation of the parent concept. For example, in Figure 3, the Parent concept called Expression(s) are learnt in concrete forms as Assignment expressions, Arithmetic expressions, etc.

2.3 Inferring the Pre-Requisite Concepts

Each node in the concept map can be considered as one lesson/topic of the course. For example, a typical programming course has a topic called Selection that will introduce the ways to write 'if..else' decision structures in the program. Another example of a commonly taught topic is that of writing various types of Expressions, such as an expression for variable assignment, an expression for arithmetic computation, etc.

Even though each lesson can be depicted as a separate node in the concept map, these concepts may require that the learner has mastered one or more pre-requisite concepts. For example, let us take the two concepts – Selection and Expressions, as shown in Figure 4. The two concepts are related to each other indirectly, through the Statement(s) node. The Statement(s) node is related to the Expression(s) node through a 'has' relationship. For example, to write an 'if..else' statement, one needs to know how to write a Boolean statement, that returns a value of true. This return value of the Boolean expression will then be used to trigger the code under the 'if' block. An 'if..else' statement block also may contain various arithmetic operations, or method calls. Thus, the inference that an 'if ..else' statement is composed of expressions, implies that the lesson on expressions should precede the lesson on Selection.

From the concept map shown in Appendix B, one can infer the order, in which lessons need to be taught, starting with the concepts that require fewer pre-requisites. For example, the lessons are typically taught in the following order: Data Units—Variables and Constants (along with primitive data types), Declaring variables, Expressions, Writing statements that have declaration and expressions, Selection structures, Iteration structures, and Methods.

3. INFERRING THE QUALITY OF KNOWLEDGE IMPARTED BY THE COURSE

The focus of the concept map presented in this paper is to reason about the composition of the introductory programming course designed to teach programming to beginners. Therefore, we will discuss how the structure of this composition impacts the quality of the course.

3.1 The breadth of the course

The breadth, or the scope of the course can be loosely defined as the number of topics covered in a course, which is typically defined in the course syllabus. The breadth of the course, as depicted by a concept map, can be assessed by the total number of concept nodes present in the map. For the example, the concept map shown in Appendix B, has 43concepts.

As discussed in the previous section, having a web-like knowledge structure, does not prevent creation of hierarchy, or levels of abstraction. Some of the concepts, for example, the Arithmetic expression, can be further split into an expression with operators such as, addition, subtraction, division, multiplication, and modulus. Similarly, the concept called Primitive Data Types can be split into integer, double, character, String, Boolean, etc. In this case, the breadth of the concept map will account for all the child nodes that form the body of knowledge.

The breadth of the course only counts the number of concepts covered in the course, but it does not convey how the course imparts knowledge about the interconnectedness of these concepts. The breadth does not convey how densely, or sparsely interconnected these concepts are, in the body of knowledge taught in the course.

3.2 Depth of the course

Compared to the breadth of a course, denseness, or depth, has always been perceived as a qualitative measure that may depend on the conceptual details taught and assessed for each topic of the course. For example, a solution to a 'dense' problem may require students to meticulously reason and connect together concepts acquired from several previously taught topics in an accurate manner. This pre-requisite knowledge may include concepts that may be an integral part of the newly introduced concept. The pre-requisites may also include those concepts that that are alternatives to the new concept and that the students may need to discern correctly. A 'dense' problem packs and

interconnects several concepts into the solution. The ability of the learners to form a coherent chain of reasoning that ties several concepts and that allows the learner and pick and choose the correct concepts leads to deeper learning. A concept map explicitly depicts the links between various concepts. Therefore, a solution to a dense problem may require students to reason along a 'chain of links' that connect various concepts in the concept map.

Deeper learning occurs when students are able to associate one concept to several other concepts listed in the course. For example, a simple Hello World program that is usually the first program that a student may write, actually binds 14 different concepts using the links depicted in the concept map. Figure 5, depicts those concepts. While it is possible and even sometimes desirable for novice students to be able to write an initial Hello World program without reasoning through so many concepts, this kind of learning will only promote superficial and rote learning in the long run. Therefore, to promote deeper learning, and mastery of concepts, the learner should be provided with instruction and practice activities that will teach how to reason well by identifying the correct concepts and their interconnectedness that can explain the solutions to the problems.

Learning Path : 15 Concepts
Concept: Business Problem
Concept: Input/Output: **Outputs Hello World**
Concept: Functional requirements : To output ***Hello World***
Concept: Sequence of operations /pseudocode: ***To output/print two words***
Concept: Method Call: ***call the System Output method to print Hello World***
Concept: User/System Input/Output: ***System Output***
Concept: Program: ***Created as a Class HelloWorld***
Concept: Executable Class: ***HelloWorld.java***
Concept: main method: ***has statements***
Concept: Method(s) : ***only main method***
Concept: Statement(s): ***has an expression***
Concept: Expressions: ***has a method call***
Concept: method call: ***System.out.println("Hello World")***
Concept: Data : ***"Hello World" used in method call***
Concept: Literals/Value: ***"Hello World" is a String literal***

Figure 5: A chain of concepts involved in a **simple "Hello World" program**

Figure 6, shows the concepts and the chain of reasoning involved in a simple program involving arithmetic expressions. This program chains 26 concepts that form a small subset of the concept map.

It is theoretically possible to create a problem whose solution will involve the entire concept map and requires students to reason through all the concepts and links. Solutions to smaller problems may constitute a smaller subset of the concept map, comprised of fewer nodes and links.

Learning path : 24 concepts
Concept: Business Problem
Concept: Functional requirements : **Obtain two inputs, add them, output the res**
Concept: Input/Output: **Input two integers, output the result**
Concept: User/System Input/Output: ***both system, input and output***
Concept: Sequence of operations /pseudocode: ***Method call, add, assign, meth***
Concept: Arithmetic :***addition- perfom input1 + input2***
Concept: Assignment: ***assign output = input1+input2***
Concept: Method Call: ***To get input1 and input2, To print output***
Concept: Program: c***reates a class AddIntegers***
Concept: Executable Class: c***reate a file AddIntegers.java***
Concept: main method: ***in the program, has statements***
Concept: Method(s): ***only main method***
Concept: Statement(s): ***has expressions***
Concept: Expressions: ***method call , Arithmetic, Assignment, Declaration***
Concept: Arithmetic: ***addition operator to add input1+ input2***
Concept: Assignment:***to assign values of input1, input2 and output***
Concept: method call: ***nextInt(), System.out.println()***
Concept: Declaration: ***declare variables input1, input2 and output as int***
Concept: Syntax
Concept: Style
Concept: Comments
Concept: Variables and Constants: ***3 ints***
Concept: Primtive Data types: ***int***
Concept: Data units: ***values of input1,input2, output***

Figure 6: A chain of concepts involved in a simple problem that performs addition on two inputs and outputs the result

4.0 IMPLEMENTING A CONCEPT MAP BASED COURSE DESIGN

The concept map depicted in Appendix B was used to assess and re-design a 15-week course on Java Programming. Initial assessment of the course indicated the need to instruct in greater detail the teaching of reasoning that goes into composing a program. The course contents were re-designed after adopting the concept map as the course content schema. In this section we reflect upon the implementation experience.

4.1 Assessment of Course Quality

The 15-week course on Java Programming includes all the topics mentioned in the course syllabus.

1) Intro to Java, writing a simple program. Basic Input/Output. Identifying errors, writing good comments.
2) Intro to variables, identifiers, assignment operator, and arithmetic operations, using int and double variables

3) Using char and String data types, int-double type conversions, math methods, and random numbers
4) Variables
5) Branches. Use of if-else branches. Relational and equality operators, logical operators.
6) Switch statements, Boolean data types.
7) String comparison, access and modify operations, char and conditional expressions.
8) While loop
9) For loop, break
10) Intro to methods, parameters, and return statements.
11) Methods with branches, loops, method name overloading.

All the above-mentioned topics involved exercises that identify the programming constructs required to solve a problem.

The instructors have commonly observed that students tend to learn the code by rote, without being able to 'see' the common patterns or inter-relatedness between concepts that occur in the structure of the program. For example, while writing a loop, many students fail to understand that the looping condition is an expression that returns a Boolean value. While Boolean expressions were previously taught, students may not fully integrate the past lesson into the new concept. As a result, many try to learn the worked out program by rote, and thereby failing to apply what was learnt to new problem. Some of the topics that were perceived to be most difficult were the topics covered later in the course such as: branches/selection, loops/ iteration, and methods. All of these topics require mastery in applying a large number of pre-requisite concepts.

One of the main motivations for re-designing the course was to re-create contents that focus on the conceptual knowledge and reasoning required to compose a programming solution to a given problem. The re-design would create instructions to explicitly show the relationships between various concepts required to create programs.

4.2 Using the concept map to assess and re-design the quality of instruction

An initial assessment of the course syllabus showed that the course had the required breadth and covered all the required concepts mentioned in the course syllabus. Additional concepts were added to emphasize the role of pseudocode and various types of sequence of operations that students had to infer from the business problem before writing the program.

The course lectures and assignments were investigated topic-by-topic to determine whether they have contents that explicitly conveys the relationships between various concepts covered in each topic. Additional lecture slides and code demonstrations were created to depict the relationships between various concepts involved in the topic. The "has" and "is a type of" phrases were used to depict the relationships between concepts.

To promote meaningful learning among students the programming demos expressed a program in-terms of its conceptual composition. All the concepts involved in the problem statements and the programming solutions were explicitly explained during the demonstration. The initial re-design of instruction had not considerably changed the course sequence and the programming activities. The only change was in the instructional narrative that incorporated the concept map and the program composition methods that included the chaining of concepts, as explained in Section 3.2. The newly re-designed lectures used the concept map to bring greater clarity to the lectures and programming demonstrations. The programming activities required students to reason through their program using the chain of concepts, as shown in Figures 5 and 6. Students were made to complete worksheets such as the ones shown in Figures 5 and 6.

4.3 Evaluating the impact of instruction re-design

In order to evaluate the impact of re-designed instruction, the learning outcomes of the re-designed course were compared with the outcomes from the previous semester. A scoring matrix, as shown in Appendix D, was used in both semesters. For comparison purposes, scores from six similar assignments were used to compare the outcomes of the re-designed instruction.

Each student was scored on a scale from 0 to 5, with 0 being the lowest score, and 5 being the maximum score. The average scores, before and after the course design are depicted for each component of the score matrix shown in Appendix D. The x−axis of each of the charts below shows the assignment number, with assignment 1 requiring knowledge of fewer concepts, and assignment 6, requiring knowledge of more concepts. Figures 7, 8, and 9

show the score matrix items that showed an improvement in learning gains, after the instruction was re-designed. Scores were obtained before and after the instructional redesign, for 28, and 26 students, respectively. The average class scores were rounded up to the closest integer value.

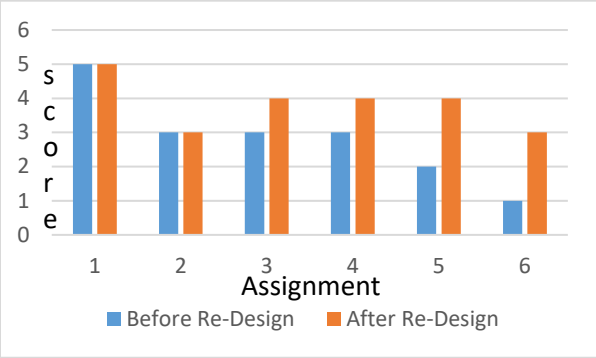

Figure 7: Average scores of students in their ability to write the order of statements correctly, as required to meet the requirements of the problem.

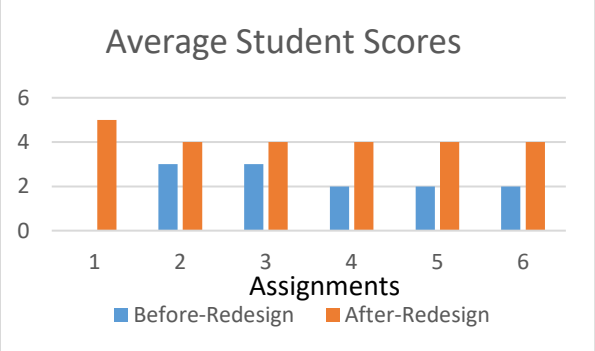

Figure 8: Average scores of students in their ability to identify the correct type of statements required to solve the problem

It can be observed from Figures 7 and 8 that the greatest learning gains, due to the re-design of instruction, were in the later assignments that were more 'dense' and packed several concepts into a reasoning chain. There were gains in the student's ability to identify and write statements in proper order. Gains were also observed in the ability of students to identify the proper type of expressions required to complete statements. However, students' ability to write correct expressions to complete the statements did not show marked improvement, as depicted in Figures 9 and 10. In most cases, the statements were incorrect because there were mistakes in the way the expressions were written. The majority of the mistakes were made in complex expressions that involved comparison and logical operators. Mistakes were also made in method calls.

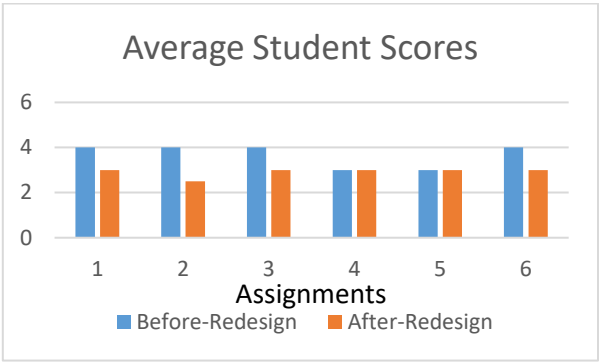

Figure 9: Average scores of students in their ability to write all the expressions correctly

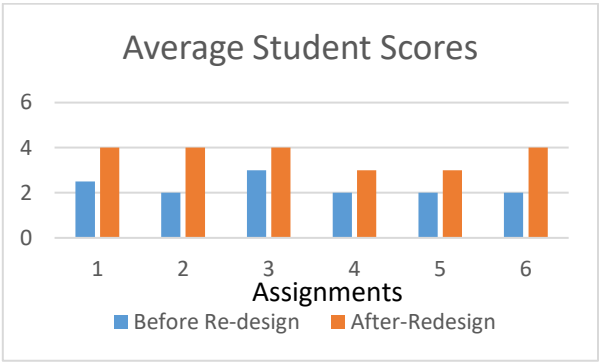

Figure 10: Average scores of students in their ability to identify proper expressions for statements.

Overall, the use of course knowledge representation using a concept map guided the instructor to create instructional material that combines conceptual knowledge with the practice of writing programs. The concept map provides students and the instructor a common vocabulary and representation to discuss the concepts. By chaining various inter-related concepts, the instructor was able to create instructional materials that makes explicit the systematic chains of reasoning required to compose programs. The effect of introducing a

new concept map based instruction was measured through assessment of weekly assignments provided to students. The results of the assessments provided the instructor with useful information about the concepts and chains of reasoning, for which improved instruction might be required.

5. CONCLUSIONS

This paper presents the possibility of using a concept map to represent the knowledge composition of a course that requires students to learn to write procedural style programs in Java. The intent of using a concept map was to map the knowledge composition of the course in the form of concepts and their interconnectedness, such that the depiction can be used to create instructional methods that can help students to learn how to compose programs. The structure of the concept map can inform the course designers about the breadth and depth of the knowledge imparted through the course. Using examples, the authors show how the interrelated-ness of concepts can be used to create chains of reasoning to explain programming solutions to different types of programming problems. Consequently, the concept map also helps students to learn meaningfully by enabling them to interconnect various concepts to produce programming solutions.

6. REFERENCES

Ausubel, D. P. (1963). The psychology of meaningful verbal learning. New York: Grune and Stratton.Ashby, W. Ross (1956).

Ausubel, D. P. (1968). Educational psychology: A cognitive view. New York: Holt.

Ausubel, D. P., Novak, J. D., & Hanesian, H. (1978). Educational psychology: A cognitive view (2nd ed.). New York: Holt, Rinehart and Winston.

Allen, B. S., Hoffman, R. P., Kompella, J., & Sticht, T. G. (1993). Computer-based mapping for curriculum development. In: Proceedings of selected Research and Development Presentations Technology sponsored by the Research and Theory Division. New Orleans, LA. (Eric Document Reproduction Services No. ED 362 145).

Brusilovsky P., Sosnovsky S., Shcherbinina O. QuizGuide: Increasing the Educational Value of Individualized Self-Assessment Quizzes with Adaptive Navigation Support. In Janice Nall and Robby Robson (eds.) Proceedings of E-Learn 2004. Washington, DC, USA: AACE, 2004, 1806-1813.

Brusilovsky P., Sosnovsky S., Yudelson M., An Adaptive E-Learning Service for Accessing Interactive Examples. In Janice Nall & Robby Robson (eds.) Proceedings of E-Learn 2004. Washington, DC, USA: AACE, 2004, 2556-2561.

Edmondson, K. M. (1993). Concept mapping for the development of medical curricula. Paper presented at the Annual Conference of the American Educational Research Association, Atlanta, GA. (Eric Document Reproduction Services No. ED 360 322).

Ganapathi, G., Lourdusamy, R., Rajaram, V. (2011). "Towards Ontology Development for Teaching Programming Language," presented at the World Congress on Engineering, London, UK.

Gavrilova, T., Farzan, R., Bursilovsky, P. (2005). "One Practical Algorithm of Creating Teaching Ontologies", Proceeding of the *NBE* 2005.

Gomez-Albarran, M., & Jimenez-Diaz, G. (2009). Recommendation and students' authoring in repositories of learning objects: a case-based reasoning approach. International Journal of Emerging Technologies in Learning (IJET), 4, 35e40

Keppens, J. & Hay, D. (2008). Concept Map Assessment for Teaching Computer Programming. *Computer Science Education, 18*(1), 31-42.

Kumar, A. (2006). Using Enhanced Concept Map for Student Modeling in Programming Tutors.. FLAIRS 2006 - Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference. 527-532.

Lee, J. H., & Segev, A. (2012). Knowledge maps for e-learning. Computers & Education, 59(2), 353e364.

Mabbott, A., Bull, S. (2004). Alternative Views on Knowledge: Presentation of Open Learner Models. In Lester, J.C., Vicari, R.M., Paraguacu, F. (eds.), Intelligent Tutoring Systems: 7th International Conference, Springer-Verlag, Berlin Heidelberg, 689-698.

Novak, J. D. & Gowin. D (1984). Learning How to Learn. Cambridge University Press.

Novak, J. D. (2002). Meaningful learning: the essential factor for conceptual change in

limited or appropriate propositional hierarchies (liphs) leading to empowerment of learners. Science Education, 86(4), 548e571.

Novak, J. D., & Canas, A. J. (2008). The theory underlying concept maps and how to construct them. Technical report IHMC CmapTools 2006-01 Rev 01-2008. Florida Institute for Human and Machine Cognition.

O'Donnell, A. M., Dansereau, D. F., & Hall, R. H. (2002). Knowledge maps as scaffolds for cognitive processing. Educational Psychology Review, 14(1), 71e86.

Sosnovsky, S., & Gavrilova, T. (2006). Development of educational ontology for C-programming. International Journal Information, Theories & Applications, 13(4), 303e307.

Swartout, B., Patil, R., Knight, K., Russ, T. Toward Distributed Use of Large-Scale Ontologies, Ontological Engineering. AAAI-97 Spring Symposium Series, 1997, 138-148.

Wang, J., & Mendori, T. (2012). A course-centered ontology of japanese grammar for a language learning support system. Frontiers in Artificial Intelligence and Applications (KES2012), 243, 654e663.

Appendix A

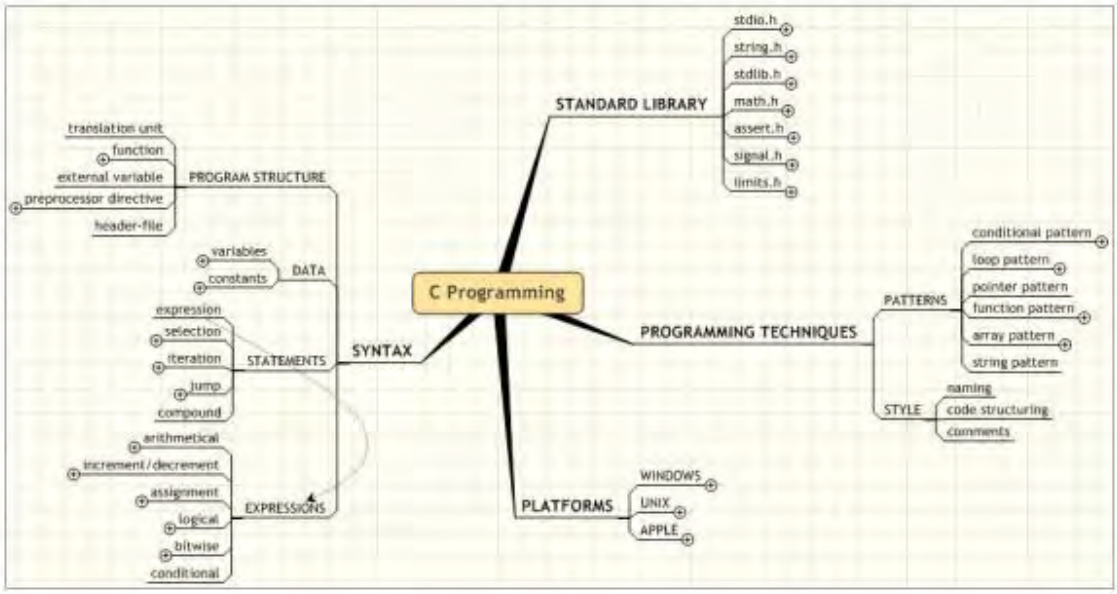

Figure A. Tree-like structure of an educational ontology for C-Programming (Sosonovsky & Gavrilova, 2006)

APPENDIX B

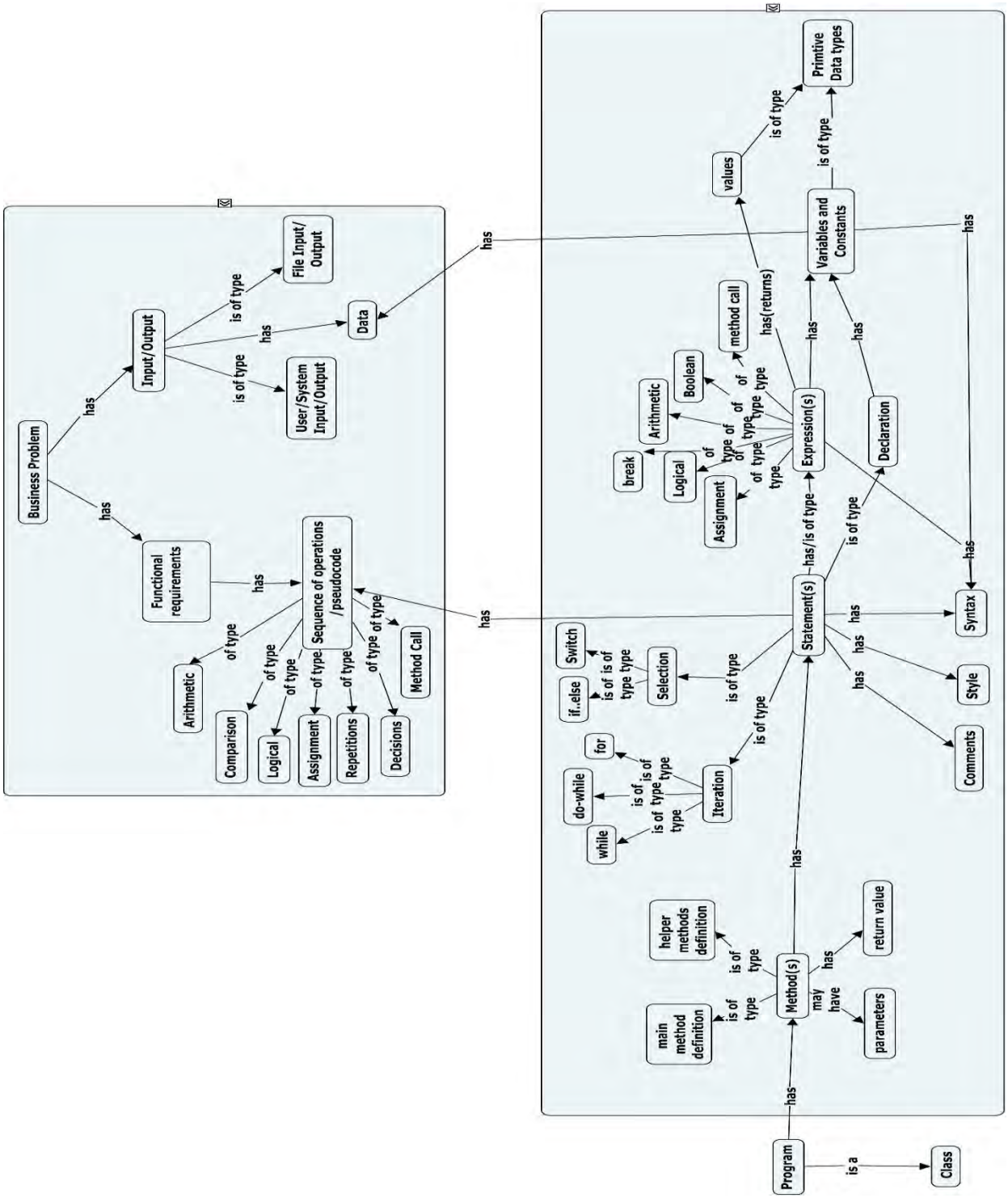

Figure B. Concept Map of an Introductory Java Programming Course

APPENDIX C

A. Objectives of the Course:

Upon completion of this course the student will be able to do the following items using the presently adopted language for this course (Fall 2010: Java):

a) Analyze business case studies and discuss strengths and weaknesses of various potential solutions.
b) Recognize and use problem solving techniques and methods of abstract logical thinking to develop and implement structured solutions of given software design problems.
c) Apply problem solving techniques and design solutions to business problems and implement these solutions by writing computer programs.
d) Write well-structured business programs.
e) Evaluate and debug programs.
f) Work in collaborative groups.

B. Catalog Description:

This course provides students with an understanding of business problems that are typically solved by writing computer programs, problem solving techniques to enable students to design solutions and programming skills learned in a traditional CS1 course. Emphasis is placed on efficient software development for business related problems. Students are required to write, test and run programs. Prerequisite: High School Algebra or Equivalent. Three credits.

C. Outline of the Course:

a) Problem Solving Techniques for Business Problems
   i) Business Case Studies
   ii) Problem Identification and Understanding
   iii) Solution Planning (flowcharts, pseudo-code, etc.)
   iv) Algorithm Development

b) Programming Concepts
   i) Structure of a Program ("Hello World")
   ii) Constants, variables and data types
   iii) Arithmetic operators
   iv) Relational operators
   v) Logical operators
   vi) Assignment statements
   vii) Input and output
   viii) Selection (if/else and switch)
   ix) Repetition (while, do/while, and for)

c) Strings

d) File Processing

e) Functions (in presently adopted language, "method'')

Figure C. A portion of the prescribed syllabus of the Introductory Programming course.

APPENDIX D

	Student is able to write the order of statements correctly, as required to meet the requirements of the problem	Student is able to identify the correct type of statements required to solve the problem	Student is able to identify the correct type of expressions to compose the statements	Student is able to write all the expressions correctly	Student is able to correctly identify the variables and its data types required to capture the data in the problem	Student is able to obtain the required inputs, as required by the problem	Student is able to correctly output data as per the problem requirements
Assignment 1:							
Assignment 2: Statements with expressions, input and output							
Assignment 3: Statements with variables,expressions, input and output							
Assignment 4: Statements with if.else / switch , variables, expressions, input and output							
Assignment 5: Statements with various types of loops,variables, expressions, inputs and outputs							
Assignment 6: Statements with if..else, loops, variables, expressions, inputs and outputs							

Figure D: A sample rubric used to evaluate the learning outcomes of each student in a **programming course. This rubric is used to measure a student's ability to write correct** programs that meet the requirements of a given business problem.