Programming Course Design: Phenomenographic Approach to Learning and Teaching

Neena Thota
Faculty of Creative Industries
University of Saint Joseph
Macau, S.A.R.
neenathota@usj.edu.mo

***Abstract*— Phenomenography is a well-known empirical research approach that is often used to investigate students' ways of learning programming. Phenomenographic pedagogy is an instructional approach to plan learning and teaching activities. This theoretical paper gives an overview of prior research in phenomenographic studies of programming and shows how the results from these research studies can be applied to course design. Pedagogic principles grounded in the phenomenographic perspective on teaching and learning are then presented that consider how to tie students' experiences to the course goals (relevance structure) and how to apply variation theory to focus on the desired critical aspects of learning. Building on this, an introductory object-oriented programming course is described as an example of research-based course design. The insights gained from the experience of running the course are shared with the community of computer science educators, as also the benefits and responsibilities for those who wish to adopt the phenomenographic perspective on learning to plan their teaching. The development of an increased awareness of the variation in students' ways of experiencing programming and the need to broaden the context of the programming course are discussed.**

Keywords—phenomenography; variation theory; introductory programming; object-oriented programming;

I. INTRODUCTION

Within the computing education research community, there is an increasing interest in phenomenography as an empirical, qualitative research approach. A number of phenomenographic studies have been conducted on students' understanding of computer systems and programming [1-6]. These studies establish that students hold a range of conceptions and motives for learning computing. Studies on specific concepts in Object Oriented Programming (OOP) [7-9] have revealed qualitative differences in conceptions of OOP.

Phenomenographic pedagogy, or *fenomenografisk didaktik* [10] in Swedish, emphasizes a learner-centered holistic approach to teaching, that integrates content and approach to facilitate and evaluate learning outcomes. The two principles of teaching that are espoused in phenomenographic pedagogy are (a) the relevance structure - making the learner personally experience the relevance of the learning situation, and (b) variation theory - emphasizing the critical aspects that lead to a change in the learner's understanding of a learning concept [11]. However, neither the findings from research on ways of learning programming nor the principles of phenomenographic pedagogy have been utilized to the fullest extent to design programming courses.

Phenomenographic pedagogy emphasizes the central role of the learner in a learning process that brings about conceptual change. Constructivism also provides a framework to think about improving teaching by focusing on those factors that encourage deep approaches to learning that lead to attainment of desired learning outcomes. However, constructivism is less clear about the specific outcomes that are the objective of deep approaches to learning. The phenomenographic perspective [11] offers a way of describing desirable learning outcomes, of dealing with the specifics of what is taught about the subject content, and of enabling changes in students' conceptions through awareness of their misconceptions. Variation theory, in particular, is seen as providing students with powerful ways of seeing or experiencing phenomena that are fundamental for deep learning.

There are many supportive arguments to use the findings of phenomenographic research to design programming courses. If learning is seen as changes in experiences, then the existing perspectives of students of programming should be acknowledged as integral components of teaching practice [5]. Knowledge of the variations in student understandings of programming can be used to promote change in learner conceptions, an approach that aims at the scholarship of teaching [12]. Another argument is that the studies contribute to student-centered teaching when teachers acknowledge their students' learning in the design of teaching activities and assessments [13]. Finally, phenomenographic studies that present the students' perspectives raise teachers' awareness of how variation in their thinking and practice might influence the learning approaches of their students [14].

A review of the literature reveals that although numerous phenomenographic studies of student learning in programming exist, the adoption of the phenomenographic perspective, or the actual implementation of phenomenographic based pedagogical design in introductory OOP courses is not prevalent. In two studies, variation theory is employed at the micro level i.e. a specific object of learning is identified and the lesson is built on

bringing the critical aspects of variations to the focal awareness of the students. In the first, a pilot study [15] with university–level engineering students, the application of variation theory in a computer lab assignment is described. Encouraging results are reported in the study in the form of increased student understanding of the relationship of the program text to the program action. In the second study, the benefits of using variation theory have also been emphasized for teaching OOP concepts related to flow of control, based on interacting entities and objects as behavioural abstractions of real world entities [16].

The main objective of this theoretical paper is to provide a sound basis for using phenomenographic research outcomes and pedagogy as the foundation for programming course design and teaching. In Sections II and III, key findings and recommendations from phenomenographic studies on programming are presented. Pedagogical theory grounded in the phenomenographic perspective of learning is described in Section IV. An example of an introductory OOP course inspired by phenomenographic pedagogy is described in Section V. The paper concludes with a discussion of the outcomes of applying the phenomenographic principles to course design (Section VI) and a summary of the guiding principles (Section VII).

II. PHENOMENOGRAPHIC STUDIES OF PROGRAMMING

A. Learning to Program

Phenomenographic research has revealed that students understand programming in a limited number of ways. Analyses of how students learn to program have revealed qualitatively distinct categories of descriptions of what it means to learn to program, orientations or ways of seeing programming, and ways of experiencing programming languages [4, 17]. These categories showed distinct levels of surface and deep understanding.

At the surface level, the studies found that a learner conceived learning to program as learning a programming language, and focused on the features and details of the programming language. At the next level of understanding, the learner conceived learning to program as learning to write programs in a programming language, and focused on the available techniques and features of the programming language. With increasing deeper understanding, students developed a problem-oriented or product-oriented approach. These orientations depended on whether the students approached learning to program as problem solving that required analysis before writing the program, or they envisaged it as a means of becoming part of a collaborative programming community.

These studies [4, 17] also show that the variation in the ways of experiencing learning to program and the capabilities to experience them, are crucial for the quality of learning. The categories of variations are deemed increasingly inclusive, with the possibility that not all students might develop a competence for experiencing programming in all of these ways. The implication for teachers is they should identify and bring educationally critical aspects of programming concepts to the attention of students, in ways that allow students to understand them.

B. Learning OOP

Phenomenographic studies that focus on approaches to learning OOP have also uncovered variations that point to distinct surface/deep approaches to learning. The ways of experiencing the act of learning to program [5] was found to vary in terms of learning approaches, perceptions of learning a programming language, motivations in learning to program, and ways of seeing programs and programming.

Surface orientations to learning were categorized [5] as following, and coding. In the first category, the students were more interested in completing the unit. They focused on the assigned task or relied on feedback, and were driven by the motivation to pass the course. In the next category, the students saw learning to program as learning the mechanics of writing code. These students focused on the programming language, were motivated to learn syntax and vocabulary, and considered programming as writing code using a specific syntax.

An investigation of the meaning of programming thinking in object-oriented courses [6] found three levels that related to surface understanding of programming. These categories expressed inclusive and hierarchical understanding directed towards the computer, the programming language, and programming in general. At the first level, learning was viewed as the use of the Java programming language with an emphasis on learning syntax. At the second level, learning to program was experienced as learning a way of thinking related to the programming language. However, the students could not offer a coherent explanation of what this thinking typified. The third level referred to learning as gaining an understanding of computer programs used in daily life. This study was the first to identify the latter two levels.

Another inquiry into perceptions of OOP [9] found three categories that focused only on the programming language and its constructs. Programming was experienced initially as learning the syntax to write code that compiled correctly. The next category dealt with learning the syntax with an understanding of the constructs of the programming language. In the third category, which was inclusive of the above two, the focus was on writing programs with knowledge of the syntax and some understanding of programming constructs.

Studies further reveal that students can progressively experience programming as learning through understanding and integrating concepts, as learning to solve a problem, and as learning what it means to become a programmer [5, 18]. When learning to program was seen as involving understanding of concepts, it led to integration of concepts and perspectives from experience and experimentation based on variation. When programming was seen as applying and integrating concepts, it was with a view to seeking insights into the logic of the language. When programming was seen as problem solving, it meant a

deep learning orientation with an effort to understand the overall problem, and to come up with solutions to the problem. The conception of a programming language as a means to solve the problem was evident in these students. Finally, learning was seen as a means to become a programmer, motivated by a desire to be part of the programming community. At this level, the cultural and communicative aspects of programming were recognized, and a shift in thinking was visible. The conclusion is that even though successful students may adopt any of the different learning approaches associated with different stages during their study, students should also be encouraged by teachers to experience the range of different ways of learning to program.

Two categories that signified deep approaches with an outward orientation towards societal and real world contexts have also been identified in [6]. The distinct conception of programming as learning a way of thinking, which enables problem solving, was evident. The authors urge educators to encourage students to move towards programming thinking to help with the abstraction in analysis and design that OOP requires. In the last category that was identified in this study, programming was perceived as meaningful when applied outside the course context. The benefits of learning to program in advanced studies, future work life, and as an expression of independence was recognized by the students.

A similar, inclusive advancement from basic understanding to learning a unique way of thinking was found in [9]. The link to logical thinking, and the moment when programming started to make sense to students was described in this study. Students also conceptualized learning to program as acquiring a new skill with application in real-life situations.

Taken together, the studies indicate that students can hold inclusive and hierarchical understandings of programming. Programming thinking is pointed out as a desirable learning outcome. From these studies emerges the possibility of learning outcomes that extend beyond learning the syntax of a programming language, to encompass programming as a way of thinking for problem solving and as a means of acculturation in the programming community.

A study on specific concepts in OOP [7] suggests that there are aspects that indicate desirable outcomes and dimensions of variations necessary for discernment of the nature of objects and classes. Viable understandings (object properties and instance variables) and misconceptions of learning [8] have also been identified. The way students experience learning in OOP was found to be related to the perception of the criteria for program correctness [9]. Educators can use knowledge of the outcomes of these studies to design learning experiences that bring these aspects to the focal awareness of students, and to enable students to reach a rich understanding of the object of learning.

III. Distilling Recommendations

Practical principles for programming teachers can be distilled from the phenomenographic studies on programming that were reviewed in Section II. Booth [19] suggests that posing programming problems that allow interpretation by students and demonstrating such interpretations in front of the class and in small groups are helpful for deepening understanding. Further suggestions include providing the opportunity and the learning environment in which variations in understanding come to light through focused student discussions conducted during the programming stages (interpretation of the problem, planning the program, critiquing the product, and debating the documentation).

When attending to the phenomenographic perspective of learning, Bruce et al., [5] urge teachers to design learning experiences, to develop teaching tools, and to tailor the curriculum to emphasize deep approaches to learning to program. They argue that it would be problematic if students did not move from being able to code, to understanding and integrating concepts and to developing problem solving skills. Specific recommendations are offered to help students to adopt more sophisticated ways of learning to program: emphasizing the breaking down of problems into smaller steps; focusing on the broad context of programming and programming languages; designing assignments in successive stages; motivating students by giving a choice of programming assignments and contextualizing the set assignments to make them more meaningful; providing ways (e.g. visual tools) to assist students to develop a sense of understanding, and building networking opportunities for the novice programmers [5].

Some other strategies that have been suggested [20] are to use variation theory in computing education to vary problems for a given construct/concept or to focus on a consistent problem but apply variations in the use of a construct or concept, or even to apply different constructs or concepts to the solving of that problem. Another strategy is to vary the programming language or development environment so that the learners discern that programming is not dependent on the programming language or programming paradigm.

IV. Phenomenography as Pedagogy

The related work reviewed so far has focused on specific objects of learning in programming. Pedagogical principles grounded in the phenomenographic perspective on learning and teaching that are described in this section focus on the relevance structure and architecture of variation at the macro or course level.

A. Structure of Relevance

A learner's experience of a learning situation creates a structure of relevance that influences the aim or direction of the learner, or when a learner is applying something learned [11]. The learning that occurs is dependent on how the learner personally experiences the relevance of the learning situation and therefore the teacher has to be aware of how and what the students are learning. From the phenomenographic perspective, it is suggested that the relevance structure be established from the

teacher's deliberations on the capabilities to be nurtured in the students, the role of prerequisites to the learning, and the overall learning goals [21]. The learners then become aware of the intended learning outcomes, the pathways to learning, and the demands of the learning tasks.

A student-centered approach to teaching that focuses on providing a relevance structure based on an awareness of students' current understandings and the type of learning approach adopted, encourages the development of deep understanding and the broadening of perspectives of important subject concepts [22]. Awareness of the students' understanding of different concepts empowers a teacher to address any concerns or needs of the students [2]. Changes in students' conceptions can also be made through an awareness of the misconceptions harbored by students [23].

Three approaches to enhancing the relevance of a course for students are discussed in Section V. These approaches focus on how the teacher can design the programming course based on pedagogical content knowledge, how the teacher can reveal the ways of learning by the students, and how the teacher relates and contextualizes student learning.

B. *Variation Theory*

In variation theory, the focus is on the ways of experiencing a phenomenon. In teaching, when a particular aspect of an object of learning is varied while other aspects are kept constant, the student becomes aware of the varying aspects [11]. A number of examples can be found in [24] of the practical application of the theory of variation in different subjects for creating: (a) variation in students' experiences of understanding of what is taught; (b) variation in teachers' ways of dealing with particular topics; and (c) a pedagogical design that influences teaching methods.

Four patterns of variation that focus on what varies and what is invariant in a learning situation have been identified as critical for learning. These patterns are contrast, generalization, separation, and fusion [24]. It is for the teacher to create these patterns of variation to enable the learner to discern the critical aspects of the object of learning. The teacher has to identify the critical aspects that present difficulties to the students, either from his or her own teaching experience or from previous phenomenographic studies. Students' preconceptions or beliefs can indicate to the teacher the critical aspects that present difficulties to the students, so that the teacher can then build on the students' prior understanding and experiences.

Students can also be made aware of the differences in understandings among their peers when dealing with the same object of learning. The teachers can design experiences to highlight these differences and help students to develop more powerful ways to learn. Furthermore, exposure to the critical aspects of professional situations, likely to be encountered in the future, enables students to develop holistic capabilities that link disciplinary knowledge and professional skills [25].

Technologies can also be used to help students to experience critical aspects of variation in the object of learning. The Conversational Framework [23], a model for learning with technologies, is underpinned by the phenomenographic perspective on student learning. In the twelve stage framework, the instructional process progresses from the curriculum aims, learning objectives, learning activities and media forms, to evaluative feedback. Technological media is incorporated for discursive activities, for discussions between teacher and student, for students and teachers to engage in adaptive and collaborative tasks, for interactive processes to give and act on feedback, and for stimulating reflective experiences.

By adopting optimal patterns of variation in dealing with the intended objects of learning in a lesson, and by paying attention to what varies and what remains invariant in a learning situation, teachers can help students to develop more powerful ways to learn [24]. Specifically, teachers can apply variation theory by emphasizing educationally critical aspects in the learning outcomes and assessments tasks, by introducing patters of variation in the learning and teaching activities, and by using learning media to enable students to experience variation in the object of learning. Section V gives examples of applying variation theory in a programming course design.

V. EXAMPLE COURSE DESIGN

In this section, the design of an introductory OOP course is described to show the building of a relevance structure and the use of variation theory. The course is taught at the University of Saint Joseph, Macau, using the objects-first approach with the Java programming language. In 2009, the course was redesigned as part of a doctoral study [26] to reflect the phenomenographic approach to learning and teaching.

The main principles of the example course design are summarized in Table 1 and the design choices drawn from the theory that has been discussed are explained in the following sections.

TABLE I. PHENOMENOGRAPHIC APPROACH TO COURSE DESIGN

Application of Phenomenographic Perspective	
Building structure of relevance	***Applying variation theory***
• Pedagogical content knowledge • Revealing the experience of ways of learning • Relating and contextualizing learning	• Educationally critical aspects • Progression of learning outcomes • Range of assessments tasks • Variation from learning and teaching activities • Variation through learning media

A. *Building the Structure of Relevance*

1) *Pedagogical Content Knowledge*

The content of the introductory OOP course is from the syllabus mandated for the program. However, the content

knowledge as it should be understood by the learners is based on the teacher's pedagogical experience of teaching the course and the research literature (that was discussed previously) on learning programming. The students' abilities to read and interpret code are seen as essential for novices to be able to design and build a new product.

In keeping with the phenomenographic perspective on differentiation between *what* the students learn and *how* the students learn [11], the intended object of learning is a specific skill or insight that is tied to the content that is to be learned. A distinction is made between *what* the students are expected to learn (the content of the java programming language i.e. the fundamental programming constructs: such as object, class, method, variables, types, expressions, and assignment; simple I/O; conditional and iterative control structures), and *how* they go about programming (recognize, trace, analyze code, design, and model real-life problems [27]).

2) Revealing the Experience of Ways of Learning

In the OOP course, questionnaires and reflective journals are used to gather the subject-specific prior knowledge and conceptions of objects and classes that students hold and to gauge students' motivation and self-efficacy levels.

A questionnaire given at the beginning of the course is designed to gather data about the student's major, prior knowledge of programming languages, prior perceptions of programming, motivation, and self-efficacy. The students are asked to select experiences [5] that apply to them from five different experiences of the act of learning to program (following, acting, conceptualizing, resourcing and taking intuitive steps), and from five different experiences of the object of learning to program: (coding, thinking differently, problem-solving, participating and satisfying clients). Students' motivation and self-efficacy before the course are gauged with questions adapted from [28].

Two other questionnaires seek to find out (a) the difficulties that students had in downloading and installing the software required for the programming course, and (b) the students' assessment of their strengths and weaknesses and their rating for their peers on the ability to share and contribute to pair and team work.

Journal writing to encourage metacognitive thinking is incorporated as a course requirement. Students reflect on their level of motivation and their learning process. The weekly journals focus on generative topics that are specifically chosen to think reflectively about programming related activities such as pair programming, team projects, peer evaluations, lecture-lab classes, java style guidelines, testing and documentation procedures, java libraries, graphical user interfaces, and program visualizations.

Based on the data from the questionnaires and feedback from the journals, teaching interventions are then planned (a) to respond to individual variation in learning outcomes and experiences; (b) to broaden the awareness of the nature of learning to program; (c) to enable immediate feedback and intervention for students with low levels of motivation or inability to cope with the course.

3) Relating and Contextualizing Learning

Effort is made during the course to relate the learning to the learner's own personal interests and motivations and to contextualize the set assignments to create authentic situations. For example, in the first week of the programming course, the students are made aware of the expected learning outcomes, the goals of the learning and teaching activities, and the demands on their time and effort. In the course orientation session, the focus is on the topics to be learnt, the software requirements, and the relevance of the course to everyday applications and future professional careers. Factors that contribute to success in programming are highlighted, tips on how to succeed in programming by being an active learner and team player are shared, and the need for professional work ethics is discussed.

Student preferences for adopting context related deep/surface approaches are kept in mind when designing the course. Deep approaches to learning OOP are encouraged through peer-managed, student-managed, and teacher-managed activities so that variations in understandings when dealing with the same object of learning are revealed and to create networking opportunities for the novice programmers. Peer-managed activities such as pair programming and programming team projects are incorporated. Student-managed activities include weekly quizzes and journal writing, and a final written exam. Teacher-managed tasks focus on lectures integrated with labs and demonstrations of the programming process. The teacher's chief responsibilities are not considered as delivering and testing programming content, but as motivating and supporting the learning process of the novice programmers.

Student-centered approaches, notably active and cooperative learning activities are planned to liven classes. Pair and team work is adopted to engage students in discussion while reading and writing code and during peer assessments and oral presentations. Pedagogical patterns [29] are employed to help students to relate to previous knowledge (consistent metaphor pattern), to make learning fun (role-play), to allow students to discover solutions (reflection pattern), to get students to explain concepts to others (explore for yourself pattern), and to allow students to experiment on their own (test tube pattern).

B. Applying Variation Theory

1) Educationally Critical Aspects

Patterns of variation are applied in teaching and learning activities focused on the critical aspects of OOP concepts. For example, the errors that novices make and the misconceptions that novices harbor about OOP [30] determine the critical aspects that are brought to the awareness of the students through the learning and teaching activities. The learning activities enable change in the capability for experiencing and being aware of the object of learning, while the teaching activities develop the

capability of discerning relevant aspects of the programming phenomenon and dealing with novel situations. Practice exercises and quizzes at different levels of proficiency are created to generate qualitatively different conceptual understandings [6] of the concepts of object and class. Guided practice activities and live demonstrations of the programming process are used in class to clarify programming concepts.

2) Progression of Learning Outcomes

Phenomenographic research [5] shows that to develop programming thinking, novice programmers should be encouraged to experience the range of different ways of learning to program. The long term goals for the introductory programming course are that the students should eventually develop problem-solving ability and design competence.

Programming related outcomes drawn from the matrix taxonomy [27] provide a progression of learning outcomes that span a range of known ways of learning to program. Students are expected to recognize the terminology and vocabulary of OOP and trace and interpret code samples. They are given opportunities to adapt or modify solutions and to present and explain the solution to others, so that they become familiar with using theory, practices and tools for problem solving (definition, specification, design, implementation, maintenance and evaluation of programs). The students are also given opportunities to design and model solutions using modularity and refactoring to demonstrate understanding of OO principles.

The emphasis in this course is also on the development of professional values and practice and on team and communications skills necessary for students in future careers. There is a holistic integrating of *what* programming concepts the students learn and *how* they go about learning and developing their skills, competencies, and attitudes.

3) Range of Assessments Tasks

A range of assessments tasks are designed to assess the progress of learning outcomes. The assessment tasks are matched with the learning outcomes to elicit the required performances of understanding. Individual work includes quizzes and an exam that tests knowledge and understanding of essential facts and concepts relating to OOP. Programming assignments that involve problem solving are done in pairs and groups to prepare students for the realities of working in multicultural programming teams. Making use of expected variation in the group, different perspectives are highlighted by the teacher in the feedback sessions, and the students gain knowledge from the variation within the group.

The use of SOLO [31] taxonomy-based criteria enables the holistic assessment of learning outcomes. The assessment of the structural complexity and relationships of programming knowledge is possible through SOLO, as it allows qualitative variations in understanding to be brought to the awareness of the teacher. The variations in expected performances in the SOLO assessment criteria provide opportunities to target higher understandings, while the assessment activities enable students the possibility to engage in the range of ways [5] of going about learning to program.

4) Variation From Learning and Teaching Activities

The learning and teaching activities are planned such that there is variation in situations within which learning tasks are met individually or in a group. Following the structure of variation, OOP concepts form the focal area, the teaching and learning activities are varied simultaneously, and the elimination of student errors and misconceptions remains invariant or constant throughout the implementation of the course.

Program design is made an integral part of the objects-first course with the use of object diagrams, use case diagrams, and CRC cards. The planning, designing, coding, and testing phases of the assignments also provide opportunities to help students to discern the variation in experience that is possible, and to develop a deeper understanding of programming beyond just writing code. Patterns of variation are employed to give students the opportunities to contrast, generalize, and separate aspects of programming assignments. The students then fuse all of these aspects to complete the term project to gain an understanding of the part-whole relationships. In these projects, the students model real world problems [6] and follow the entire programming process from analysis to design and implementation of code.

5) Variation Through Learning Media

The phenomenographic based conversational framework [23] that integrates use of media is adapted and extended for learning and teaching of programming concepts. Diverse narrative, discursive, adaptive, interactive, reflective, and collaborative experiences are made available to enhance the learning experiences. A Computing Augmented Learning Management system (CALMS) integrates OOP content within the Moodle learning management system. Lecture notes, code samples, and tutorial exercises are uploaded online. A variety of material is presented to appeal to different learning styles and to challenge more experienced students.

Helpful resources for novice programmers are made available in the form of learning objects, Java games, links to videos, animations, and multimedia tutoring, software manuals, and how-to guides. Support for OOP is provided with a range of interactive tools: (a) Jeliot for visualizations; (b) the JUnit extension to BlueJ for testing programs; (c) the Violet editor for UML diagrams; and (d) the JEWL library class for creating graphical user interfaces.

VI. DISCUSSION

The experiences of implementing the course are discussed in this section and the evaluations of the course design in terms of student experiences can be found in [32, 33].

Designing the OOP course with a phenomenographic approach has shown that teachers have to put in additional effort to investigate students' qualitatively different ways of experiencing the object of learning. A heightened awareness is

essential for identifying the variation in students' ways of experiencing. This awareness develops from pedagogical experience and is useful for identifying the critical features related to a particular way of experiencing programming.

A mindset shift is required for teachers to move to the phenomenographic perspective on learning and teaching. For those who teach with a strong focus on the syntax and semantics and underlying concepts of a particular programming language, the shift to an emphasis on the broader context of programming and programming language requires an understanding of how to build the structure of relevance and the architecture of variation in learning.

There are also some practical problems to adopting the phenomenographic perspective for teaching. Questionnaires and journal writings are beneficial to gain insight of the prior knowledge and conceptions that students hold, but require time and effort to create and implement. Effort is also required to evaluate and revise the learning experiences in the light of the variation in the learning outcomes. When learning resources and media are made available to the students and learning takes place outside the classroom, the intended object of learning as envisioned by the teacher and the lived object of learning as experienced by the student [11] may not overlap. The use of visualizations and interactive media, though beneficial for understanding some concepts related to classes and objects, can add to the cognitive load for students.

The preparation of a variety of informational sources, programming assignments, and learning activities can be daunting for teachers. Similarly, learning within and from student groups has to be carefully orchestrated so that the required dimensions of variation are revealed. Student engagement or learning cannot be guaranteed by simply varying the teaching and learning activities. The experienced variation should simultaneously focus on features critical for achieving a change in understanding.

The benefits of the holistic approach in encouraging a progression in the levels of thinking about the act of learning to program, and in thinking about the object of learning to program cannot be denied. However, not all of the students may fulfill the expectations of reaching a complete understanding of the full spectrum of learning outcomes in an introductory OOP course. The task to increase motivation and to spark learning in all the novice programmers can be challenging, but rewarding.

Specific recommendations for other teachers interested in adopting phenomenographic learning theory for the design of their programming courses are:

- Incorporate reflective journal writing and personal plans for students to explore their programming experiences and to express new understandings as well as misconceptions.
- Orient the students at the beginning of the course about the intended learning outcomes so they become more engaged in learning when they see the relevance of programming to their chosen major and future career needs.
- Offer a progression of competencies to achieve in programming assignments and projects that are interesting, challenging, and model real life situations. Encourage program planning and design along with learning the syntax of a programming language, so that students gain a broad perspective of programming.
- Integrate programming content with learning management systems or other social media to encourage discussion, debate, articulation, and expression of ideas. Educational media (IDEs, visualizations) offer opportunities to students to investigate, explore, experiment, and practice programming.
- Encourage feedback from peers, tutors, and software to help students to understand programming errors, and provide meaningful and timely feedback through adaptive and formative assessments.
- Probe for knowledge of students' prior experience with programming, their motivation for learning, and their self-efficacy, so that the course activities can be tailored to encourage deep learning.
- Explain and clarify ideas with examples, props, role-plays, visualizations, and active exercises. Incorporate collaborative learning to help students to develop vital interpersonal skills and discipline needed for professional contexts.
- Provide a learning environment that enables students to learn programming by themselves, from information sources, from the lecturer or tutor, and from their programming pair or team members.

VII. SUMMARY

The example of the introductory OOP course design has demonstrated how different learning experiences can be brought about by applying a structure of relevance and variation theory.

The building of the structure of relevance can be achieved by: (a) awareness of pedagogical content knowledge as it should be understood by learners; (b) ensuring learners reveal their experience of ways of learning (including the what and how of learning); (c) relating learning to the learner's own personal interests and motivations and contextualizing the set assignments to create authentic situations. In the learning context, patterns of variation can be employed to: (a) create awareness of educationally critical aspects for nurturing understanding; (b) allow a progression of learning outcomes that span the range of ways of learning; (c) provide a range of assessments tasks that assess the progression of learning outcomes; (d) offer learning and teaching activities such that there is variation in situations within individual and group work; (e) supply variation through learning media with networking opportunities for learning.

This paper has contributed to existing knowledge about computing education by proposing and describing the application of the phenomenographic perspective on teaching and learning at the macro or course level. Focus on a structure of relevance and on variation theory has been shown as a possible way to create a change in the learner's capability of experiencing a learning situation.

REFERENCES

[1] A. Berglund, *Learning Computer Systems in a Distributed Project Course: The what, why, how and where* vol. 62. Uppsala, Sweden: Acta Universitatis Upsaliensis, 2005.

[2] A. Berglund. (2006, June 1). Phenomenography as a way to research learning in computing. *Bull. Applied Computing and Information Technology 4(1)*. Available: http://www.naccq.ac.nz/bacit/

[3] A. Berglund and M. Wiggberg, "Students learn CS in different ways: Insights from an empirical study," *Proc. 11th Annual Conf. Innovation and Technology in Computer Science Education (ITiCSE)*, Bologna, Italy, June 2006, pp. 265-269.

[4] S. Booth, *Learning to Program: A Phenomenographic Perspective*. Göteborg, Sweden: Acta Universitatis Gothoburgensis, 1992.

[5] C. Bruce, C. McMahon, L. Buckingham, J. Hynd, M. Roggenkamp, and I. Stoodley, "Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university," *J. Information Technology Education,* vol. 3, pp. 143-160, 2004.

[6] A. Eckerdal and A. Berglund, "What does it take to learn 'programming thinking'?," *Proc. Int. Workshop Computing Education Research (ICER)*, Seattle, WA, USA, Oct. 2005, pp. 135-142.

[7] A. Eckerdal and M. Thuné, "Novice Java programmers' conceptions of "object" and "class", and variation theory," *Proc. 10th Annual Conf. Innovation and Technology in Computer Science Education (ITiCSE)*, Caparica, Portugal, June 2005, pp. 89-93.

[8] J. Sorva, "Students' understandings of storing objects," in *Proc. of the 7th Baltic Sea Conference on Computing Education Research*. vol. 88, R. Lister and Simon, Eds., Koli, Finland: Australian Computer Society, Sydney, NSW., 2007, pp. 127-135.

[9] I. Stamouli and M. Huggard, "Object oriented programming and program correctness: The students' perspective," *Proc. 2nd Int. Workshop Computing Education Research (ICER)*, Canterbury, UK, Sept. 2006, pp. 109-118.

[10] T. Kroksmark, *Fenomenografisk Didaktik* vol. 17. Göteborg: Jönköping University Press, 1987.

[11] F. Marton and S. Booth, *Learning and Awareness*. Mahwah, NJ: Laurence Erlbaum Associates, 1997.

[12] E. Thompson, L. Hunt, and K. Kinshuk, "Exploring learner conceptions of programming," in *Proc. 8th Australian Conference on Computing Education*. vol. 52, D. Tolhurst and S. Mann, Eds., Hobart, Australia: Australian Computer Society, 2006, pp. 205-212.

[13] G. S. Åkerlind, "A phenomenographic approach to developing academics' understanding of the nature of teaching and learning," *Teaching in Higher Education,* vol. 13, pp. 633-644, Nov. 2008. doi: 10.1080/13562510802452350.

[14] K. Trigwell, M. Prosser, and P. Ginns, "Phenomenographic pedagogy and a revised approaches to teaching inventory," *Higher Education Research & Development,* vol. 24, pp. 349-360, Nov. 2005. doi: 10.1080/07294360500284730.

[15] M. Thuné and A. Eckerdal, "Variation theory applied to students' conceptions of computer programming," *European J. Engineering Education,* vol. 34, pp. 339-347, July 2009. doi: 10.1080/03043790902989374.

[16] E. Thompson, "From phenomenography study to planning teaching," *Proc. 15th Annual Conf. Innovation and Technology in Computer Science Education (ITiCSE)*, Bilkent, Ankara, Turkey, June 2010, pp. 13-17.

[17] S. Booth, "Learning to program as entering the datalogical culture: A phenomenographic exploration," in 9th European Association for Research on Learning and Instruction Conf. (EARLI), Fribourg, Switzerland, 2001.

[18] C. Bruce, C. McMahon, L. Buckingham, J. Hynd, and M. Roggenkamp, "Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university.," Faculty Information Technology, Queensland University of Technology, Brisbane, Australia, 2003.

[19] S. Booth, "On phenomenography, learning and teaching," *Higher Education Research & Development,* vol. 16, pp. 135-158, 1997. doi: 10.1080/0729436970160203.

[20] J. Suhonen, E. Thompson, J. Davies, and K. Kinshuk, "Applications of variation theory in computing education," *Proc. 7th Baltic Sea Conf. Computing Education Research*, Koli, Finland, Nov. 2007, pp. 217-220.

[21] M. L. Lo and W. Y. Pong, "Catering for individual differences: Building on variation," in *For Each and Everyone: Catering for Individual Differences through Learning Studies*, M. L. Lo, W. Y. Pong, and C. P. M. Pakey, Eds., Hong Kong: Hong Kong University Press, 2005, pp. 9-26.

[22] C. Cope. (2003,). Educationally critical characteristics of deep approaches to learning about the concept of an information system. *J. Information Technology Education 2,* 415-427. Available: http://jite.org/

[23] D. Laurillard, *Rethinking University Teaching: A Conversational Framework for the Effective Use of Educational Technology*, 2nd ed. London, UK: RoutledgeFalmer, 2002.

[24] F. Marton and A. Tsui, *Classroom Discourse and the Space of Learning*. Mahwah, NJ: Lawrence Erlbaum Associates, 2004.

[25] J. Bowden and F. Marton, *The university of learning: Beyond quality and competence*. London, UK: Routledge, 1998.

[26] N. Thota, "Developing a holistic approach to learning and teaching introductory object-oriented programming," Ph.D dissertation, School of Intelligent Systems and Technology, Univ. of Saint Joseph, Macau, 2011.

[27] U. Fuller, C. G. Johnson, T. Ahoniemi, D. Cukierman, I. Hernán-Losada, J. Jackova, E. Lahtinen, T. L. Lewis, D. M. Thompson, C. Riedesel, and E. Thompson, "Developing a computer science-specific learning taxonomy," *ACM SIG. Bull.,* vol. 39, pp. 152-170, Dec. 2007. doi: 10.1145/1345375.1345438.

[28] L. Soh, A. Samal, and G. Nugent, "An integrated framework for improved computer science education: Strategies, implementations, and results," *Computer Science Education,* vol. 17, pp. 59-83, Mar. 2007. doi: 10.1080/08993400701203782.

[29] J. Bergin, J. Eckstein, M. L. Manns, and E. Wallingford, "Patterns for gaining different perspectives," *Proc. 8th Conf. Pattern Languages of Programs (PLoP)* Monticello, IL, Sept. 2001.

[30] N. Ragonis and M. Ben-Ari, "A long-term investigation of the comprehension of OOP concepts by novices," *Computer Science Education,* vol. 15, pp. 203-221, Sept. 2005. doi: 10.1080/08993400500224310.

[31] J. B. Biggs and K. F. Collis, *Evaluating the Quality of Learning: The SOLO Taxonomy*. New York, NY: Academic Press, 1982.

[32] N. Thota and R. Whitfield, "Holistic approach to learning and teaching introductory object-oriented programming," *Computer Science Education,* vol. 20, pp. 103-127, Jun. 2010. doi: 10.1080/08993408.2010.486260.

[33] N. Thota, "Repertory grid: Investigating personal constructs of novice programmers," *Proc. 11th Koli Calling Int. Conf. Computing Education Research*, Koli, Finland, Nov. 2011, pp. 23-32.