1st International Conference on Higher Education Advances, HEAd'15
Universitat Politècnica de València, València, 2015
DOI: http://dx.doi.org/10.4995/HEAd15.2015.356

Development of a Classification Scheme for Errors Observed in the Process of Computer Programming Education

D. Zehetmeier* **, A. Böttcher, A. Brüggemann-Klein*, V. Thurner****

* Technische Universität München, Faculty of Informatics
Boltzmannstraße 3, D-85748 Garching
** Munich University of Applied Sciences, Faculty of Computer Science and Mathematics
Lothstr. 64, D-80335 München

Abstract: Every semester, we observe more or less the same principal difficulties among our students who are striving to learn the intricacies of software development. Basically, they run into the same kind of errors throughout their learning process as previous student generations. Based on this fact, we suspect that there is a set of underlying problems which are causing these errors. Our goal is to identify and tackle these basic problems, in order to deal with errors effectively in our teaching and coaching activities, rather than merely treating observable symptoms. To achieve this, we develop a comprehensive and topic-independent error classification scheme and employ this to classify errors found in literature and in our own courses. This classification scheme is mainly based on the cognitive dimensions of the revised Bloom's taxonomy for educational objectives. Each error is based on a deficiency in certain competencies. Therefore, it is possible to develop a set of interventions for each error class, which focuses on the specific deficits that are the main cause for all the errors of this class.

Keywords: Computer Science Education; Student assessment

Introduction

Many years of experience in teaching software development and software engineering in higher education have shown that over and over again, a significant part of each new generation of students runs into more or less the same principal difficulties and produces the same kind of errors throughout their learning process. Informal discussions with colleagues (both national and international), as well as literature research, implies that many of these difficulties and errors occur universally and thus seem to be of a more general nature, rather than being caused by our own individual teaching styles.

For example, the notion of 'if-loop' (rather than if-then-else being a 'choice') is still widely spread, although all the lecturers whom we have personally asked so far swear that they have never ever said 'if-loop' when students were present.

As the work of a significant number of students shows the same errors, we suspect that there is some set of underlying problems that is responsible for causing these errors. To deal with these errors in an effective way, we therefore have to identify these basic problems, rather than merely curing the symptoms that have been observed in individual cases.

Goals

Students' difficulties and errors observed in the process of computer science education and, more specifically, software development education, are manifold. In fact, there are so many of them that it is almost impossible to deal with each error individually.

Instead, we attempt to categorise the observed difficulties and errors and cluster them into classes. As key criterion for this classification, we use the underlying problem that causes the observed difficulties and errors. More precisely, we focus on which basic

competency is insufficient or lacking completely, but would avoid the problem if it were sufficiently developed.

To validate our classification scheme, we categorise a number of typical errors, both from literature and those identified in class work of our own students.

Thus, firstly we aim to understand the underlying causes of the observed errors. Secondly, we boil them down to a manageable set of crucial competencies that must be sufficiently developed in our students, so that they are able to effectively acquire the computer science related expertise required for the academic degree they aspire to. On this basis, we plan to develop a set of interventions which systematically address those base competencies that are identified as being crucial, but missing in our students.

State of the Art

Error classification schemes have already been investigated to a certain extent. To gain a comprehensive overview of typical faults and 'things that are done wrong', we extended our literature review on *errors* to the general area of STEM (science, technology, engineering, and mathematics). However, as classification schemes tend to be more domain-specific, we focused our review of existing *classifications* on the domain of computer science.

Errors and Misconceptions

The most comprehensive term for 'things that are done wrong' is the concept of *error*. An error is "the state or condition of being wrong in conduct or judgment" (Dictionary). For example, an error in an exam is everything that is incorrect, or a missing answer where an answer would be required. A study that extensively uses this term is published by Hristova et al. (Hristova, Misra, Rutter, & Mercuri, 2003).

Another common term is *misconception*. It is often used in scientific papers, but usually not defined explicitly. Misconceptions and their influence on teaching are discussed in many disciplines, like physics, chemistry (Barke, 2006), biology (Dreesmann, Graf, & Witte, 2012) or computer science (Pea, 1986).

For the notion of misconception, many synonymous terms are used in literature, such as *alternative conceptions* (Barke, 2006), *preconceptions* (Barke, 2006), *naive beliefs* or *bugs* (Pea, 1986). Bahar (Bahar, 2003) states that the term *misconception* is widely used in research, is well-known to the public, and indicates that the concept in a student's mind differs from the scientific concepts. Therefore, we adopt this expression throughout this work.

If the students' existing ideas are a misconception rather than correct knowledge, problems will occur when new content in this area is provided, e.g. when the misconception runs contrary to the true scientific concept. As a consequence, as all new information is interpreted based on existing knowledge, undetected misconceptions will seriously inhibit the learning process.

In our opinion, Dreesmann (Dreesmann, Graf, & Witte, 2012) uses the most complete definition, naming all relevant characteristics of a misconception. Accordingly, in this paper we use the following definition translated from (Dreesmann, Graf, & Witte, 2012):

Misconception is a logical and coherent concept. Thus, it fits into personal experience and knowledge. Nevertheless, it is wrong or contrary to scientific concepts.

Known Error Classifications

Most of the errors described in literature are either not classified at all (such as Humbert (Humbert, 2006) and Rabel (Rabel, 2011)), or classified by a schema based on the content domain, in whose teaching the error occurred.

For example, Sorva (Sorva, 2008) uses three classes of errors: understanding of variables, understanding of object variables, and understanding of the relationship between primitive and object variables. These error classes are specific for the understanding of data storage topics, but would not work with algorithms, for example.

The paper "Exploring Programming Misconceptions" (Sirkiä & Sorva, 2012) investigates different types of errors, classified as: miscellaneous basic concepts, functions, or object-oriented programming. Thus, they are following the topics of introductory courses on software development.

As a consequence, errors in more advanced topics, like threading or generics, are difficult to classify, as they usually involve a variety of problems from different fundamental topics.

In contrast, Hristova (Hristova, Misra, Rutter, & Mercuri, 2003) and Pea (Pea, 1986) introduce more general classifications. Pea (Pea, 1986) derives three classes of errors from one "superbug". The superbug describes that many students implicitly assume that a computer can think, or interpret, or has a mind. From this initial superbug, students derive a variety of erroneous notions; for example, that different lines of code can be active at the same time; or that a program can act in foresight; or that computers can do something that has not been specified in the program.

In addition, Hristova et al. (Hristova, Misra, Rutter, & Mercuri, 2003) classify errors according to a schema which is well-established in computer science, i.e. into syntactic, semantic and logic errors.

Syntax errors are the ones based on misspelling, punctuation and word order in a program.

Semantic errors occur on a higher intellectual level. They deal with the meaning of the code and arise from mistaken ideas of how a programming language interprets certain instructions.

Finally, *logic errors* are the most general type of error, as they result from the programmer's misguided thinking, rather than from language characteristics.

Detection and Clustering of Errors

In order to use any classification scheme, it is necessary to be aware of the errors that have to be classified. Several methods to identify errors are known from literature:

Sirkiä (Sirkiä & Sorva, 2012) analyses solutions that students submitted to a system for Visual Program Simulation (VPS). In contrast to this, in the study of Sorva (Sorva, 2008), students are interviewed.

To identify and cluster errors, we use the following techniques:

- Similar to (Sirkiä & Sorva, 2012), we analyse solutions that students turned in as assignments or in exams. To this end, we go through the pile of solutions twice. In the first iteration, we scan through all the solutions, to identify and take notes on the observed errors. Then, we organise the observed errors into clusters, leading to a collection of some three to eight main clusters. In the second iteration, we resurvey all the solutions and assign each erroneous solution to an error cluster. Having accomplished this, we are able to count the number of hits for each cluster. The higher the number of solutions in a cluster, the more important it is to identify the cause of the error and to find an adequate remedy. This process is depicted in Figure 1.
- We observe our students through lab sessions and try to analyse and categorise their errors on the fly. In this setting, we are able to immediately ask them questions regarding their solutions. Thus, we can retrieve valuable information on underlying problems that led to their errors.
- In our previous teaching experience, we have collected errors that are made by a significant number of programming novices. If we are aware of an error that frequently occurs in a specific context, we ask questions or design assignments in a way that is likely to provoke this error, thereby making the underlying problems visible. Thus, we help students to become aware of their error, to reflect on their own thinking and, finally, to reach a correct solution on their own.

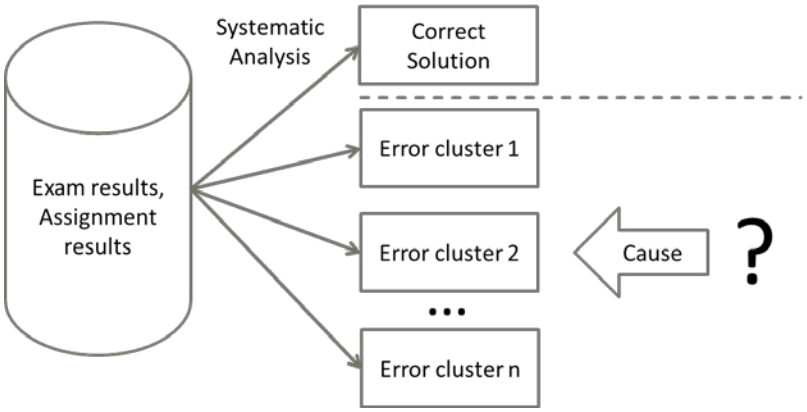

Figure 1. Error identification process

As an example, we look into one of the initial exercises that our first year students have to solve in their introductory course on software development. Here, students are required to implement a Java class *Sheep*, to represent and draw a sheep composed from several ellipses, which are used as basic shapes. We provide a class for drawing ellipses, hints on coordinates of the sheep's parts, as well as a screenshot showing a prototype of our sheep. In the exercise, the sheep is required to change its colour and its position.

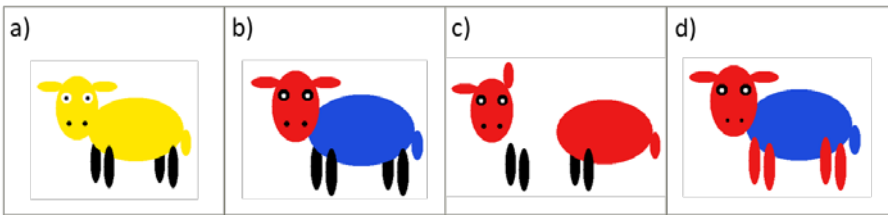

Figure 2. Different examples of sheep programmed and turned in by our students. We see one correct variant (a) and three kinds of errors: Colour hard-coded (b), insufficient passing of parameters for coordinates (c), and wrong drawing order (d).

Figure 2. (a) depicts a correct solution, where all the woolly parts of the sheep are of the same colour and the sheep is topologically correct. Sheep (b) is multi-coloured. When the sheep is moved, sheep (c) has its torso severed from the rest of its body parts. Sheep (d) is multi-coloured like sheep (b), and has all of its four legs in front of its body, rather than two legs on the off-side and two on the side facing the front. In addition, several students did not have any clue what to do, and were unable to create anything at all. Each of these errors represents one error cluster.

For all the identified and clustered errors, in a next step we try to conjecture the reason why students solved the problem exactly in the way they did. This is crucial for finding the cause of this kind of error, which, in turn, is a prerequisite for successfully dealing with the principal error, rather than just correcting mere symptoms.

To achieve this, we look into the technical realisation of the different sheep error classes. Here, we discover that the error in sheep (b) is caused by hard-coding colour values by copy-and-paste, rather than parameterising the colour information. Sheep (c) was butchered by not passing parameters for coordinates from the body to the parts, whereas the creators of sheep (d) first drew the sheep's body and all the sheep's legs afterwards, being unaware that the sequential processing order of the different statements influences the final result.

Obviously, the visible error symptoms are highly specific to the exercise in question. However, the underlying problems as well as their *causes* are of a more general nature. Therefore, as a next step we develop a scheme to classify errors according to their causes.

Classification Scheme

We deem the error classifications described in literature so far to be insufficient, as they are highly specialised and thus do not provide a single general classification approach that is suitable for a large variety of errors.

When searching for a more comprehensive and topic-independent classification scheme, our central idea is to relate error causes to the competency levels and categories of cognitive processes that were defined by the revised Bloom's taxonomy (Anderson, et al., 2001). In our teaching experience, this taxonomy has proven to be a suitable basis for describing teaching goals. Usually, the revised Bloom's taxonomy is used to describe competencies on different skill levels. Analogously, it is possible to categorise the identified deficiencies according to the corresponding Bloom levels.

More precisely, the revised Bloom's taxonomy focuses on the cognitive domain. It is structured into six increasingly complex levels called *cognitive process dimensions* (see Table 1.), which classify the learners' thinking behaviour. Each of these dimensions has several sub-dimensions, to allow for a more detailed clarification of the levels (Anderson, et al., 2001). Furthermore, Anderson et al. distinguish four general types of knowledge, i.e. *factual, conceptual, procedural* and *metacognitive knowledge*.

Table 1. Cognitive process dimensions according to (Anderson, et al., 2001), and their definitions.

Level	Categories and cognitive processes	Definition
1	REMEMBER	Retrieve relevant information from long-term memory.
2	UNDERSTAND	Construct meaning from instructional messages. Build connections between new information and prior knowledge.
3	APPLY	Locate and use procedures to perform exercises (familiar and routine approaches) and solve problems (procedure initially not known).
4	ANALYSE	Break material down to its components and identify how the parts are related and what is the overall structure.
5	EVALUATE	Make judgments based on criteria (e.g. quality, effectiveness, efficiency and consistency) and standards.
6	CREATE	Put elements together to form a new product. Mentally reorganise parts into a pattern not clearly presented before.

In our error classification scheme, we organise errors in a way that is similar to Bloom's taxonomy, in that it specifies what kind of competency is lacking, thus causing the observed error (see Table 2.). In addition, we name and characterise each error class and relate it to its corresponding Bloom level, i.e. the competence level that is deficient if this error occurs.

Lack of accuracy (sloppiness) is independent of all the cognitive processes described in the revised Bloom's taxonomy. Therefore, we introduce another category that is below all of those categories defined by (Anderson, et al., 2001). Inspired by Donald E. Knuth (Knuth, 1989), we name it *MENTAL TYPO*, indicating a lack of concentration, accuracy or, as Knuth says, 'less brainpower left for small details'. An example would be to leave out the brackets after a method call or to forget a semicolon.

KNOWLEDGE GAP is the second class of error. It correlates to Bloom's REMEMBER level. Typical deficits would be not knowing one's type of learner, too little diligence or not learning definitions by heart. In a context of informatics education, this could occur if students do not know the definition of the terms class and object. Furthermore, it could be that students think Java String is a primitive data type, as they REMEMBER a misconstrued and thus incorrect definition.

The third class is called *MISCONCEPTION*, which is partly what has been defined in section 'State of the Art'. The new definition includes wrong and missing connections.

Table 2. Classification scheme for errors, including a description of the error, the underlying deficits as well as the competencies that are lacking in each case.

Error Class		Description	Deficits in Base Competencies (Lack of)
6	*LACK OF INNOVATION*	Inability to imagine a new product or synthesise a new solution from known information and methods.	Ability to synthesise a lot of information (being creative, being innovative)
5	*QUALITY GAP*	Inability to evaluate software (either self-created or ready-made) against general quality standards.	Pragmatism, transfer general criteria to specific example (thinking concretely, thinking critically)
4	*STRUCTURAL BLINDNESS*	Inability to distinguish components and their internal interaction, in a given setting.	Structuring unknown/external data according to a systematic and methodical approach (being able to structure)
3	*WRONG CHOICE*	Wrong problem classification and selection of solution process.	Assignment of a problem to a solution process (decision-making, evaluating), context-sensitivity (thinking holistically, analytically)
2	*MIS-CONCEPTION*	Faulty concept in mind that fits into previous personal experiences, or not understanding a concept at all.	Correct connection between new information and previous knowledge (thinking holistically)
1	*KNOWLEDGE GAP*	Not knowing definitions or terms.	Diligence, knowledge about oneself (e.g. type of learner) (being able to reflect)
0	*MENTAL TYPO*	Sloppy work	Concentration, brainpower to get the small details right, nervousness (being accurate, being focused, being efficient)

Thus, if students interpret new information in a wrong way, and form their understanding on this basis, this results in a misconception. Furthermore, not understanding an issue or a topic is also a misconception in this schema. Hence, students with an error in this class were not able to connect new information correctly with previous knowledge, or they built wrong connections. One common example is that students are often unable to distinguish between identity and equality, which is an important concept in many programming languages.

WRONG CHOICE is the term selected for the fourth class. It indicates as a deficit a faulty mapping of a problem to the solution process, and vice versa. This error can have two reasons: a wrong problem classification or the selection of an inappropriate solution mechanism. Using an **`enum`** instead of **`inheritance`** is an example of this error class.

Error class five describes the inability to identify and to distinguish components and their internal interaction, in a given setting. Thus, we call this error class *STRUCTURAL BLINDNESS*. An instance of this error class is that students are unable to understand or debug external code. Another example is that students have difficulties in analysing a task description. The underlying deficit is the inability to structure unknown content, a lack of identifying structure and the inability to work systematically and methodically.

A *QUALITY GAP* occurs if a deficit in pragmatism exists. Another error that is rather specific for the area of computer science is a transfer problem. Students are not able to transfer quality standards concerning software to their personal or unknown code. This could appear in code snippets like `FIVE = 5;` for a definition of a constant value. More generally, any student who writes code that is logically correct, but does not meet the quality standards, has a quality gap – provided we are sure that they had already been taught about quality standards (otherwise this could also very well be a knowledge gap).

The final error class is *LACK OF INNOVATION*, corresponding with Bloom's level CREATE. For example, students are unable to create an appropriate algorithm to solve a specific task. Furthermore, if a student copies another student's solution or does not hand in anything at all, this is included in this error class as well. Deficits behind this error might be insufficient creativity or the inability to synthesise individual pieces of information.

Examples for the Assignment of Errors in the Scheme

To demonstrate the classification process as well as the universality of the scheme, we classify errors both from literature and from our own courses. As an example, we refer to the papers of Hristova et al. (Hristova, Misra, Rutter, & Mercuri, 2003), Sorva (Sorva, 2008) and Pea (Pea, 1986).

- *Unbalanced brackets* (Hristova, Misra, Rutter, & Mercuri, 2003): This error is caused by sloppiness. Thus, it is a *MENTAL TYPO*.
- *Java String is a primitive data type* (Sorva, 2008): This is a *KNOWLEDGE GAP*. Students learned a faulty definition, or did not learn the definition at all, even though they had been provided with the correct definition.
- *Not knowing the meaning of an object declaration* (Sorva, 2008): This is also a *KNOWLEDGE GAP*. Students are not able to remember the definition of a declaration.
- *Computer knows different lines at the same time* (Pea, 1986): This is a *MISCONCEPTION* as students have a faulty understanding of how a computer really works.
- *Code has more meaning than it actually has* (Pea, 1986): This is also a *MISCONCEPTION*, as students have a faulty understanding, which might originate from their human interaction experiences. The communication partner interprets much more than just the spoken words, such as facial expression, gesture and the context of the conversation. Thus, statements within a conversation are interpreted. If students transfer this understanding to a computer, they are not grasping the fact that a computer needs precise and self-contained instructions.
- *Only move the Sheep's body* [course]: This is a *MISCONCEPTION*, too. Here, students are actually transferring their knowledge from everyday experience: If they move their body, the head and everything else follows automatically. In programming, this has to be explicitly expressed in the code.
- *While-loop instead of if-statement* [course]: In our courses, it sometimes happens that students use a while-loop instead of an if-statement. They evaluate the condition and do something once. In order to achieve this, they change the condition variable within the loop, so that the loop's body is executed only once. This is an error which belongs in the *WRONG CHOICE* category, as students decided to use an inappropriate construct, although he or she had already learned the appropriate one.
- *Not revealing the object structure* [course]: In the assignment 'Sheep', many students just used the basic parts of the sheep and put them together in the main method, rather than hierarchically structuring them into more complex objects such

as eyes or head. For example, eyes have an iris and a pupil. Correspondingly, a head includes nostrils, eyes and ears. Note that the required parts and their relationships were depicted in a UML-diagram. Nevertheless, students simply ignored these relationships. Therefore, this error is of type *STRUCTURAL BLINDNESS*.

- *20 conditions in if-statement instead of loop* [course]: In our practical course, students used 20 conditions in an if-statement to check whether a word (with a maximum length of 40 characters) is a palindrome or not. Although the code worked fine, this is a bad programming style. Hence, students have a *QUALITY GAP*.
- *Duplicate code* [course]: We mentioned above that students often have duplicate code in their programs. Classification schemes from literature do not cover this type of error. Within our schema, it is a *QUALITY GAP*. Although the produced code works correctly, it neither meets known quality standards, nor conforms with proven practice. As students already had lecture units on good programming style, they should have known better.
- *Not solving a problem* [course]: One of our assignments had two parts, which were only slightly different. All the students solved the first part. Although the second part did not require any additional expert knowledge, only half of the students managed to solve the second task. The difficulty was that they had never had a similar task before. Hence, they were unable to develop a new solution. This indicates that students have a *LACK OF INNOVATION*.

Conclusion

Summing up, we first described an approach to find error clusters based on students' exam or assignment results. As a next step, we introduced our error classification scheme and applied this to classify these errors. Finally, we sketched the underlying causes of the observed error clusters.

The developed classification scheme is based on the well-known revised Bloom's taxonomy (Anderson, et al., 2001). We applied it for classifying errors that we observed in the process of computer science education. In addition, we mapped deficits to our teaching goals, as both are based on the same taxonomy. The resulting classification scheme is comprehensive and topic-independent. It is possible to classify all the errors found in literature and throughout our courses. As a set of examples, we classified around further 30 errors during our research.

A benefit of this scheme is that new errors can be classified by various people. Thus, it is not necessary to present a complete list of all possible errors. Rather, any person able to follow our classification process can classify errors. To verify whether our classification scheme is applicable in a more general way, in future work we will attempt to classify errors from other disciplines.

On the basis of our error classification scheme we can now explicitly create teaching goals that specifically address common errors. Currently, we are looking for existing interventions, as well as developing new ones, for each error class of the scheme. These interventions should focus on typical underlying deficits for each error class. Furthermore, interventions must be general enough to be appropriate for every error correctly classified in the corresponding class.

References

Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Mayer, R. E., Pintrich, P. R., . . . Wittrock, M. C. (Eds.). (2001). *A Taxonomy for Learning, Teaching, and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives* (1 ed.). New York: Longman.

Bahar, M. (2003). Misconceptions in Biology Education and Coneptual Change Strategies. *Educational Sciences: Theory & Practice, 3*, 55-64.

Barke, H.-D. (2006). *Chemiedidaktik - Diagnose und Korrektur von Schülervorstellungen.* Springer.

Dictionary, O. E. (n.d.). Misconception. *Misconception*. Retrieved from http://www.oed.com/

Dreesmann, D. C., Graf, D., & Witte, K. (2012). *Evolutionsbiologie - Moderne Themen für den Unterricht.* (S. B. Heidelberg, Ed.) Spektrum Akademischer Verlag. Retrieved from http://link.springer.com/book/10.1007/978-3-8274-2786-1

Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003). Identifying and Correcting Java Programming Errors for Introductory Computer Science Students. *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education* (pp. 153-156). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/611892.611956

Humbert, L. (2006). *Didaktik der Informatik - mit praxiserprobtem Unterrichtsmaterial.* Teubner. Retrieved from http://link.springer.com/book/10.1007/978-3-8351-9046-7

Knuth, D. E. (1989). The errors of TEX. *Software: Practice and Experience, 19*(7), 607-685.

Pea, R. D. (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research, 2*, 25-36.

Rabel, M. (2011). Grundvorstellungen in der Informatik. *Informatik mit Kopf, Herz und Hand - Praxisbeiträge zur 14. GI-Fachtagung "Informatik und Schule" (INFOS2011)*, (pp. 61-70).

Sirkiä, T., & Sorva, J. (2012). Exploring Programming Misconceptions: An Analysis of Student Mistakes in Visual Program Simulation Exercises. *Proceedings of the 12th Koli Calling International Conference on Computing Education Research* (pp. 19-28). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2401796.2401799

Sorva, J. (2008). The Same but Different - Students' Understandings of Primitive and Object Variables. *Proceedings of the 8th International Conference on Computing Education Research* (pp. 5-15). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1595356.1595360