Evaluating OO Example Programs for CS1

Jürgen Börstler
Dept. of Computer Science
University of Umeå, Sweden
jubo@cs.umu.se

Henrik B. Christensen
Dept. of Computer Science
University of Aarhus, Denmark
hbc@daimi.au.dk

Jens Bennedsen
IT University West
Aarhus, Denmark
jbb@it-vest.dk

Marie Nordström
Dept. of Computer Science
University of Umeå, Sweden
marie@cs.umu.se

Lena Kallin Westin
Dept. of Computer Science
University of Umeå, Sweden
kallin@cs.umu.se

Jan Erik Moström
Dept. of Computer Science
University of Umeå, Sweden
jem@cs.umu.se

Michael E. Caspersen
Dept. of Computer Science
University of Aarhus, Denmark
mec@daimi.au.dk

ABSTRACT

A significant part of learning to program is to be exposed to examples that may work as templates, guidelines and inspiration for the students' own programs. It is therefore important that textbooks provide high quality examples. In this paper, we discuss properties of example programs that might affect the teaching and learning of object-oriented programming. Furthermore, we present an evaluation instrument for example programs and report on initial experiences of its application to a selection of examples from popular introductory programming textbooks.

Categories and Subject Descriptors

K3.2 [**Computers & Education**]: Computer and Information Science Education—*computer science education*

General Terms

Experimentation, Human Factors, Measurement

Keywords

CS1, example programs, object-orientation, quality

1. INTRODUCTION

Examples are important tools for teaching and learning. Both students and teachers cite example programs as the most helpful materials for learning to program [9]. Research in cognitive science confirms that "examples appear to play a central role in the early phases of cognitive skill acquisition" [18]. Moreover, research in cognitive load theory has shown that worked examples play an important role in order to increase learning outcome [5]. With carefully developed examples, we can better avoid misconceptions [4, 7].

There are two major quality aspects of examples: technical quality and didactical (i.e., pedagogical) quality. Examples work as role models; novices use examples as templates for solving new problems [15]. Exemplification is a well researched topic in mathematics education [10] where "the choice of examples that learners are exposed to plays a crucial role in developing their ability to generalize" [21]. Examples must therefore be consistent with all learning goals; follow the principles, guidelines, and rules we want to instill in our students; adhere to the course design and the expected level of the student. Otherwise, students will have a difficult time recognizing patterns and telling an example's non-essential properties (noise) from those that are structurally or conceptually important. It is therefore important to present examples in a way that conveys their "message", but at the same time be aware of what learners might actually see in an example [12].

In this paper, we discuss essential properties of example programs and formulate criteria which are used to develop an evaluation instrument. We then present the results of using this instrument on a selection of program examples from popular textbooks. Finally, we discuss the lessons learned from performing this evaluation and outline how our work can be taken further.

2. RELATED WORK

Although examples are perceived as one of the most important tools for the teaching and learning of programming, there is very little research in this area. Most often example issues are only discussed in the narrow context of a single simple and concrete example, like the recurring "Hello World"-type discussions [6, 19], or they are regarded as a language issue [2, 14]. Only few authors have taken a broader view by investigating features of example programs and their (potential) effects on learning.

Wu et al. [20] studied programming examples in 16 high school computer textbooks and concluded that most of them "lacked detailed explanation of some of the problem-solving

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ITiCSE'08 '08 Madrid, Spain
Copyright 2008 ACM X-XXXXX-XX-X/XX/XX ...$5.00.

steps, especially problem analysis and testing/debugging". Almost half of the examples fell into either the math-problem (27%) or syntax-problem (21%) category.

Holland et al. [8] provide guidelines for designing example programs to prevent object-oriented misconceptions, which are successfully used by Sanders and Thomas [16] for assessing student programs.

Malan and Halland [11] describe four common pitfalls that should be avoided when developing example programs. They argue that examples that are too abstract or too concrete, that do not apply the taught concepts consistently, or that undermine the concept they are introducing, might hinder learning.

Furthermore, there are many studies of software development in general showing that adherence to common software design principles, guidelines, and rules [3], as well as certain coding, commenting, naming guidelines, and rules [13, 17] support program understanding.

There is also a large body of research on worked examples providing general guidelines regarding the form and presentation of examples [5].

However, to our knowledge, neither of the above principles, guidelines, and rules have been used to evaluate example programs from programming textbooks.

3. RESEARCH APPROACH

This project is carried out by two research groups from two different countries.

During an initial two-day workshop, a large number of example programs from different textbooks were discussed to identify common strengths and weaknesses. The goal was to define a set of criteria to effectively discriminate between different levels of "quality", based on accepted principles, guidelines, and rules from the literature (see Section 2) and our own teaching experience. The outcome of this workshop was an initial evaluation instrument and a test set of textbook examples. In the context of this work an example is considered as a complete application or applet plus all supporting explanations related to this particular program (in contrast to code fragments). The instrument was tested on two examples by four reviewers, which lead to several revisions of the instrument. After testing further examples, the instrument was finally refined to the one described in Section 4.

The instrument was then used by six reviewers (two female, four male; age 37–48) to evaluate five example programs. All reviewers are experienced computer science lecturers in object-oriented programming, most of them at the introductory level. The results of the evaluation are presented in Section 5.

To evaluate the instrument, we decided to focus on early examples. We chose examples of different levels of quality and complexity covering the following aspects; the very first example of a textbook, the first exemplification of developing/writing a (user-defined) class, the first application involving at least two interacting classes and a non-trivial (but still simple) example of using inheritance. Table 1 summarizes the features of our five examples E1–E5.

4. EVALUATION INSTRUMENT

Inspiration for the evaluation instrument was drawn from the checklist-based evaluation by the Benchmarks for Science Literacy project [1] by defining a set of specific, well-defined criteria that can be evaluated on a uniform scale. All criteria should be based on accepted programming principles, guidelines, and rules; educational research; and the groups' collective teaching experience. The resulting set of 11 criteria was grouped into three independent aspects of quality; *technical quality* (three items), *object-oriented quality* (two items) and *didactic quality* (six items).

Table 1: Categorization of example programs.

	First example	First user-defined class	Several classes	Inheritance
E1	—	—	X	—
E2	X	—	—	partly
E3	—	X	—	—
E4	—	X	partly	—
E5	—	—	X	X

Technical quality (T1–T3). The criteria in this category focus on technical aspects of example programs that are independent of the programming paradigm. Examples should be syntactically and semantically correct, written in a consistent style and follow accepted programming principles, guidelines, and rules (see Table 2).

Table 2: Checklist items for technical quality.

T1	**Problem versus implementation.** The code is appropriate for the purpose/problem (note that the solution need not be OO, if the purpose/problem does not suggest it).
T2	**Content.** The code is bug-free and follows general coding guidelines and rules. All semantic information is explicit. E.g., if preconditions and/or invariants are used, they must be stated explicitly; dependencies to other classes must be stated explicitly; objects are constructed in valid states; the code is flexible and without duplication.
T3	**Style.** The code is easy to read and written in a consistent style. E.g., well-defined intuitive identifiers; useful (strategic) comments only; consistent naming and indentation.

Object-oriented quality (O1–O2). The criteria in this category address technical aspects that are specific for the object-oriented paradigm, i.e., how far an example can be considered a role model of an object-oriented program. In contrast to technical quality, the principles, guidelines and rules covered here are specific for the object-oriented paradigm (see Table 3).

Table 3: Checklist items for object-oriented quality.

O1	**Modeling.** The example emphasizes OO modeling. E.g., emphasizes the notion of OO programs as collections of communicating objects (i.e., objects sending messages to each other); models suitable units of abstraction/decomposition with well-defined responsibilities on all levels (package, class, method).
O2	**Style.** The code adheres to accepted OO design principles. E.g., applies proper encapsulation and information hiding; adheres to the Law of Demeter (no inappropriate intimacy); avoids subclassing for parameterization; etc.

Didactical quality (D1–D6). The criteria in this category deal with instructional design, i.e., comprehensibility and alignment with general learning goals for introductory (object-oriented) programming (see Table 4).

To summarize, one could say that T1–T3 and O1–O2 assess the actual code of an example program and D1–D6 assess how it is presented to the learner. The categories complement each other; an example of high technical and object-oriented quality will not be very effective, if it cannot be understood by the average learner. However, such an

Table 4: Checklist items for didactic quality.

D1	**Sense of purpose.** Students can relate to the example's domain and computer programming seems a relevant approach to solve the problem. In contrast to, e.g., flat washers which are only relevant to engineers, if the concept or word is at all known to students outside the domain (or English-speaking countries).
D2	**Process.** An appropriate programming process is followed/described. I.e., the problem is stated explicitly, analyzed, a solution is designed, implemented and tested.
D3	**Breadth.** The example is focused on a small coherent set of new concepts/issues/topics. It is not overloaded with new "stuff" or things introduced "by the way". Students' attention must not be distracted by irrelevant details or auxiliary concepts/ideas; they must be able to get the point of the example and not miss "the forest for the trees". In contrast to, e.g., explaining JavaDoc in detail when the actual topic is introducing classes.
D4	**Detail.** The example is at a suitable level of abstraction for a student at the expected level and likely understandable by such a student (avoid irrelevant detail). In contrast to, e.g., when an example sets out to describe the concept of state of objects, but winds up detailing memory layout in the JVM).
D5	**Visuals.** The explanation is clear and supported by meaningful visuals. E.g., uses visuals to explain the differences between variables of primitive (built-in) types and object types. In contrast to, e.g., showing a generic UML diagram as an after-thought without relating to the actual example.
D6	**Prevent misconceptions.** The example illustrates (reinforces) fundamental OO concepts/issues. Precautions are taken to prevent students from overgeneralizing or drawing inappropriate conclusions. E.g., multiple instances of at least one class (to highlight the difference between classes and objects); not just "dumb" data-objects (with only setters and getters); show both primitive attributes and class-based attributes; methods with non-trivial behavior; dynamic object creation; etc.

example might still be a very valuable teaching resource, in case the educator using it finds better ways to explain it.

All ratings in the resulting checklist are on a Likert-type scale from 1 (strongly disagree) to 5 (strongly agree). Since all checklist items are formulated in the same (positive) way, 5 is always best. An example of a filled-in checklist can be found at `http://www.cs.umu.se/research/education/checklist_iticse08.pdf`.

5. RESULTS

The results presented here are based on the evaluation that was made in order to answer two questions:

- Can the instrument distinguish between "good" and "bad" examples?
- Do reviewers interpret the items of the instrument in the same way?

Figure 1 summarizes the results of this evaluation of five examples, E1–E5 (see also Table 1). As can be seen, only one example (E1) is consistently rated very high across all three quality categories. Low average ratings have almost always a relatively high standard deviation (i.e., disagreement between reviewers).

Besides the overall high rating of E1, there are several other noteworthy observations. The overall technical quality of the reviewed example programs is very high, except for E5 which did not correctly implement its stated requirements. The section on object-oriented quality has the largest variation. It should, however, be noted that E2 is a "Hello World"-type example which cannot be expected to achieve high ratings in this category. Given that we used examples from quite popular textbooks, the overall ratings for didactic quality and the ratings of E3–E5 on object-oriented quality were surprisingly low.

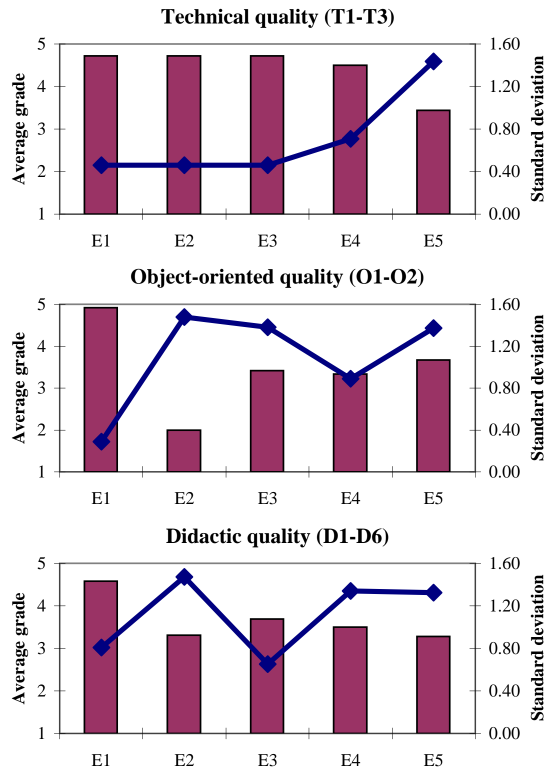

Figure 1: Average grade (bars) and standard deviation (line) for evaluation of five examples. Results are shown by item category (technical, object-oriented, and didactic quality).

Figure 2 shows the overall distribution of ratings for each of the six reviewers, R1–R6. It can be noted that the reviewers utilize the rating scale differently. Reviewer R5, for example, used the best grade (5) only half as much as the average (21.8% compared to 43% for all reviewers together). Reviewer R6, on the other hand, did not use a single 1. However, except for reviewer R5, the distributions of ratings are quite similar (in total the usage of rating 1 was only 7.3%).

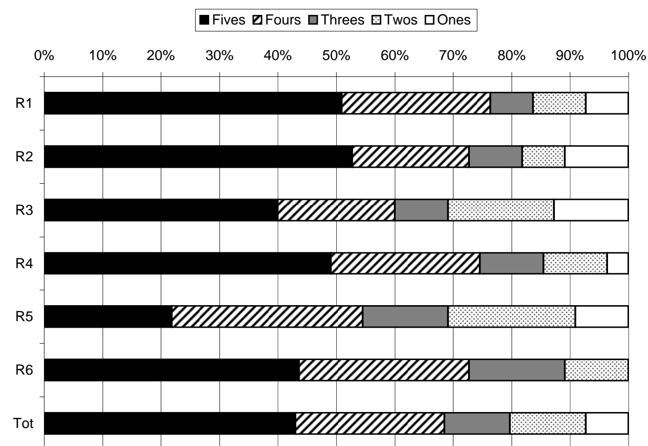

Figure 2: Distribution of ratings between reviewers.

It seems that teaching experience somewhat influences the grading. One reviewer, R5, has exclusively taught advanced programming courses for the last couple of years, and grades given by R5 tend to be slightly lower on average.

To summarize, it is evident that the instrument distinguishes between examples. Furthermore, the example with the overall highest ratings, E1, is also considered to be a "good" example by the authors. However, when looking at Figure 2, it is evident that there are differences in ratings among the reviewers. These differences will be discussed further in later sections.

6. DISCUSSION

The purpose of the presented instrument has been to replace the intuitive "I know it, when I see it"-knowledge with a more objective measurement. So what conclusions can we make from the results? What about the choice of items? How well does the instrument reflect the experienced teachers opinion of the example?

6.1 Quality categories

Overall, we find that the scoring does rank the examples as we would have done based on an informal discussions and ranking. Furthermore, the reviewers find the three categories natural and covering the critical issues of examples.

Technical quality (T1–T3). Assuming that textbook authors have developed and tested their examples carefully, one would in general expect the technical grades to be very high. This is also shown in the ratings. The only exception is E5, that contains a defect and the resulting program does not fulfill the requirements. This is well captured by the items.

Object-oriented quality (O1–O2). In general, the object-oriented characteristics of examples seems to be captured by this quality. E1 received the highest rating by the instrument and was agreed upon to be the best example in this respect when discussed in the group. E2 is the first example given in that textbook and is not really focused on object-oriented techniques which is reflected by its low OO-quality score.

Didactic quality (D1–D6). When comparing the results in this category, one example is rated high. The others at approximately the same level, although with different, and in most cases high, standard deviations. When investigating the ratings of the individual items, we noticed that the disagreement among the reviewers were high in many of the items in all examples. It seems that the group of reviewers do not share a common understanding of the meaning of the items and how they should be rated.

6.2 Items

Not doing the initial study, (see Section 3), thorough enough, we underestimated the semantical issues concerning the items. It was implicitly assumed that all reviewers had the same interpretation of each single item. During the evaluation described in Section 5 it became evident that the rating still was difficult in some cases.

O1 vs. O2 Examples not considered to be object-oriented caused discussion on how to rate O2 in relation to O1. Since O2 was meant to be independent of O1, some reviewers gave high O2-grades despite a low rating of O1. The intention is that lack of object-orientedness should result in low ratings. Therefore, it is necessary to agree on (and describe) the intended use of O1 and O2.

Maybe O1 and O2 could be replaced by a conditional assessment with O1 as the overall rating for OO or not and O2 as a more detailed assessment of the OO-characteristics. If an example is considered to be non-OO (e.g., in "Hello World"-type examples), O2 is not rated.

D3 vs. D4 It is often difficult to decide where to put the critique on breadth vs. detail of an example (and/or its explanation). It is clear that D3 and D4 are related, and it might be cases where an example has been penalized twice.

D5 There was confusion on how to rate a complete lack of visuals compared to "bad" visuals. This discussion can be extended to other items as well, e.g., O1, O2, and D2. It is has been problematic for some reviewers not to be able to separate "violating" from "not addressing".

D6 When investigating the items once more, we started to believe that the item D6 maybe should be regarded as "object-oriented quality" rather than "didactic quality". Making this change resulted, however, only in minor changes of the results as compared to Figure 1.

Granularity and impact Other aspects concerning the items are the granularity and the impact of each item in a compound rating. If a total score were to be used, the weight of individual items must be decided. It is clearly not as important that an example is supported by visuals (D5) as that it is prevents misconceptions (D6). Moreover, the number of items in each category will in fact lead to a implicit weighing of categories.

6.3 Documentation

Written documentation is important in all communities to establish a common ground. Using documentation will ensure that independent reviewers build their evaluations on the same definitions and have a common understanding of the implications of the rating scale. Furthermore, documentation of the reviews including not only the rating in numbers but also the comments from the reviewer makes it possible to investigate "out-liers" further.

Some examples One topic of discussion was when to rate an item for an example as 1 and when to rate it as 5, i.e., to get a common understanding of the extremes for each item. During these discussions, examples were often used to illustrate these extremes. If this instrument is to be used in a community, we strongly recommend that a written instruction, containing such examples, should be supplied with the instrument. In the rest of this section, we will present some examples that were used in our work. As a future work, a written instruction will be developed.

A class with four primitive attributes where the constructor initializes only two of them will give a low T2 rating since the attributes are dealt with in two different ways. Another example is the same code fragment appearing several times without being refactored into a method.

O1 is rated low, e.g., in the following cases: classes with too many and/or unrelated responsibilities; procedural programs that are just casted into classes; over-use of class methods or attributes; or examples not showing multiple instances of a class when possible.

An example should get low ratings in D1 if its domain is

too complex or "off topic" to be understandable or interesting for the average computer science student. This will happen even if the example has high ratings in technical quality and object-oriented quality.

When a large amount of new concepts and issues are introduced, D3 will get low ratings. One example is when the introduction of user-defined classes is interleaved with explanations of the syntax and purpose of JavaDoc.

If an example class only contains 'set'- and 'get'-methods but no meaningful behavior, students may equate classes with plain records and this should result in low D6 ratings. Another example is if all classes are instantiated only once; then the distinction between object and class easily becomes blurred.

7. SUMMARY AND CONCLUSIONS

In this paper, we have described the design and test of a suggested instrument for evaluating introductory examples. The purpose of the instrument is first and foremost to be used for evaluating object-oriented examples for educational purposes. Preferably it should be possible to use the instrument both to compare individual examples and to evaluate a larger set of examples, e.g., textbooks.

The result of the test shows that the instrument is working as predicted. An individual can use the instrument to make comparisons among examples. However, the instrument must be refined before being used by a community of reviewers. The differences among ratings of some of the items are too large to make the results reliable, since we believe that part of it is due to the interpretation of the items.

The refinement has already been initiated and consists of rewriting the items as well as complementing the instrument with instructions and examples of how to perform the evaluation, as indicated in Section 6.

8. REFERENCES

[1] AAAS. Benchmarks for science literacy, a tool for curriculum reform, 1989. `http://www.project2061.org/publications/bsl/default.htm`, last visited 2007-12-07.

[2] L. Böszörményi. Why Java is not my favorite first-course language. *Software-Concepts & Tools*, 19(3):141–145, 1998.

[3] L. Briand, C. Bunse, and J. Daly. A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. *IEEE Transactions on Software Engineering*, 27(6):513–530, 2001.

[4] M. Clancey. Misconceptions and attitudes that infere with learning to program. In S. Fincher and M. Petre, editors, *Computer Science Education Research*, pages 85–100. Taylor & Francis, Lisse, The Netherlands, 2004.

[5] R. Clark, F. Nguyen, and J. Sweller. *Efficiency in Learning, Evidence-Based Guidelines to Manage Cognitive Load*. Wiley & Sons, San Francisco, CA, USA, 2006.

[6] M. H. Dodani. Hello World! goodbye skills! *Journal of Object Technology*, 2(1):23–28, 2003.

[7] M. Guzdial. Centralized mindset: A student problem with object-oriented programming. In *Proceedings of the 26th Technical Symposium on Computer Science Education*, pages 182–185, 1995.

[8] S. Holland, R. Griffiths, and M. Woodman. Avoiding object misconceptions. In *Proceedings of the 28th Technical Symposium on Computer Science Education*, pages 131–134, 1997.

[9] E. Lahtinen, K. Ala-Mutka, and H. Järvinen. A study of the difficulties of novice programmers. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 14–18, 2005.

[10] Liz, Bills, T. Dreyfus, J. Mason, P. Tsamir, A. Watson, and O. Zaslavsky. Exemplification in mathematics education. In *Proceedings of the 30th Conference of the International Group for the Psychology of Mathematics Education, Vol. 1*, pages 126–154, 2006.

[11] K. Malan and K. Halland. Examples that can do harm in learning programming. In *Companion to the 19th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 83–87, 2004.

[12] J. Mason and D. Pimm. Generic Examples: Seeing the General in the Particular. *Educational Studies in Mathematics*, 15(3):277–289, 1984.

[13] P. Oman and C. Cook. Typographic style is more than cosmetic. *Communications of the ACM*, 33(5):506–520, 1990.

[14] N. Ourosoff. Primitive types in Java considered harmful. *Communications of the ACM*, 45(8):105–106, 2002.

[15] P. Reimann and T. J. Schult. Turning examples into cases: Acquiring knowledge structures for analogical problem solving. *Educational Psychologist*, 31(2):123–132, 1996.

[16] K. Sanders and L. Thomas. Checklists for grading object-oriented cs1 programs: Concepts and misconceptions. In *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 166–170, 2007.

[17] A. Takang, P. Grubb, and R. Macredie. The effects of comments and identifier names on program comprehensibility: an experimental investigation. *Journal of Programming Languages*, 4(143):167, 1996.

[18] K. VanLehn. Cognitive skill acquisition. *Annual Review of Psychology*, 47:513–539, 1996.

[19] R. Westfall. 'Hello, World' considered harmful. *Communications of the ACM*, 44(10):129–130, 2001.

[20] C.-C. Wu, J. M.-C. Lin, and K.-Y. Lin. A content analysis of programming examples in high school computer textbooks in taiwan. *Journal of Computers in Mathematics and Science Teaching*, 18(3):225–244, 1999.

[21] R. Zazkis, P. Liljedahl, and E. J. Chernoff. The role of examples in forming and refuting generalizations. *ZDM Mathematics Education*, 40:131–141, 2008.