PROGRAMMING IS ALSO DEBUGGING

Rudolf Pecinovský
ICZ a.s., Na Hřebenech II 1718/10, 140 00 Prague 4,
University of Economics, Prague, Faculty of Informatics and Statistics, Department of Information Technologies
rudolf@pecinovsky.cz

ABSTRACT:

When we talk about teaching programming, we mostly discuss only how to teach the art of coding. Possibly we accompany it with a discussion on teaching the art of analyzing and designing. During these discussions we do not realize that students (and professional programmers as well) do not spend most time by coding but by debugging. However, the standard courses usually do not deal with the art of proper debugging. The paper addresses this handicap and is searching for the way to incorporate this topic into our courses.

KEYWORDS:

Teaching of programming, debugging.

1 INTRODUCTION

Most of so called programming textbooks deals mainly with the syntax of the language being taught and with explaining the contents of the most important libraries. Sometimes we meet a textbook that also tries to teach how to design programs. However, the art of finding errors and fixing them is discussed very rarely, and if so, the authors are touching this subject (e.g. [2], [5], [6], [8], [9]) only on a few pages. Because of the limited space they can only present the basic rules and principles mostly without illustrative examples allowing students to exercise their understanding similarly to the other explained subjects.

Omitting the explanation of debugging may mislead the readers making them to think that programmers simply write the program and the program works. When then the readers start to write own programs, they become disillusioned. Their programs are not as perfect as they are supposed to be. In addition they get frustrated because of their inability to find and fix errors. It is often strengthened by the teacher, who is under pressure of time allowing him/her to show only, where the error should be corrected, and at the same time preventing him from explaining the issue in more depth. In the worth case he/she only quickly corrects the error so as to keep explaining the subject. As a consequence the student is not capable of repairing a similar error next time.

Many teachers claim that they teach students how to find and repair bugs. However, taking a closer look at their teaching, we reveal that they teach only how to write tests for revealing the expectable errors. The teacher suggests what should be tested and how it should be tested. In the standard courses and textbooks you do not find any explanation how to react when the program does not work. The goal of this article is to suggest the way to change it.

In the following sections first I mention the typical situation we meet daily, when students writing a program discover an error. (Many teachers know them and they surely are able to add other examples.) Then I will discuss what we need to be able teach designing programs together with their debugging.

2 STUDENT CATEGORIES

Unfortunately, in every class there are students who are not able to find the source of the error and grope for explaining the behavior of their program. Before we start to ponder on teaching

debugging, we should remind ourselves that students are not a homogenous group. We can split them into several categories:

- The students, who want to specialize themselves in a different profession (mostly management) and who consider the mandatory subject of programming an inevitable evil.
- The students, who are interested in programming and who want to learn it. We split them again into two categories:
  - The beginners who have never been programming before.
  - The students, who know to program and who bring some inevitable wrong habits with their experience, too.

Each category needs a specific approach. Let's have a look at some typical situations and reactions we can meet across all categories (maybe without the most experienced students).

2.1 "There is an error in your program"

One of the most common cases is the situation, where the student announces that there is an error in the teacher's program. The student wrote the program exactly according to the teacher's instructions (often he/she copied it from the board) and the computer reports an error. Students are therefore sure that the error must be on the teacher's side. When you ask them what an error occurred, they answer that they do not know – simply an error. This situation appears most often especially in the beginning lessons.

All these students are absolutely sure that their program is the letter-to-letter copy of the teacher's one – no mistakes, no typos. These students are not surprised that the program works on the other computers and does not work on their one. They are pretty sure that the error comes from outside. It is astonishing that almost no student tries to read the error message that often clearly tells what the problem is. Maybe, in the first lessons the beginners do not understand these messages, but they should at least make an attempt. When the message will appear the next time, they can build on its previous experience.

The errors appearing after copying down some example programs are mostly syntactic errors, which professional programmers do not take for errors, but typos, because they are immediately and simply reparable. In such cases the compiler messages are mostly sufficiently clear for fixing them also by beginners.

2.2 "It reports an error"

A similar situation occurs, when an error appears in a program written by the students themselves. The students are certain that the error cannot be in their small program and that it therefore should be in the incomprehensible "monster" compiling and running their program.

It's true that the error message from the thrown exception is often much less comprehensible than the error messages from the compiler announcing a syntax error. Nevertheless, even the exception messages can often be analyzed simply to obtain sufficient information for detecting the source of problems.

In such cases the most important is the acceptance of the rule: "The more beginning the programmer is, the sooner he/she starts searching the error in the compiler, operating system, colleagues, the teacher or in anything else outside its code". The experienced programmer, who runs exotic parts of libraries and frameworks, meets such an error once in 10 years in average. However, the programs in the beginning courses use the parts of the environment, which have been tested through and through many times and so we can trust that they do not hide some peculiar error waiting for us.

2.3 "Computer does something else than the program prescribes"

Another typical problem happen when the program's behavior differs from the behavior prescribed in the source code. For instance, the students show you the parts, where the program assigns the right value to the right variable and in the next step the computer says that the value was not assigned. They complain that the computer runs their program incorrectly, what means the computer is doing something different from what the program prescribes.

It is only a variation of the behavior described above. I am always asking: "Why do you think that the computer is doing something incorrectly? Are you sure, that the computer runs through this part of the program?" I must repeatedly stress: "Do not trust anything you see in the code without verifying that the computer runs through it. Believe only the behavior the computer shows you. In such cases you need first to verify that the program really runs through the problematic part of the code."

2.4 Summary

The situations described above belong to the primitive ones. Their solutions are often simple and do not require some deep knowledge of debugging. Mostly, it is enough when students adopt some internal discipline: start to read the obtained messages and to search for the source of errors inside their code only. The genuine debugging starts when the simple solutions are unacceptable and the students are spending hours by looking in their code, by comparing their programs with the demonstration ones and by useless investigation where the error can be this time.

3 HOW TO TEACH DEBUGGING

I make a step aside. Most mathematics assignments on basic and grammar school's level can be solved by simple substituting the right values to the appropriate formula. The only exceptions are geometrical constructional tasks and factoring algebraic expressions. There are no general rules for solving these tasks. The students should learn these subjects by solving sufficient amount of examples.

Teaching programming is similar in that there are many problems, which we can solve by simple "substituting values into a formula":

- If the program is supposed to choose one of the several possible continuations, we define the appropriate condition and use a conditional statement.
- If we are supposed to do something with all elements in an array, we use a loop.
- If we need to repeat the same segment of the code at several places, we define a subprogram and call it from these places.
- And so on, and so on…

Beside these tasks there is a big amount of tasks, whose solution cannot be described in a general way. Finding and fixing bugs (debugging) belongs to this category too. Although, there are some rules for behaving in some typical situations (e.g. when the `NullPointerException` is thrown), however even these rules do not guide to the goal because they only indicate the right path.

3.1 Literature

As I mentioned in the beginning, almost any book teaching programming and especially books trying to introduce the beginners to the world of programming do not deal seriously with debugging. If the authors mention debugging in their texts, they explain only general rules such as "for each class define its test class", "use logs", "use the `assert` statement" and so on.

There are several books concentrating on debugging programs under development (e.g. [1], [3], [4], [7], [14]), however, these books are not suited for the beginners. The beginners need another kind of books.

3.2 Collections of examples

When we learned mathematics we used textbooks with a collection of examples helping us to learn how to solve mathematical problems. There are some programming textbooks based on solutions of examples (e.g. [2], [8], [9]). There are also some textbooks with collection of programming exercises. However, I do not know any published collection of programs for teaching debugging.

Such an attempt to prepare a mini collection of programs with common bugs and to show how to find and fix these bugs is work [13]. The work is supplemented by a set of animated explanations showing the debugging process using the debugger integrated in BlueJ IDE. We can reproach its author for many inaccuracies and some immaturity, however, this work shows the direction.

4 SUMMARY

The paper explains why teaching debugging should be an integral and equipollent part of teaching programming, despite it is still being overlooked. It summarizes the present status and emphasizes the insufficient concentration on this field in comparison with the time spent by this effort. It reminds several most frequent problems which we encounter at students and it suggests a solution. Finally, it introduces a student work preparing a foundation of the future collection of solved examples showing how to find and fix bugs in programs.

REFERENCES

[1] BUTCHER Paul: *Debug It!: Find, Repair, and Prevent Bugs in Your Code.* Pragmatic Bookshelf 2009. ISBN 978-1-934356-28-9

[2] DEITEL H. M., DEITEL P. J.: *Java How to Program, 7th Edition.* Prentice Hall 2007, ISBN 978-0-132-22220-4

[3] DiMarzio J.F.: *The Debugger's Handbook.* Auerbach Publications 2006. ISBN 978-0-8493-8034-1.

[4] GRÖTKER Thorsten, HOLTMANN Ulrich, KEDING Holger, WLOKA Markus: *The Developer's Guide to Debugging.* Springer 2010. ISBN 978-90-481-7387-7.

[5] HORSTMAN C. S.: *Big Java (3rd Edition)*. John Wiley and Sons 2007. ISBN: 978-0-470-10554-2.

[6] LIANG, Y. Daniel: *Introduction to Java programming: comprehensive version.* Pearson Education 2010. ISBN 978-0-13-213080-6

[7] METZGER Robert Charles: *Debugging by Thinking: A Multidisciplinary Approach.* Digital Press 2003. ISBN 978-1-5555-8307-1.

[8] PECINOVSKÝ Rudolf: *OOP – Naučte se myslet a programovat objektově.* Computer Press 2010. ISBN 978-80-251-2126-9.

[9] PECINOVSKÝ Rudolf: *Myslíme objektově v jazyku Java – kompletní učebnice pro začátečníky, 2. aktualizované a rozšířené vydání.* Grada 2008. ISBN 978 247 80 2653 3.

[10] PECINOVSKÝ Rudolf: *Výuka OOP žáků základních a středních škol*. Sborník konference Objekty 2003, Ostrava, ISBN 80-248-0274-0.

[11] PECINOVSKÝ Rudolf: *Proč a jak učit OOP žáky základních a středních škol*. Žilinská didaktická konferencia, 2004, Žilina.

[12] PECINOVSKÝ Rudolf: *Výuka programování pro praxi*. Sborník konference Informatika XXI 2008, Lázně Luhačovice.

[13] ZÁVĚRKA Jakub: *Alternativní výukové materiály pro vstupní kurzy programování* – Bakalářská práce, VŠE 2010.

[14] ZELLER Andreas: *Why Programs Fail – A Guide to Systematic Debugging.* Morgan Kaufman 2009. ISBN 978-0-12-374515-6.