MODULA-2 VERSUS C++ AS A FIRST PROGRAMMING LANGUAGE - SOME EMPIRICAL RESULTS

Martin Hitz
Institut für Angewandte Informatik und Informationssysteme
University of Vienna
Rathausstrasse 19/4, A-1010 Vienna, Austria
hitz@ifs.univie.ac.at

Marcus Hudec
Institut für Statistik, Operations Research und Computerverfahren
University of Vienna
Universitätsstrasse 5, A-1010 Vienna, Austria
hudec@ifs.univie.ac.at

ABSTRACT

The success of an experiment of using C++ as a first programming language for students of a specific type of computer science is presented.

The paper motivates the shift from Modula-2 to C++ in the curriculum, shortly describes the course and discusses the statistical evaluation of the results of the last Modula-2 course and the first C++ course, respectively.

The main findings of the study are the fact that in contrast to most expectations, the shift from a typical "educational" language to a much "dirtier" language had no significant effect to the performance of the students taking the course.

INTRODUCTION

In October 1992, the Department of Applied Computer Science and Information Systems at the University of Vienna began using C++ as language for its introductory course in programming. This course is primarily dedicated to beginners of *Wirtschaftsinformatik*, a hybrid study offered at the faculty of Social and Economic Sciences, combining courses in computer science and economy. Due to time restrictions, the *Wirtschaftsinformatik* curriculum features only this single regular programming course. Other languages are dealt with only in advanced computer science courses such as comparing programming languages, object-oriented programming or the like, where an in-depth-treatment of these languages is out of scope.

This change is the last in an evolutionary sequence of language replacements since the study was first offered in 1974: We began with Fortran, switched to PL/1, followed by Pascal in 1977 and taught Modula-2 from 1987 to 1991. All those moves where motivated by pure didactic considerations, as the specific language in itself was not considered the goal of the course, but rather a vehicle to teach the abstract principles of programming. While this aim was apparently achieved in the past (the Pascal and Modula-2 era), the students were somewhat unsatisfied when they realized that they could not really use the language once they started to do some practical work, either outside the university (where only a minority of firms employed Pascal or Modula-2) or in subsequent computer science courses, where C has been predominant in the last years. Instructors, on the other hand, were also frustrated with the fact that they had to cope with the usual "traps & pitfalls"-kind of problems a self-taught C programmer is likely to fall into before they could turn to the specific subjects of their courses. In the recent past, moving to C as a first language was sometimes discussed, but was always rejected because of its obvious weaknesses from a didactics point of view. Specifically, its willingness to silently accept many nonsensical programs was deemed to be the major problem for beginners.

With the advent of good quality, PC-based C++ compilers, however, another round of discussion on a curriculum revision started. While some conservative arguments in favor of a dedicated teaching language persisted, a marginal majority within the department was convinced that it was worth while trying C++. The arguments pro C++ can be summarized as follows:

- The type system is strong enough to eliminate a major part of the problems beginners were expected to fall into when programming in C.
- The transition from C++ back to C, which is still going to be used for a while in industry as well as in some computer science courses, was considered much easier than the transition from Modula-2 to C.
- In the recent past, the department has been offering several advanced courses dealing with the object oriented paradigm, notably the object oriented software engineering course. For these courses, knowledge of C++ was extremely useful.
- From a "marketing" point of view, we realized that, graduates knowing C++ are more and more welcomed in industry.

It should be noted, though, that due to time limitations, object oriented programming was explicitly excluded from this introductory course. Thus, the subject selected was C++ without inheritance. However, *object based* programming was treated as a major idiom, replacing the notion of a Modula-2 *module* by a C++ *class*. With respect to the empirical comparison presented in this paper, the restriction of C++ features is insignificant, as those features left out are not available in Modula-2 anyway.

In the remainder of this article, we describe the content of the course in some more detail (Section 2) and present the experiences we gained comparing the results collected during the first C++ course with the ones of the last Modula-2 course (Section 3).

COURSE CONTENTS

The basic assumption on the students attending the course is that they do not have any programming experience yet. This assumption usually holds for roughly 75% of about 150 students enrolled, although the number is decreasing steadily because more and more high-schools start offering some programming education.

To satisfy this target group, we have to start teaching programming from scratch, while sacrificing higher concepts such as inheritance and polymorphism. The subset of C++ taught roughly matches the language features of Modula-2 and may alternatively be described as "a safer C with classes and templates". Genuine object orientation is offered in more advanced courses, which are hopefully going to take advantage of the new basic education in the first year.

The course consists of 13 weeks with 135 minutes of lecture each, followed by two hours of laboratory exercises in groups of approximately 10 students supervised by a teaching assistant, in which 3 assignments have to be completed using Borland C++ 3.0 in a DOS environment.

The course schedule is organized roughly as follows:

Lecture	Contents
1, 2	Basic language independent concepts: Programming, algorithms and their design (step-wise refinement), programming language, data objects, statements; compiling, linking, executing
3	Plunging into C++: simple data types, variables, expressions, assignment, `if`, functions and "procedures", simple stream i/o
4	Functions and "procedures", addresses, pointers, parameter passing (by value and by reference using pointers)
5	Loops, `switch`, advanced expressions
6	Advanced scalar data types, references, type conversions, static typing
7	Arrays and their relation to pointers, C-structs, orthogonality of concepts
8	Scopes, lifetime of data objects (`auto`, `static`), definition vs. declaration
9	Classes as user defined data types, information hiding, abstract data type, function and operator overloading, default arguments
10	Constructors, destructors, `static`, `inline`, `friend`
11	dynamic data structures (dynamic arrays, linked lists)
12	templates
13	input / output, files

Each student is randomly assigned one out of 10 to 12 problems per assignment category. All assignments are given in advance at the beginning of the course, with the approximately equally distributed deadlines. Students are encouraged to hand in their results as soon as they are done in order to save time for the following assignment.

Assignment I is a simple first program of about 100 source lines that does not need any sophisticated data structure, e.g. solving a system of three linear equations in three variables. However, procedural abstraction is strongly encouraged already at the very beginning. The over-proportional time allotted to Assignment I allows for getting used to the lab and to the PC environment, playing around with some mini-examples as presented in the lectures etc.

Assignment II typically needs arrays, pointers and simple records. Examples are vector and matrix arithmetic or simple hashing schemes.

Assignment III involves at least one user defined data type, which is to be designed as a reusable part in advance, tested in a test bed and finally used in the given application. It is also expected to be generalized to a template, once it is able to play its primary role within the application.

Assignments are marked by the teaching assistants. After Assignment II, a first oral examination takes place, which is mainly based on a discussion of the solutions of the first two assignments, thus offering an opportunity to clarify any misunderstandings that might have occurred. At the end of the course, a final oral exam takes place, emphasizing on the more theoretical concepts of the language and on programming in general. The grading policy is to judge primarily between "pass" and "fail" and to bias the distribution of positive grades (1=very good to 4=sufficient) towards the lower range in order to give an incentive award.

As accompanying material, the students are given handouts tailored to the lecture. As supplemental textbooks, we recommend Lippman's C++ Primer [2] for the beginners and Stroustrup's book [4] as well as a German textbook [1] for those who already have some programming experience.

COMPARISON OF RESULTS

As the decision in favor of C++ was not reached unanimously, we tried to prepare a comparison of our first results to those of the preceding Modula-2 course. All grading data of the Modula-2 course were stored and the set of assignments was kept the same, although the language dependent wording was changed accordingly to C++[1].

The resulting data set consists of 172 and 140 cases from the Modula-2 course and from the C++ course, respectively. For each case, we basically recorded the teaching assistant's scoring of the three assignment. Each score is made up of four components evaluating different aspects of the solution:

- program structure
- algorithmic correctness
- quality of the user interface
- quality of the documentation

While these scores measure the quality of the student's *products,* another dependent variable, the result of the final exam, is linked to his/her *understanding* of the underlying concepts. Finally, we may also observe the dropout behavior in each course.

As independent variables, we consider the course taken (Modula-2 or C++) and the three problems assigned.

The data analysis is based on descriptive statistical methods complemented by both, parametric (Student's t test) and nonparametric test procedures (Wilcoxon's rank-sum test) for the comparison of location parameters. Practical calculations have been performed using S-Plus [4].

The results of all analyses are consistent with each other and tell us that *on the whole no significant change of the performance of the two groups of students can be observed.* Some minor differences are discussed below.

a) Dropout

The total dropout rates are approximately identical (39.5% for Modula-2, 37.9% for C++). However, a significant difference could be detected in the dropout *behavior*: In Modula-2, only 7.0% of the beginners quit before completing Assignment I, while 35.0% of those having completed Assignment I did not pass the course for various reasons (i.e., dropped out after Assignment I). In C++, 14.3% (twice as many!) quit immediately, and only 27.0% dropped out during the second phase.

1. In the most recent course, assignments have been thoroughly adapted to the new language, yielding incommensurable results not included in this study.

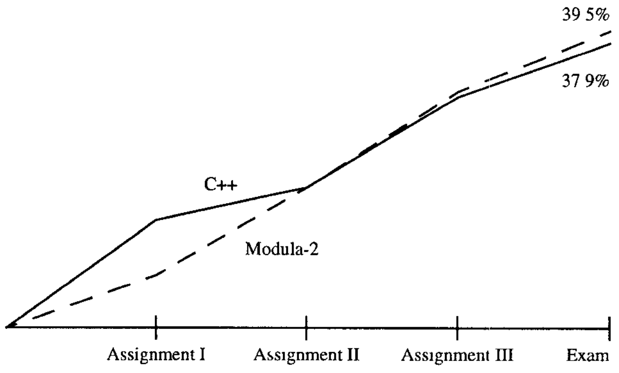

Figure 1: Dropout rates

Our interpretation of this fact is that at the first glance, C++ looks much more complicated than Modula-2, immediately discouraging many students, while Modula-2 is looking pretty, but revealing the complexity of programming *per se* only at a later stage.

A closer look reveals that the lower dropout rate in the beginning of the Modula-2 course had increased up to the corresponding level of the C++ course in the second phase, when usage of pointers is demanded.

b) Assignment scores

Assignment scores of both samples are represented by boxplots, which provide an informative explanatory tool for the graphical comparison of the distribution of samples:

The white line in the middle marks the position of the median, the lower and upper part of the box show the first and third quartile. Thus, the box represents the limits of the middle half of the data. Extreme points (i.e., observations outside of a standard span from the quartiles) are highlighted. The gray shaded double cone around the median indicates a confidence interval for the location on a 5% significance level.

The mean score over all assignments and all four grading dimensions (Figures 2-6) was slightly inferior in C++, although this difference turned out to be statistically not significant. Investigating isolated score dimensions, one finds a significant decrease in the quality of the user interface (Figure 3. The p-values are 0.034 and 0.017 for t test and Wilcoxon rank sum test, respectively), whereas the differences with respect to program structure and logic are again insignificant.

After discussions with the teaching assistants involved, we tend to attribute the deterioration of the user interface to one or both of the following reasons:

- Detection of input errors (a major requirement of the user interface in our assignments) is, at least for

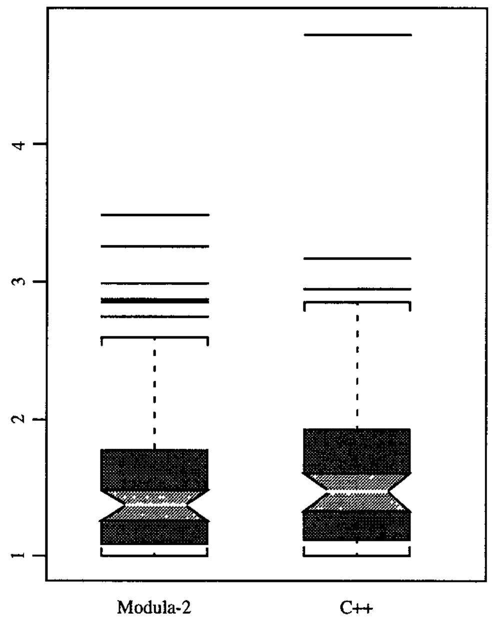

Figure 2: Total Average Scores

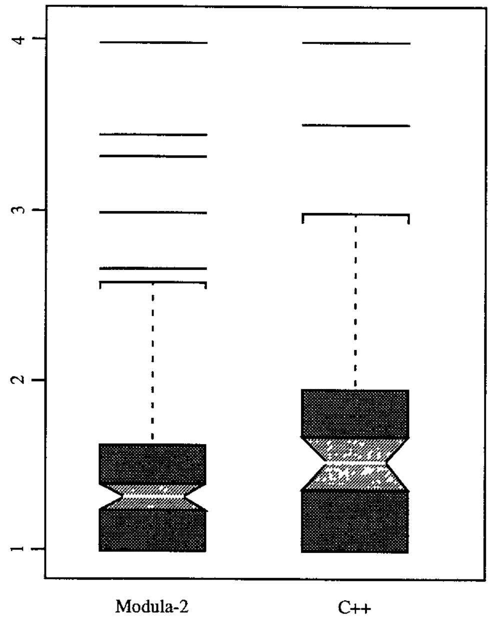

Figure 4: User Interface

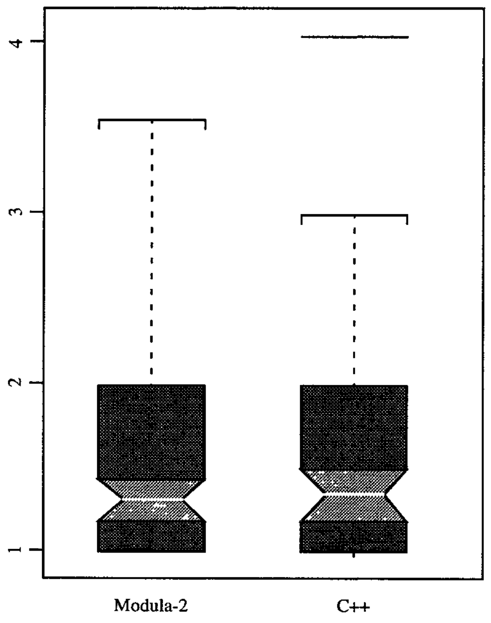

Figure 3: Program Structure

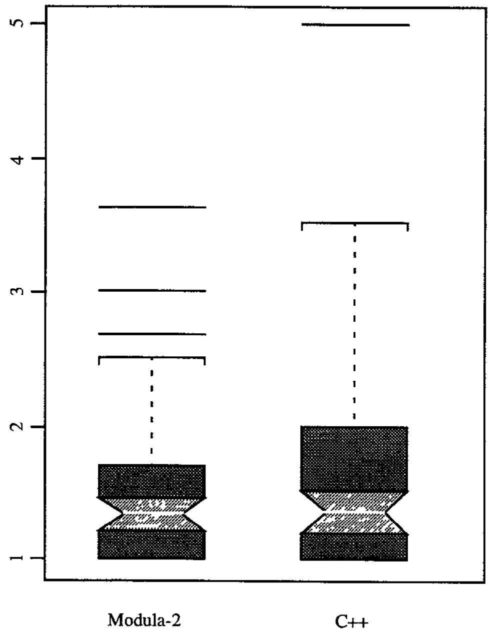

Figure 5: Algorithmic Correctnes

beginners, harder to accomplish with iostream (which we asked the students to use) than with the Modula-2 input output library.

- Solving the basic problem took longer in C++, so less time was spent to beautify the interface before handing in the assignment.

Comparing the results of the program evaluation of each of the three different assignments separately, it turns out that none of the dimensions showed significant differences within the group of Assignment I, where only primitive control and data structures are used. It thus seems that the somewhat awkward notation of C++ is not as much a problem to beginners as we had expected.

c) Oral exams

Interestingly, oral exams by five teachers (the same for both courses) yielded nearly exactly identical results (Figures 7 and 8), although we had expected the C++ course to be slightly worse in this variable, too. Of course, data can be biased due two one or both of the following reasons:

- Professors tended to be more "merciful" in the new situation of a first programming course taught in C++, a

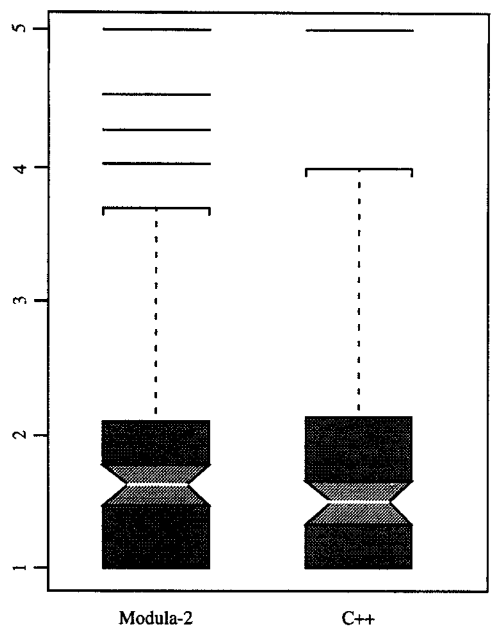

Figure 6: Documentation

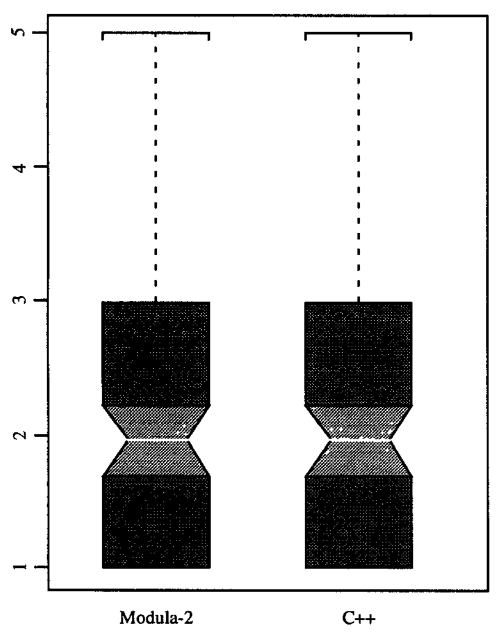

Figure 7: First Oral Exam

language they consider hard to learn.

- The rumor that C++ is rather hard to learn had the effect that students put more effort in the thorough study of the concepts of the language.

SUMMARY

From a statistical point of view, we must admit that the study presented is not very satisfying, because in most cases, we had to accept the null hypothesis ("mean scores do not differ"). It should be noted that these negative results do not imply a statistical proof of equivalence, since they

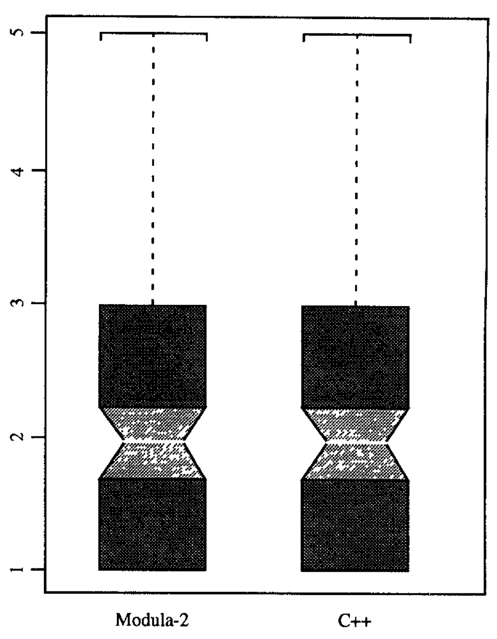

Figure 8: Final Oral Exam

suffer from the so-called "type 2 error".

However, from the viewpoint of a teacher of programming languages, the general experience and impression of the experiment of starting with C++ as a first language are rather encouraging, bolstering similar observations by Reid [3].

The practical progress of the course together with the empirical results as presented here convinced the former sceptics in our department. As a consequence, C++ is now established as the first programming language, at least for the near future.

Finally, it can be noted that the feedback of the participating students was so positive that many students from other disciplines were attracted to take part in the following year's course.

ACKNOWLEDGMENTS

The authors a indebted to Miss Minu Ayromlu for recording all data analyzed in this study.

REFERENCES

[1] Hitz, M. *C++ - Grundlagen und Programmierung.* Springer Verlag Wien - New York, 1992.

[2] Lippman, S. B. *C++ Primer (Second Edition).* Addison - Wesley 1992.

[3] Reid, R. J. C++ as a first programming language. *C++ Report* May 1993, pp. 41-44.

[4] *S-Plus for Windows.* Statistical Sciences Inc., Seattle, Washington, March 1993.

[5] Stroustrup, B. *The C++ Programming Language (Second Edition).* Addison - Wesley 1992.