An Empirical Study of Introductory Lectures in Informatics Based on Fundamental Concepts

Markus Schneider

Institut für Informatik
TU München
Boltzmannstr. 3
85748 Garching
markus.schneider@in.tum.de

Abstract: Before carrying out an empirical study in the area of didactics, one has to clarify the quantity to be measured. Often it is measured to which extent a student has acquired the basic concepts of the respective field. However, in Informatics this is a problem, since the question of the basic concepts is still discussed controversially.

This paper first analyses the introductory lectures in Informatics given from 2000 to 2003 at the Technische Universität München, extracts the learning targets and relates them to the most established fundamental concepts. The resulting sequence of concepts represents one dimension of the matrix of measurable quantities. The other dimension represents the complexity of the respective problem. By relating the elements of this two-dimensional taxonomy to the problems of the final exams of the considered lectures, one get the possibility to evaluate these lectures conceptually. This evaluation is done using the points the students achieved in the respective problems. Thereby, the analysis is performed both for the students as a whole and for male and female students separately.

1 Introduction

The first academic year at the university is perhaps the most crucial phase of the study of informatics. Here most of the students decide whether to abort or to continue the study until reaching an academic degree and the abortion rates in this phase is much higher than in the rest of the study. Therefore, for the first academic year a well-founded didactical concept is necessary not to lose motivated and engaged students yet at the beginning of the study.

Analysing the learning success of three lectures “Introduction to Informatics I/II” the presented empirical study is a contribution therefore.

The three lectures were given at the Technische Universität München: Academic Year 2000/2001 (Brügge 2000/2001), Academic Year 2001/2002 (Broy 2001/2002), and Academic Year 2002/2003 (Knoll 2002/2003). The author of this paper has accompanied these lectures and organized the tutorials associated to the lectures. During the three years, the number of students decreases from about 950 beginners in 2000 to about 350 beginners in 2002. Since the failure rate per academic year is about 30%, the number of students attending the lectures two- or more times is high.

The methodical basis of the here presented study is on the one hand similar to the strategy of the PISA-study (e.g.: Kunter et al. 2002); the evaluated problems are classified by their complexity. On the other hand, a classification concerning the learning targets is used. The basis of this scheme is related to the fundamental concepts proposed by Schwill (1993). The resulting two-dimensional matrix of measurable quantities represents the core of the study.

2 The structure of the considered lectures

First, the considered lectures have to be analyzed concerning their contents. The contents define the learning targets and these targets will be related to the above-mentioned fundamental concepts.

The tables 1-3 present the raw structure of the here considered lectures and show the contents of the lectures “Introduction to Informatics I” and “Introduction to Informatics II” in the 3 academic years from 2000 to 2003. Thereby, the order of the subjects represents their temporal order.

Academic Year 2000/2001

Principle of the lecture: The students are confronted from the very first with motivating systems having moderate or high complexity. Thereby object modelling techniques act as a guideline and offer the possibility to decompose this system into small subsystems. Implementing these small subsystems, the students are confronted with the various program paradigms; but also with theoretical topics, like program verification, semantics of recursive functions etc..

Table 1: Academic Year 2000/2001

Lecture	Subject	Concepts	Program Language
Introduction to Informatics I	Object-oriented modelling of systems	Object, Class, Aggregation, Inheritance, Interface	Java
	Algebras	Abstract algebra, concrete algebra, signature, algorithm, text rewriting system	
	Boolean algebra	Boolean algebra	
	Term rewriting system	Term rewriting system, interpretation, correctness	

Lecture	Subject	Concepts	Program Language
	Functional programming	Functional modelling, recursion, conditional expression, correctness, semantics, fixed point theory	
	Imperative/ OO-programming	Assignments, loop, imperative Programming embedded in OO-techniques, array	
	recursive data structures	Sequence, tree, algorithms on recursive data structures	
	Object-oriented programming	Inheritance, abstract class, polymorphism	
Introduction to Informatics II	UML	Class diagram, sequence diagram, use cases	Java
	Software Engineering	Design-Pattern: strategy, adapter, composite-pattern	
	OCL	Detailed design, contracts, AVL-Tree	
	Predicate logic	Predicate logic	
	Program-Verification	Hoare-Calculus, correctness	
	Exceptions	Exception handling	
	Event-oriented programming	Model view controller, observer	
	Automata and formal languages	DFA, Chomsky hierarchy, pumping lemma	
	Machine-oriented programming	v. Neumann architecture, abstract machine-oriented language, basic control structures	

Academic Year 2001/2002

Principle of the lecture: In some aspects, the basic concept of this lecture is dual to the one of the academic year 2000/2001: Firstly, the student develops small and simple systems using Boolean algebra and text rewriting systems and avoiding technical questions. In the course of the year, the student implements system using first the functional paradigm and then the imperative or object-oriented program paradigm. Theoretical questions like verification or semantics are discussed in the context of the respective paradigm.

Table 2: Academic Year 2001/2002

Lecture	Subject	Concepts	Program Language
Introduction to Informatics I	Information and Representation	Interpretation of Information, Boolean Algebra, Interpretation of Terms, Sequence, Formal Language	Gofer
	Algorithms and Algebras	Algorithm, text rewriting algorithms, algebra, algebraic specification, term rewriting systems, important algebraic structures	
	Program Languages	BNF, syntax, semantics	
	Functional programming	Functional modelling, recursion, basic recursive algorithms on numbers and sequences, semantics, correctness	
	Imperative programming	Statement, loop, procedure, Hoare-Calculus, array, reference	Pascal
Introduction to Informatics II	Recursive data structures	Sequences, stack, tree, algorithm on recursive data structures	
	Object oriented programming	Class, object, inheritance, polymorphism	Java
	Coding and Information	Coding techniques, information theory, security	
	Combinatorial and sequential circuits	Normal forms of Boolean functions, arithmetical circuits, combinatorial circuits and DFA	
	Computer architecture and machine-oriented programming	v. Neumann architecture, abstract machine-oriented languages, basic control structures, techniques of addressing, recursive data structures	MI

Academic Year 2002/2003

Principle of the lecture: Again, the principle of this lecture is to pass from simple to complex systems: First, small systems are developed using declarative program languages, whereas complex and object-oriented systems are discussed at the end of the academic year. The design of this lecture has some remarkable features: First, the fundamental principles of the functional program paradigm are introduced, to use these programming techniques for a practical discussion of the topics, cryptography, algebras, automata, etc. Spiral like, the theoretical aspects of these topics are discussed in the second part of the lecture.

Table 3: Academic Year 2002/2003

Lecture	Subject	Concepts	Program Language
Introduction to Informatics I	Overview on functional programming	Functional modelling, recursion, pattern matching, recursive data structure: sequence, trees	OCaml
	Information theory	Information theory, entropy, coding-trees, Huffman coding	
	Cryptology	LZW-algorithm, CRC	
	Algebra	Algebras, text-rewriting system, term-rewriting system	
	Predicate logic	Predicate logic, logic programming, BNF	Prolog, OCaml
	Formal languages and Automata	Chomsky hierarchy, DFA, NFA, Pumping Lemma	
Introduction to Informatics II	Chomsky hierarchy and Automata	PDA, stack-oriented programming, Turing-machine	Postscript, Forth
	Correctness of Functional programming	Partial and total correctness, noetherian induction	OCaml
	Recursive data structures	Trees, AVL-Trees, Algorithm on recursive data structures, B-trees	
	Theory of functional programming	Fixpoints of functions and data structures, Lambda-calculus	
	Object-oriented modelling	Foundation of object modelling, UML	
	Imperative programming	Assignment, Loop, Arrays	
	Object-oriented programming	Class, object, inheritance, polymorphism, design-pattern	
	Correctness of imperative programming	Hoare-Calculus	

Lecture	Subject	Concepts	Program Language
	Machine-oriented programming	v. Neumann architecture, stack, abstract machine-oriented languages, basic control structures	MI

Common Contents of the three lectures

Whereas the order and the intensity of the respective topic vary from lecture to lecture, all three lectures have common topics:

- Abstract and concrete algebras
- Text- and term rewriting systems
- Functional programming in theory and practice; verification of functional programs; semantics of functional languages
- Imperative programming and its verification using Hoare-Calculus
- Object-oriented modelling using UML, and object-oriented programming
- V.Neumann architecture and machine-oriented programming
- Combinatorial and sequential circuits
- Formal languages, Chomsky-hierarchy and Automata
- Information theory and coding

3 Fundamental learning targets and fundamental concepts

To get the fundamental learning targets from the above-mentioned topics, these topics are structured following the fundamental concepts of Schwill.

Fundamental concepts of Informatics

The basis of Schwill's classification is the central task of Informatics: The process of software development. Analysing this process, he derivates the following fundamental concepts:

- Development of algorithms:
  - Design paradigms: Branch and Bound, Divide and Conquer,
  - Concepts of programming: Recursion, Iteration, Indeterminism,
  - Sequential/concurrent processes
  - Evaluation: Verification and complexity
- Structured partition:
  - Modularisation: Top down method, bottom up method, specification, abstract data types,
  - Hierarchical Structures: Trees, Compilation,
  - Detection of orthogonal structures
- Languages
  - Syntax
  - Semantics

The fundamental learning targets of the lectures

Relating the above-mentioned topics to this catalogue of concepts, we propose the following relation:

Table 4: Fundamental concepts and lecture topics

Topic	**Fundamental concepts**
Abstract, concrete algebras	Modularisation
Text-, term rewriting systems	Concepts of programming
Functional, imperative, object-oriented and machine-oriented programming	Concepts of programming, syntax, semantics
Verification of functional-, imperative programs	Evaluation
Object-oriented modelling, UML, modelling of automata	Modularisation, Hierarchical structures;
Information theory, coding	
Formal Languages, Chomsky hierarchy and Automata	Languages
Combinatorial and sequential circuits	

The topics "Information theory", Combinatorial, and sequential circuits cannot related directly to the proposed concepts. Here, an extension of the concept catalogue or a more differentiated taxonomy seems necessary.

The relation "OO-modelling, .., modelling of automata" □ "Modularisation, Hierarchical structures" seems unusual; one would expect a concept like "modelling". However, such a concept would be too unspecific, since "modelling" includes all modelling-techniques from graphical modelling to the modelling of algorithms by concrete implementations.

It is natural to assume, that the fundamental concepts in the above topic-concept relation define the fundamental learning targets of the lectures. Therefore, after the first academic year the lecturer of the considered lectures expects from the students a basic knowledge and practical abilities in the following areas: **Structured partition, text and term rewriting systems, Concepts of functional, imperative, object-oriented and machine-oriented programming, Evaluation, Information theory and (formal) languages.**

The matrix of measurable quantities

The evaluation of the lectures is performed using the points the students achieved in the problems of final exams. Since the topics of the problems are associated to the various learning targets, one gets a one-dimensional classification of the problems with regard to the learning targets. Furthermore, the complexity of the problems varies; so, we expand the one- dimensional classification by a second dimension representing the degree of complexity. Each element of the resulting two-dimensional classification is associated to a set of problems and the statistical analysis is carried out for all such sets. Therefore, the two-dimensional classification scheme defines a matrix of measurable quantities.

Table 5: Number of marked problems per category

		Complexity		
		Low	Intermediate	High
Learning Target	Structured Partition	804 (162/642)	2068 (424/1644)	0
	Text-, term rewriting systems	703 (136/567)	570 (95/475)	0
	Functional Programming	800 (161/639)	1631 (1293)	544 (87/457)
	Imperative Programming	800 (161/639)	0	703 (136/567)
	Object-oriented Programming	254 (53/201)	0	0
	Machine-oriented Programming	153 (27/126)	101 (26/75)	0
	Program-Evaluation	0	1165 (235/930)	0
	Languages (Formal)	565 (110/455)	543 (105/438)	0

Table 6 shows the mean value of the points the students achieved in the problems of the various categories; these values are given relative to the maximal reachable points. As above each element of the matrix gives the overall mean value; the mean value for female/male students is given in brackets. The detailed results for the standard deviation can be omitted for the following tables 6-9, since it reaches for all calculations values of about 30%.

Table 6: Mean value of points per category

		Complexity		
		Low	Intermediate	High
Learning Target	Structured Partition	75% (74%/76%)	44% (39%/46%)	-
	Text-, term rewriting systems	57% (48%/59%)	41% (28%/45%)	-
	Functional Programming	57% (50%/59%)	36% (24%/39%)	28% (19%/31%)
	Imperative Programming	46% (40%/48%)	-	26 % (10%/29%)
	Object-oriented Programming	56%(52%/57%)	-	-
	Machine-oriented Programming	56%(52%/57%)	38% (38%/37%)	-
	Program-Evaluation	-	38% (39%/38%)	-
	Languages (Formal)	54% (51%/55%)	40% (28%/42%)	-

The overall results

The (relatively) highest results have been achieved for the learning target "Structured Partition". For the learning targets related to program paradigms (i.e. text-, term rewriting systems, functional programming, … machine oriented programming) we have mean values of about 55 % (low complexity), 38% (intermediate complexity) and 27% (high complexity). It is worth noting, that these values are lowest for imperative programming, whereas the functional paradigm shows better results.

The results for the more theoretical topic "Formal Languages" are in the same interval. The problems on "Program-Evaluation", a rather mathematical topic, reach a mean value of 38 % for problems with intermediate complexity.

To discuss these results it is useful to take into account that a student fails the final exam, if his total amount of points is less than 40 % of the maximal reachable points. Assuming that the average student ought to be able to solve problems with intermediate complexity at the end of the first academic year, one recognizes, that about the half of the students do not achieve the learning targets related to the "Concepts of Programming". The same is valid for the "theoretical" learning targets "Program-Evaluation" and "Formal Languages". Only the "Structured Partition" exceeds definitely the limit of 40%.

The results for the individual lectures

So far, the results for the three lectures as a whole have been presented. Such an analysis was possible, since all three lectures are based on the same learning targets, although the didactical principles of the lectures differ. Therefore, the question arises, whether the above outlined results change dependent on the didactical principle of the lecture. Within the framework of the here considered data such a statistical analysis is not meaningful, since the number of data of the individual lectures gets too low.

Without statistical precision, a basic tendency can be described: The results given in the previous paragraph seems to be valid also for the individual lectures! No principal differences are recognizable!

The results for female/male students

Table 6 shows the numerical results for female/male students in brackets. Dependent on the learning target the results differ more or less. With respect to the learning target "Program-Evaluation", the results show minor differences; male and female have nearly the same mean value. Looking at the learning target "Structured Partition", the differences gets more significant; dependent on the level of complexity the mean values of the female students are 2% - 7% less than the one of the male students. Greatest differences are recognizable for the learning targets "Formal Languages" and "Concepts of Programming": They come for problems with low complexity to 5% - 9% and for those with intermediate or high complexity to 10%-20%. The result for the learning target "Machine-oriented programming" and intermediate complexity differs from that tendency; but since we have only 101(26/75) marked problems of this category, this value is perhaps less significant.

5 Conclusions

Independent on the didactical principle of the lecture, the evaluation of three lectures "Introduction to Informatics I/II" shows the existence of fundamental didactic problems. Greatest problems arise teaching the major learning target of the first academic year, the various concepts of programming. The comparison of the results of the functional and imperative paradigm shows, that the students in the first semester have greatest problems with the imperative paradigm. On the other hand, problems concerning text- or term rewriting systems are better solved than those concerning functional programming are. What may be the reasons of these results?

The syntactic complexity of a program paradigm increases from text/term rewriting systems, over functional to imperative programs. (The semantic complexity behaves perhaps reverse.) The above results suggest, that this higher syntactic complexity is the problem for the students. Algorithms in functional program style are very close to the natural description of a solution, whereas the imperative solution is often shadowed by technical details. This suggests teaching the various program styles in the order of increasing syntactic complexity. Further, it is important to consolidate a program concept, so that the majority of the students have real practical experience with the paradigm. Therefore, the imperative and the object- oriented program paradigm ought to be discussed not until the second semester.

Undoubtedly, a moderate strategy in teaching programming concepts is necessary to increase the learning success. But also the absolute values for the learning targets related to the programming concepts are not satisfying; an average mean value of 38% for problems with moderate complexity is very (too?) low! Are there fundamental problems in teaching programming concepts in the framework of the traditional lecture?

It is a basic principle of learning psychology that the learning success results to a high degree from the self-activity of the student. This is valid especially for the learning of the various programming concepts. However, in the traditional structure of the lecture, the student has a passive role; often the tutorials associated to the lectures do not demand sufficient self-activity from the student. Therefore, it is evident that the student cannot solve programming problems of the final exams having intermediate or high complexity.

Therefore, it is necessary to work out lecture models, which support the students in their self- activity. Such models are currently prepared at the Technische Universität München.

The differences between female and male students particularly in the area of programming concepts are the other major problem resulting from the above analysis. Dependent from the degree of complexity male students reach 10% - 20% higher values than the female students.

What might be the reasons for these differences ?

From studies on school informatics (e.g.: N.Finck, 1998) one knows: The average female student have less precognition on program languages or computer application than the average male student does. Obviously, this fact results in different programming abilities in the first academic year at the university. The only possibility to even out these differences is to support the self-activity of all students especially during the first academic year. Therefore, all efforts point to the same measure: The self-activity of the students has to be supported resolutely! Adequate teaching models incorporating this guiding idea will be discussed in future works.

Literature

Brügge B.: Einführung in die Informatik I, Einführung in die Informatik II, http://atbruegge27.informatik.tu-muenchen.de/teaching/ws00/Info1/, http://atbruegge27.informatik.tu-muenchen.de/teaching/ss01/Info2/.

Broy M.: Informatik, Springer Verlag 1998

Finck N.: Koedukation im Informatikunterricht – Ein Erfahrungsbericht. In: Hyper-Forum für Informatik und Schule, Universität Potsdam, 1998

Knoll A.: Einführung in die Informatik I, Einführung in die Informatik II, http://www6.in.tum.de/info1, http://www6.in.tum.de/info2

Kunter et al.: PISA 2000, Dokumentation der Erhebungsinstrumente, Max Planck Institut für Bildungsforschung, 2002

Schwill A.: Fundamentale Ideen der Informatik. In: Zeitschrift für Didaktik der Mathematik, 1993/1