Process Control Education for Computer Science: Facts and Fiction

U. Schmid, H. Haberstroh, St. Stöckler

Technical University of Vienna
Department of Automation
Treitlstraße 3, A-1040 Vienna.
Email: {s, hh, stoe}@auto.tuwien.ac.at

August, 1993

Abstract

We survey our efforts and experiences regarding the primary practical *Process Automation* for undergraduate students of all branches of computer science at the Technical University of Vienna. It is argued that this task is very different from teaching process control in traditional engineering disciplines, imposing considerable restrictions to contents and organization of a suitable practical. The particular design of two generations of practicals *Process Automation* and our experiences obtained during the last few years are also presented.

Keywords. process control practical, computer science.

1 Introduction

Teaching principles of process control to undergraduate students with a strong technical background, for example in mechanical or electrical engineering, is an enjoyable task. The familiarity of the audience with engineering mathematics, e.g., differential equations and time and frequency domain analysis, helps greatly with introducing topics like system and control theory. A similar statement holds for basic process interfacing topics, namely, sensor and actor devices. Moreover, since most traditional engineering disciplines' curricula provide introductional courses on computer science topics like programming languages, one can even rely on some basic knowledge in that field.

The situation is radically different when teaching process control to undergraduate computer scientists, in particular at the Technical University of Vienna[1]. Only a minority of all first-year students of computer science, namely, those who have attended a (secondary) *Höhere Technische Lehranstalt* have got a technical background; the majority, however, have been in some *Allgemeinbildender Höherer Schule*. Therefore, most first-year students have almost no knowledge of technical disciplines, and even the mathematical and physical background

[1]Computer science at the TU Vienna splits into several branches, e.g., technical computer science, artificial intelligence, and graphical data processing. The appropriate curricula consist of a first general part which is the same for all branches, and a subsequent specific one. The general part aims at a broad general introduction to all important topics of computer science, of course including process control.

varies considerably. Although the first part of the computer science' curricula provide some introductional technical courses, e.g., electrical engineering and computer hardware, they are by far not sufficient to establish a reasonable level of technical knowledge among all students.

Designing a selfcontaining primary course in process control for such an audience is of course very difficult: It should assume almost no prior engineering knowledge, should contain topics which are interesting for all participants (of all branches), must allow for approximately 200–400 students per year, and, last but not least, must be run by a very small staff.

Our Department of Automation is responsible for teaching such a primary course, namely *Process Automation*, located in the first part of all computer science curricula. The course consists of a *lecture* and an associated *practical*. The lecture —designed and run by the department's head, O.Prof. G.-H. Schildt— aims at an overview of process control, see [Sch93]; the purpose of the practical —designed and run by the authors— is to consolidate the knowledge of some selected areas. In this paper, we concentrate on our efforts and experiences regarding the practical *Process Automation* only. Note that it constitutes a matter of its own right (i.e., is relatively independent of the lecture) because of the required selection of topics.

The remainder of the paper is organized as follows: In Section 2, we elaborate on some principal issues characterizing our particular framework. The following Section 3 contains a description of our "first generation" practical (1989–1993) and the resulting experiences. Section 4 finally presents the intentions and expectations underlying our "second generation" practical, which is to be started in 1994.

2 Principal Issues

The facts pointed out in the previous section impose some invariants which must of course be observed when planning a practical in process control:

- Most students have no technical/engineering background. What topics can reasonably be taught in spite of that situation?
- Students of some non-technical branches of computer science do not need any(?) process control knowledge. What topics should nevertheless be taught?
- Very limited time for lecture ($\approx$ 2 hours/week) and practical ($\approx$ 1 hour/week). What may reasonably be done within that time?
- Location of the course in a particular semester of the computer science curriculas, determining:
  - The number of participants (e.g., $\approx$ 350 in the 4. semester, $\approx$ 200 in the 6.). How to organize a practical for that many students?
  - Basic knowledge from previous courses (mathematics, programming, electrical engineering, ...) one may rely upon. What if a required course like *Computer Architectures* is located in a later semester?
- Limited teaching staff (3 assistant professors, 10 tutors, one semester per year) and equipment (16 networked workstations, connected to a few process control target systems). What kind of practical may be run by that few people on the available equipment?

Planning a practical means answering two questions, subjected to the invariants above:

- *Contents*: What topics should be taught?

- *Organization*: How should those topics be taught?

In order to shorten and simplify the presentation of the particular contents and organization of our two practicals (contained in Sections 3 and 4), we proceed with some issues which may be regarded as general ones:

- It is almost impossible to introduce process control at a level of system complexity where problems within the scope of a computer scientist[2] arise, e.g., real-time systems, fault tolerance, and AI in process control, without introducing basic technical issues first.

  We should mention that we explicitly deny exercises which are solely based on playing around with existing tools, e.g., process visualization systems or robots providing capabilities for high-level (e.g., teach-in) programming. Such tasks may be fun for students, but we feel that they do not contribute to an *understanding* of process control issues.

- Lacking technical and engineering background necessiates a thorough mathematical introduction to system and control theory. This is usually covered by dedicated courses in other disciplines; an integration into a basic course takes a lot of the available total time — with questionable success.

  Moreover, the value of *classical* control theory for the average computer scientist is questionable. Apart from the fact that somebody aiming at the graphical data processing or artificial intelligence business will hardly ever need control theory, it becomes more and more exceptional even for a technical computer scientist to deal with a standard control problem directly (e.g., at the programming level).

- Hardware-related problems of process interfaces, e.g., noise, grounding problems, and debouncing, are usually beyond the scope of a primary course. However, understanding and, in particular, programming of process interfaces should be of interest even for a non-technical computer scientist; the interaction between a device driver and a peripheral controller ultimately relies on similar principles.

- Given the limited time provided for the practical, a project-like organization (possibly relying on complicated tools) required for working on (a few) comprehensive exercises does not make sense. Preferable are several small examples, aiming at one particular topic directly, without unnecessary "overhead".

  On the other hand, an exercise should be interesting (ideally fun), which is usually only the case with more complex ones. A switch and a LED panel is simple to understand and to handle; most students, however, would prefer exercises dealing with more realistic technical processes — provided they are easy to handle!

- In view of the limited time it is also of vital importance to exploit existing knowledge. For example, in a primary course, one cannot usually afford the overhead associated with introducing a new programming language like *PEARL* or *Ada*. Relying on the *C* language, which is the "heart" of an (earlier) course in *System Programming*, is surely a better choice (in this respect).

To support the discussion of organizational matters, we finally summarize some advantages and disadvantages of the most important "standard models" for computer science practicals:

[2] In fact, we think that there are too much of too advanced topics — they are often beyond the scope of researchers in the field as well . . . .

(1) *Workshops*

Organized like a seminar, the participants are split up into small groups of 5–10 students each. Working on the exercises is performed more or less collectively, with a strong interaction between the students and the responsible teacher.

+ Optimal knowledge acquisition by individual care and control.
+ Personal relationship between students and teacher, optimal feedback.
+ High utilization of equipment enforced.
+ Knowledge and didactical capabilities of teachers are continuously improved.
– Very high teaching capacity required (because of the large number of groups); preperational effort for teachers very high.
– No suitable way to employ tutors.
– Fixing a date for regular meetings needed; individual scheduling of work time impossible.
– Individual assessment (i.e., marking) difficult.

(2) *Lab groups*

The participants are split up into groups of approximately 20–30 students each. All members of a group have to individually work out a number of exercises and meet their teacher regularily (weekly) for a (spot-checking) examination and discussion.

+ Individual understanding enforced.
+ Individual assessment possible.
+ Individual scheduling of work time possible.
+ Tutors may efficiently be used for supporting participants.
+ Good feedback.
– Require a large number of short examples, which have to be approximately equivalent (among the different groups) for reasons of fairness.
– Fixing a date for (long) regular meetings needed.
– Examination in front of the whole group is stressing.
– Discussion requires supporting equipment, e.g., a video beamer.

(3) *"Classical" computer practicals*

Examples are individually assigned to participants (or, alternatively, to small teams of 2–3 students each) and examined. The final assessment, however, depends on the results of a few additional tests.

+ Individual understanding enforced.
+ Individual assessment possible.
+ Individual schedule of work time allowed.
+ Tutors may efficiently be used for supporting participants.
+ Reasonable feedback.
– Require a very large number of short examples, which have to be approximately equivalent for reasons of fairness.

- Separate provisions for discussions/teaching needed.
- Examination is time-consuming (source-code reading!), boring and not very efficient due to possibly copied solutions.
- Marking of the tests required.

So every standard model has its particular advantages and disadvantages. We tried to find ways how to merge and extend them in order to combine the most important advantages, but without retaining the most cumbersome disadvantages.

3 The "First Generation" Practical (1989–1993)

When we started planning the practical *Process Automation* in 1988 we had the opportunity to design a completely new course, i.e., there were no need to use already existing equipment or to teach certain topics. This gave us the possibility to design an up-to-date practical, taking into account all the facts mentioned in the preceding section (see [SS91] for a detailed presentation of contents and organization).

3.1 Contents

In a first step we decided which topics are to be taught and defined three major subjects, nameley *Control Theory*, *Petri Nets*, and *Development of Process Control Software*.

- **Control Theory.** Because of the lack of technical background, we restricted the topics in this field to an introduction to the basics of classical (continous) control theory. The aim was to impart a rather intuitive understanding of simple closed-loop controls to the participants.

  A special visualization software has been developed which allows to explore the behavior of basic control loop elements (e.g., PID-controller, PT_1, PT_2, ...) and closed-loop systems in the time domain. This software (offering only a limited choice of elements and input-signal waveforms, thereby stressing the essentials only) allows to inspect the influence of characteristical parameters on the behavior of control systems in the time domain without dealing with formal/mathematical methods (like Laplace-Transformation).

- **Petri Nets.** An important step in designing software for process control is modeling of parallel tasks and their interactions. Petri Nets provide a simple and expressive means to do this.

  The ability to model given systems using state-transition systems (like Petri Nets) is vital not only in the field of process automation but also in many other branches in computer science. The basic teaching objective is to enable the participants to recognize the structure (states and transitions) of a parallel system and to identify standard situations (e.g., synchronization or mutual exclusion). Once modeled, it is (more or less) easy to analyze the behavior of the abstract system. The analysis of Petri Nets comprises the examination of a multitude of different possible transition sequences (i.e., execution sequences).

  Again, a special graphical software was developed to support computer aided teaching of Petri Nets models. This software allows to create (syntactically correct) Petri Nets and to perform simple analysis tasks.

- **Development of Real-Time Software.** This most important part of the practical (consuming approximately 66% of the total time) gives a first introduction to the work of an computer scientist in the field of process automation. The objectives of this part are twofold; on one hand the students should learn how to write device driver routines for the installed process interfaces, on the other hand, they have to develop (simple) application software controlling the connected models (using their own device drivers).

  Although, it is difficult to keep pace with the rapid evolving fields of process control hardware, real-time operating systems, and software development tools, we tried to provide a state-of-the-art environment according to the actual industrial standard. This system consists of standard VME-Bus components (M68000 CPU and several analogous and digital I/O-cards), a real-time operating system, a C-programming language cross compiler, and a high level language debugger. This microprocessor system is connected to several models including a train set.

3.2 Organization

The invariants listed in Section 2 (number of students, limited teaching staff and equipment) severly constrained our choices regarding the form of organization; obviously, teaching in workshops or lab groups is very time consuming and therefore not adequate to our resources. Facing the facts, we decided to develop a new concept of teaching the topics introduced above. The main components of this concept are:

- Leave the *scheduling of time* to the participants. This induces a maximum of flexibility because the students may choose an individual schedule, for example, just two extensive sessions or weekly sessions all over the semester. To support planning, each workstation can be reserved one week in advance (however, the weekly working time is limited).

- Provide *extensive literature*, containing all topics in detail and including necessary manuals. These scripts ([SS93a, SS93b, SS93c]) make students independent of the presence/absence of an assistant professor or tutor.

- Exploit the advantages of *computer aided teaching* by means of specially designed software for teaching certain topics.

The scripts appertaining to the practical include a large collection of exercises the students are free to choose from (see [SS93c]). Each example is classified according the objectives it covers, a crosstable of all learning objectives and all problems helps to choose appropriate ones. During certain times (all over the week) the participants can address tutors to discuss particular problems or to verify their solutions; tutors are selected students trained in a special course preceding the practical. To support the students four special lectures and two discussions are held by the assistant professors during the semester. The individual marking is done in two independent written test.

3.3 Experiences and Conclusions

This form of practical has been run by the authors five times (1989–1993) and gave them the opportunity to gather a lot of experiences:

- The most interesting and surprising one is that many students do not appreciate the possibility of individual schedules over the whole semester. They claim for certain milestones where solutions to particular problems are to be delivered.

- On the other hand, the (short term) reservation of work time/workstations is appreciated.
- The idea underlying our selection of topics, namely, to cover a relatively broad range of process control topics at some reasonable high level, has proved unsuccessful.

  In fact, we found that the average participant —even if he/she is willing to spend some time on the practical, which is by far not obvious— has considerable difficulties in mastering the whole bulk of exercises. This is mainly because of (1) the fact that some of the topics seem to be more or less uninteresting for a computer scientist, (2) the missing mathematical and technical background, and (3) the heterogeneity of the selected topics. The problem is further complicated by the quickly approached (high) level of view, which of course imposes a very brief treatment of seemingly(?) straightforward trifles.
- Our complex process models (in particular the railway model) have proved deficient in operation and inflexible with respect to the development of additional exercises.
- Lacking knowledge in computer architecture and system programming led to unexpected problems in solving problems concerning these topics (e.g., writing device drivers).

These experiences were considered when the 'second generation' of the practical was planed. The next section will give an overview of the new structure.

4 The "Second Generation" Practical (1994–199x)

The contents and organization developed for our second generation practical are of course based on the experiences with the first generation one. However, in order to keep the paper concise, we omit a detailed reasoning of each and every decision — the key issues are usually straightforward anyhow.

4.1 Contents

The contents of our second generation course cover a very restricted topic only: *programming of process interface hardware*, however, treated in some detail. Our catalogue of exercises is based on a (fine-grain) collection of subject matters, primarily

- *Digital I/O*: Both preliminaries and more advanced features like change-of-state detection for digital inputs, port I/O with handshaking, ...,
- *Analog I/O*: Basics and more advanced exercises covering timer-triggered analog input, basic signal processing, ...

For each subject matter, we selected[3] a number of (equivalent) exercises which stress the major issue directly, i.e., without unnecessary overhead.

The process models required for the exercises are kept as simple and robust as possible; LED and switch panels, oscilloscope, and waveform generator are often sufficient. All models are connected to an M68030-based VME microprocessor *target system* via the low-cost, multi-vendor *M-Modules* I/O-System, which provides dozens of different process interface modules, see [MMSa] and [MMSb] for details. The programming of more complex exercises is supported by the target's real-time operating system (ISI/SCG's pSOS^{+m} and add-on's).

[3] In fact, at the time of writing we were still in the phase of selection.

In fact, the development environment employed is similar to the one used in the first practical. It consists of several[4] identical *development cells*, each comprising four PC-based *workstations* and one (abovementioned) M68030 VME *target system* (including all process models) connected via Ethernet. The workstations provide a *C* cross-development environment (Microtec MCC68K$^+$, XRAY68K$^+$) supporting our target systems. All development cells are also connected to an additional DEC Alpha *(file) server*.

Programs may be developed and compiled at any workstation simultaneously; the actual degree of simultaneity depends on the number of software licences we can afford. Debugging is also commanded and controlled from the workstations, but requires (of course) the target system and is therefore mutual exclusive (that is, per development cell). Note that any workstation provides the same capabilities to the users since all user data are located at the file server and all development cells are identical.

4.2 Organization

We have decided that our practical should be organized as a combination of *lab groups* and *classical practicals*. Each student has to solve several exercises which have to be presented within a certain schedule. In addition the teacher will ask some questions to the topic of the exercise. This allows a relatively objective determination of the individual capabilities which is based on several ratings. For each of the examinations the student has to register separately by reserving a *time slot* with his teacher.

The time for working out the exercise on the computer can be chosen individually. During the practical the students are supervised by assistant professors and tutors. In order to realize such a practical with an acceptable effort, it has to be organized strictly. To support the organization as well as the examinations we have developed a special software. This system includes *registration* and both *scheduling* and *support* of *examination*. In the sequel we describe the (planned) course of the practical and the supporting software.

Within the first week of the semester each student has to register on the computers of the practical. The registration program records the essential data and allocates an assistant professor and a first exercise to the student. It also installs the programming environment of the student (accounts, home directory, ...).

During the first week the students can also buy the scripts which include the collection of all possible exercises. After this week reserved for registrations, the students can use the computers at any time for their exercises.

The examinations for each exercise take place within a specified week. Each teacher can create schedules (by computer) where he can determine the *time slots* for examination individually. The only restriction is that there should be no more than two examinations concurrently so that the third assistant professor can step in when one of the two drops out. The students can chose a *time slot* for their examination, but only for the current exercise.

The course of an examination will be as follows: The candidate has to present and discuss his/her solution which must be provided before the examination week at a specified location. He/she has to provide the listing of the program too. To get an idea of the candidate, the assistant professor can look into all relevant data (Name, exercise, present marks etc.) of that candidate during the examination. Additional questions to the topic of the exercise will be asked to which the student may answer verbal or in writing.

There can be used predefined rating criteria for the records the teacher makes during the examination. A mark will be computed and proposed by the system. If the assistant professor

[4]We will start with two development cells in 1994; one or two additional ones are planned for 1995.

is not satisfied with the proposition, he can of course enter any text and rating him/herself. Finally, the student gets his next exercise.

The system automatically computes statistics of the marks of every exercise and question. This provides a feedback for the teachers to find out the topics the students have problems with.

The examinations are executed on the same computers the students work with, so that they can use their familiar environment for the presentation of their solutions. Therefore the data of the students has to be stored only encrypted on floppy disks to avoid manipulations. The catalogue with questions and the collection of all exercises are available via network but also encrypted and not readable by students.

The questions and collection of solutions are additionally kept in hard copies so that examinations can also be held in the case of a computer breakdown. For this case separate forms are developed to enable the easy offline recording of the examination data.

4.3 Expectations

- An important feature of the contents of our new practical is *flexibility and scalability*. Since we aim at teaching particular subject matters by means of short examples relying on dedicated small (and therefore cheap) process models, it is affordable to discard inappropriate ones. Moreover, additional subject matters and related examples —possibly including some basic control theory or even fuzzy control— are easily added later on if desired.
- The personal supervision of the students by assistant professors should provide optimal feedback.
- The personal examination with additional questions shows the understanding of the topic (students cannot "cut and paste" the examples).
- Computer supported tutoring could be introduced in the next generation course to make individual scheduling of work time more flexible for students.

References

[MMSa] MUMM e.V.: *Basic M-Modules Specification*, Nürnberg, Germany.

[MMSb] MUMM e.V.: *M-Module Directory*, Nürnberg, Germany, August 1993.

[Sch93] Schildt, G.-H.: *Prozeßautomatisierung*, Techn. Univ. Wien, Inst. f. Automation, 1993. (in german)

[SS91] Schmid, U.; Stöckler, St.: *Konzept der Laborübung Prozeßautomatisierung*, Techn. Univ. Wien, Inst. f. Automation, Report 183-1/21, June 1991. (in german)

[SS93a] Schmid, U.; Stöckler, St.: *Prozeßautomatisierung — Übungsskriptum*, Techn. Univ. Wien, Inst. f. Automation, 1993. (in german)

[SS93b] Schmid, U.; Stöckler, St.: *Prozeßautomatisierung — Anhänge (Manuals)*, Techn. Univ. Wien, Inst. f. Automation, 1993. (partly in german)

[SS93c] Schmid, U.; Stöckler, St.: *Prozeßautomatisierung — Beispielsammlung*, Techn. Univ. Wien, Inst. f. Automation, 1993. (in german)