A Course in Software Portability

James D. Mooney
Dept. of Statistics and Computer Science
West Virginia University
Morgantown, WV 26506

ABSTRACT

This paper describes an experimental course on the topic of software portability, and initial experience in teaching this course. With the continuing proliferation of both applications and computing environments, the need for portability is being increasingly recognized. A large proportion of the software now being developed will eventually need to be ported to new environments. Yet this topic is missing from most computer science and software engineering curricula.

The course described here was designed to explore practical issues in the development of portable software. Lectures and discussions on portability topics are combined with the ongoing development of a simple software project designed to expose a variety of portability problems. During the course the project is ported to several environments and redesigned to improve its portability.

This course has been taught experimentally with encouraging results. Student assignments have used novel and effective methods to overcome portability barriers. Feedback from students indicates that they have become more aware of portability issues to be considered in software development, and have gained experience with system interface issues in several programming environments.

1. INTRODUCTION

The continuing proliferation of both applications and computing environments creates an urgent need to overcome the barriers to software portability. It is increasingly likely that much of the software now in use or being developed will face the need to be ported to new environments during its lifetime. In a recent survey conducted to identify important future challenges for software development [Lewis and Oman 90], the problem of portability placed high on the list.

It is often believed that the portability problem has been solved by the development of standardized programming languages, or by the emergence of universal computing environments such as UNIX. Each of these factors are important, but they still fall far short of supplying complete solutions to the portability problem. This is especially true when it is necessary to port current software with complicated requirements such as timing constraints or interactive user interfaces. Some of the problems of software portability are discussed in a recent survey by the author [Mooney 90].

In spite of the recognized importance of portability, knowledge and awareness of portability issues and techniques is not widespread. Much research remains to be done, but even the knowledge that has been developed on this topic is not being well disseminated. Portability considerations are important in the software development process, but they are not a clear part of common methodologies or curricula in software engineering. As a consequence, much software is being produced that is not as portable as it could be, and the task of porting existing programs remains difficult.

The course described in this paper is a contribution towards solving this problem. The course combines a series of lectures and discussions on portability topics with laboratory experience in the actual development and porting of a simple portable program. In the laboratory series a single program is designed for a single environment, ported to at least two additional environments, and redesigned to improve its portability. During initial class testing, many students devised novel and effective strategies for overcoming portability barriers encountered in these projects.

2. OUTLINE OF THE COURSE

2.1 Textbooks

A principal problem to be solved in the design of a new course is the choice of a suitable text. There are a limited number of books devoted to the subject of software portability, and most of these address a limited body of issues. A classic but now dated treatment of portability issues is given by [Brown 77]. Other more recent books on the topic include [Dahlstrand 84], [Henderson 88], [LeCarme et al 90], and [Wallis 82].

None of these books is designed as a textbook, and each has its strengths and weaknesses. LeCarme was selected for the course because it offers up-to-date coverage of a reasonable range of topics, and especially because it includes a number of case studies on the porting of commercial microcomputer software. Information in these studies was obtained by interviews, since few such studies have been openly published.

2.2 Course Content

There are several points of view that can be taken for software portability, which depend on the specific objectives to be attained. Our course focuses on two principal objectives in the software development and maintenance process:

1. To develop original software in such a way that it can be more easily ported to a variety of target environments;

2. To choose effective strategies for porting existing software to specific new environments.

Topics for the course were selected to meet these objectives. Course material was taken both from the text and other available sources, including the instructor's research and experience in standardization projects. The text was supplemented with several class handouts.

A complete list of topics covered in the course is given in the Appendix.

2.3 Presentation Format

The course was initially taught during a six-week intensive summer session (five 90-minute classes per week). A standard lecture format was used during most class sessions. However, since the topic includes many unresolved issues, some subjects were best explored by open discussion. One open area of particular interest is the problem of how to specify and measure portability attributes and requirements. Students were encouraged to contribute their own ideas to this discussion, leading to a better understanding of the issues by class and instructor alike.

3. PROJECT DESCRIPTION

This section describes the laboratory project used in the course to provide students with direct experience in overcoming the problems of portability. The chosen strategy was based on a sequence of assignments all using the same basic program. The goal was to develop an initial version of the program, investigate the problems of porting it to various environments, and improve its intrinsic portability based on that experience.

3.1 Resources and Support

To perform experiments in software porting, it is necessary to make use of multiple computing environments. Moreover, to enable students to complete a meaningful set of experiments in the time frame of a single course (especially a six-week session), a reasonable and consistent infrastructure must be provided.

The need for a variety of computing environments may seem imposing at first, but the majority of Computer Science and related departments now have access to a reasonable assortment of personal computers, as well as one or more timesharing systems available via networking. In our department we made use of four distinct environments:

1. A cluster of VAX systems running VMS, available to the entire university via network;
2. A distributed UNIX environment available in our department, including both VAX and SUN computers;
3. A large assortment of Macintosh computers available in our undergraduate labs;
4. A limited assortment of IBM-PC-Compatible systems.

An essential element for providing a consistent infrastructure is the selection of a suitable programming language. Such a language must be suitable for the application, and well standardized; moreover, it must be available for each target environment. Language conversion was not considered to be a reasonable porting strategy within the context of this course.

The only language which came close to meeting the necessary requirements for our facility was C. C compilers were available for all four of the chosen target environments. Most of the compilers were reasonably conformant to the ANSI C standard. Moreover, the comprehensive standard C library, also available in reasonably consistent form, enhances the portability of various operations such as character input or file access, that may create problems when other languages are used. For all of these reasons C was chosen as the project implementation language.

An additional problem to be faced was the physical transport of files among the various environments to be used. Since our goal clearly was source portability, not binary portability, it was sufficient to provide transport for text files only. No common media exists among such diverse systems. However, the department network facilities linked the larger systems with at least some of the Macintoshes and PCs. File transfer software was available (either ftp or kermit) to provide reasonable support for all of the transfers required.

3.2 Problem Description

The goal in selecting a programming problem for the portability assignments was to compose a specification in which the basic functionality was not complex, but elements were included which would expose a number of specific portability problems.

The problem chosen for the initial course was the design of a simple interactive quiz program. The program was required to read a set of multiple choice questions from a file, and then conduct an interactive quiz based on a random selection from these questions. At the end of the quiz, statistics were to be displayed showing the results of the quiz. To focus attention on portability issues in the limited time available, a skeleton for the program was provided by the instructor.

The first version of the program was developed using VMS, then ported to the UNIX network. Later assignments combined functional enhancements, additional porting, and redesign for improved portability.

The primary portability problems included in the first version of the program were:

1. Acquisition of command line parameters (question file name, number of questions to ask);
2. Display formatting (especially, a common procedure for "clearing the screen");
3. Time measurement (the final statistics had to include the actual time taken for the quiz).

Students were required to recognize during this assignment that perfect portability did not always imply identical behavior in all respects. In particular, the preferred method for processing command line parameters was to use the conventional syntax for each environment, rather than a common convention for the particular program.

Later assignments included functional enhancements which led to additional portability problems:

1. Setting a time limit, after which the quiz was to be terminated (with extra credit for *immediate* interruption);
2. Maintaining an auxiliary score file to record the ten highest quiz scores (using an appropriate name for each environment);
3. Graceful termination and proper reporting of error conditions.

The final assignment required porting to a personal computer environment. Extra credit was offered for conversion to a menu-driven user interface, in which the quiz could be conducted repeatedly using various options. Most students took advantage of this opportunity.

4. ADDITIONAL ISSUES

Our quest for maximum portability during the assignments led to exploration of some problems and issues that were not originally anticipated.

4.1 Version Management

The question of how to manage different versions of a program module for multiple systems led to examination of the role of conditional compilation. This facility, available through the C language preprocessor, enables code to be optionally compiled depending on the condition of certain preprocessor variables. This strategy can be used to support multiple versions by defining a variable for each version to be used; the disadvantage is that all versions must be combined in a common source file, which will change as each new version is defined. We concluded that conditional compilation can sometimes be a useful tool for version management, but should not be considered as a primary strategy for portable design.

4.2 Hiding Processor Differences

Our UNIX environment includes several different processor types serving a common set of users and sharing a network file system. Users can freely log on to any processor; however, executable files prepared for one processor type cannot be executed on another. This led to confusion for students who habitually used different processors. It clearly is necessary to compile programs separately for each intended target processor. The related problem we explored was how to make the difference invisible to subsequent users, so that identical commands could be used regardless of the processor being run.

In a similar vein, subtle differences across microcomputer families led to cases in which programs developed using one model would not execute properly on another. These differences ranged from different operating system versions to different types of graphic hardware. In some cases, students had to confront the choice between taking advantage of special capabilities such as color graphics, or maintaining a wider range of portability.

4.3 Ownership and Location

Some peripheral but interesting problems were encountered through attempts by the instructor to test student programs on the timesharing systems, when those programs were owned by different accounts and placed in different file directories. This type of access was always possible, but required careful attention to file protection settings and file name specification. A common mistake was to place the quiz score file in the current directory (which could vary) rather than in a fixed location.

5. DISCUSSION AND CONCLUSIONS

This paper has presented an outline for a course on software portability, and some experience gained during its initial implementation. This course will continue to be developed, and further refinements will of course come from further experience.

A fully suitable textbook is an unfulfilled requirement. LeCarme contains much beneficial material, but excessive space (for our purposes) is devoted to issues of language translation, and some other issues are not treated as fully as desired. More significantly, the book is not a text; it is not always tutorial in its approach, and it contains no assignments, summaries, or detailed suggestions for class use. The same is true for the other books available; no true text for teaching software portability has yet appeared.

Student feedback from the initial presentation of the course has been quite positive. Students found that they not only developed an awareness of portability issues, but also gained useful experience in programming for a variety of specific environments. Several students produced outstanding (and impressively portable) project implementations. Some found themselves scanning volumes of system documentation and contacting vendors or discussion groups to answer subtle questions about timer interrupts or command handling.

The course has been taught only in a six-week summer session, but it should be easily adaptable to a longer term. The class hours would be about the same, but students would have more time to digest the material, and especially more time to work on laboratory assignments. The assignments consumed the greatest share of time; it is likely that more lecture material could be covered in the longer calendar span of a normal semester.

The resources required for this course are significant, but the course can be adapted to many different configurations as long as a few distinct system types are available. This requirement should be met by most institutions. This year the C Language was the only feasible choice for the project. More affordable Ada compilers are now appearing for smaller environments, which may make this language a reasonable alternative in the future.

Even if this course is beneficial, however, the question remains: how could it fit into a typical curriculum? Software portability deserves a clearer place in the spectrum of software engineering education. A course like this could be an optional offering in a sequence of development-oriented courses. Alternately, a shortened version of this course could be used as one component of a broader course in software engineering.

Software portability is not an identified topic in older curriculum proposals, but this course can easily find a place in the new Computing Curricula 1991, jointly developed by ACM and IEEE/CS [Tucker 91]. The course can stand alone as an Advanced Software Engineering course, or its content can be integrated into basic software engineering courses. It contributes to knowledge units SE2 and SE4. Its central recurring concept is reuse, but it also deals with aspects of binding, efficiency, evolution, levels of abstraction, and trade-offs and consequences.

Perhaps only limited material on this topic can be used in most existing curricula. Nonetheless, our experience has shown that some explicit attention to portability issues is valuable, and that this topic can most effectively be explored through actual experimentation.

ACKNOWLEDGMENTS

I would like to thank the students of CS 291/391 and the anonymous referees for their helpful comments.

REFERENCES

[Brown 77] P.J. Brown (ed.), *Software Portability*, Cambridge University Press, Cambridge, England 1977.

[Dahlstrand 84] I. Dahlstrand, *Software Portability and Standards*, Ellis Horwood, Chichester, England 1984.

[Henderson 88] J. Henderson,, *Software Portability*, Gower Technical Press, Hants, England, 1988.

[LeCarme et al 89] O. LeCarme, M. Pellissier Gart, and M. Gart, *Software Portability with Microcomputer Issues*. McGraw-Hill, New York, 1989.

[Lewis and Oman 90] T.G. Lewis and P. Oman, The Challenge of Software Development, *IEEE Software*, Vol. 7, No. 6, Nov. 1990, pp. 9-12.

[Mooney 90] J. Mooney, Strategies for Supporting Application Portability, *IEEE Computer*, Vol. 23, No. 11, pp. 59-70.

[Tucker 91] A. Tucker, (ed.), *Computing Curricula 1991*, ACM/IEEE-CS Joint Curriculum Task Force, ACM, Order No. 201880, 1991.

[Wallis 82] P. J. L. Wallis, *Portable Programming*. John Wiley & Sons, 1982.

APPENDIX: TOPIC OUTLINE

GENERAL CONCEPTS

What and why; The growing importance of portability; Degrees of portability; Binary and source portability; Transportation and adaptation; Program interfaces; Program design, system design, and installation.

THE ROLE OF STANDARDS

What is a standard? Defacto and formal standards; Principal standards organizations; Examples.

TRANSPORTATION ISSUES

Media compatibility; File systems; Transportable media vs. networks.

REPRESENTATION ISSUES

Standard languages and their limitations; Use of "portable subsets"; Portable compilers; Changing languages; The linking problem: standard intermediate forms.

INTERFACE ISSUES

Standard libraries; The operating system interface; Portable operating systems; Standard environments; Interface levels (low and high); The hardware interface; Interpreters and abstract machines.

SPECIAL REQUIREMENTS

Memory management; User interaction; Real-time applications; Multitasking applications; Architecture-independent parallelism; Changes of algorithm.

OTHER ISSUES

Specification and metrics; Dynamic portability across a network; People portability; Data portability; Portability vs. reusability; Cultural adaptation: internationalization, user interface; Non-technical issues (copyright, etc.).

CASE STUDIES

Portable applications; Portable compilers; Portable operating systems; Portable libraries; Standard representations; Standard interfaces.