Microprocessor software engineering training: a case study

by CHRISTINE L. BRAUN
SofTech, Inc.
Waltham, Massachusetts

INTRODUCTION

The rapid growth of the computer software field has resulted in a problem that is beginning to threaten that growth—a critical scarcity of skilled software engineers, trained in modern software engineering practices and able to function effectively in a real working environment. In response to this need, industrial training, through public courses, in-house programs, and audio-visual packages, has become one of the fastest-growing areas of the computer field. However, such programs frequently fail to provide the depth of training that is really required to produce skilled software engineers, and are often too general-purpose to address the particular needs of the company procuring the program.

ITT Defense Communications Division (ITTDCD) recognized this shortage of skilled professionals and realized that a serious commitment to training would be required. They determined that their needs could best be met through an in-house course, tailored to the ITTDCD working environment and designed to train existing scientists and engineers to the point where they could function effectively in that environment.

ITTDCD, together with SofTech, Inc., conceived a course to accomplish this objective, to be designed and presented by SofTech. The course was planned to run for nine weeks, covering all phases of a software project as it is conducted at ITTDCD, from analysis, through design, coding, and testing, and was oriented toward microprocessor software development, in keeping with ITTDCD’s application requirements. Modern software engineering practices were to be incorporated and emphasized throughout. Because ITTDCD produces software in a DoD environment, military standards for development and documentation were to be introduced and employed. A major aspect of the course was to be a single running class project, to be followed through the entire software life cycle as it was presented in the course, thus simulating the actual working environment that the students would experience after the course.

This paper describes the curriculum developed for the course and presents our experiences in conducting it for the first time. The discussion emphasizes the attempts to parallel the actual working experience, and points out the many counterparts to “real life” that in fact occurred during the program.

OVERVIEW

The students

The course was initially presented to a group of ten scientists and engineers, all of whom had chosen to take the course and indicated a desire to transfer to a software engineering position afterward. All students had some familiarity with computer architecture, and all had previously been exposed to programming, typically at the level of one college FORTRAN course. Thus the course was geared toward students with this background.

The engineering background of the class permitted a relatively sophisticated presentation of certain topics, such as the hardware-software interactions and tradeoffs during system analysis, the microprocessor architecture, etc. It also led to selection of a class project more readily understood by engineers. This mix of hardware engineering background with training in modern software engineering is expected to produce individuals uniquely qualified for microprocessor software engineering applications.

Curriculum

The nine week program is organized into four high level subject areas called “modules.” These are:

- Functional Analysis
- Top Level Design
- Detailed Design
- Microprocessor Coding and Testing

Each module is further subdivided into one or more “units” of one week or less, each addressing a particular course topic. Figure 1 presents the curriculum outline, and Figure 2, is a timeline showing the phasing of the various units.

The Functional Analysis and Top-Level Design modules present only an introduction to these topics, reflecting the goal of the course to produce software developers rather than analysts. However, it is essential that a programmer understand the relationship of his activity to that of the systems analysts and designers, who define the requirements he must meet and the interfaces he must observe. The intent
Module 1 - Functional Analysis
Unit 1.1 - Introduction to Analysis

Module 2 - Top Level Design
Unit 2.1 - Introduction to Design

Module 3 - Detailed Design
Unit 3.1 - PASCAL as a PDL
Unit 3.2 - Structured Programming
Unit 3.3 - Documentation, Testing, and Debugging
Unit 3.4 - Design Reviews and Structured Walkthroughs
Unit 3.5 - Host Computer Facilities
Unit 3.6 - Detailed Design Lab

Module 4 - Microprocessor Coding and Testing
Unit 4.1 - Microprocessor Architecture and Instruction Set
Unit 4.2 - Advanced Coding Techniques
Unit 4.3 - Microprocessor Development Facilities
Unit 4.4 - Microprocessor Lab Exercise

Figure 1—Curriculum outline.

Presentation approach

The course runs for 6 hours per day (in two 3-hour sessions) five days per week. It combines lectures, exercises, and lab sessions to permit students to gain direct experience with new concepts. Where practical exercises and labs are oriented toward the overall class project, to simulate the various aspects of a real-life project. The lecture format combines slide presentations with more informal blackboard walkthroughs, and encourages class participation and interaction at all times.

To permit evaluation and enhancement of the program for further use at ITTDCD, the first presentation of the course has been recorded on videotape. Students also completed weekly unit evaluation forms to provide an input to this process. A report evaluating the first presentation and recommending revisions and enhancements is to be prepared by SoftTech. for use by ITTDCD in planning further training activity.

MILITARY STANDARDS

A major requirement for a software engineer in ITTDCD’s development environment is an understanding of military software development and documentation standards—experience that must be acquired on the job by many new employees. The training program introduces this concept from the start, explaining the requirements at each phase of the software life-cycle. The particular standard followed in the course is the Navy’s MIL-STD-1679, a new standard that will be applicable in much of ITTDCD’s future business. The course briefly describes other military standards and compares and contrasts them to 1679.

MIL-STD-1679 specifies various practices to be employed in software development, and also dictates a particular sequence of documentation to be produced. During the course the students prepare (or are given) several of the standard documents for the class project system. Documents not actually developed in the course are explained in detail. Other requirements, e.g., coding standards, are practiced as the
class develops software. The military standards for project reviews, quality assurance, and configuration management are also explained and practiced. Students are also taught to read and interpret Contract Data Requirements Lists (CDRLs) and Data Item Descriptions (DIDs).

THE DEVELOPMENT SYSTEM

The host development system introduced in the course is the PDP-11/70 under the UNIX* operating system, a host used on several ITT projects. Running under UNIX is the Change Control Library Facility (CCLF), a program support and configuration control tool developed by SofTech under separate contract to ITTDCD.

The target microprocessor taught is the Intel 8080. Students develop 8080 programs on the PDP-11/70 using the XAS8 Assembler and Linkage Editor, both developed by SofTech under separate contract to ITTDCD. Programs are debugged using the Stand Alone Emulator Package (SAEP), an 8080 instruction simulator developed for ITT by BDM Corporation.

THE CLASS PROJECT

The project implemented as the major exercise in the course is the Auto Tune Antenna System (ATAS), a hypothetical microprocessor-based system that involves processing representative of actual ITTDCD applications. ATAS is a system that automatically tunes a submarine antenna to a desired transmission frequency and maintains tuning, relieving the radio operator of the need to continually monitor and adjust the tuning. (A submarine antenna’s tuning is affected by the action of the waves against the antenna.) Figure 3 illustrates the ATAS operator’s console, and Figure 4 shows the system I/O interfaces.

The basic functions of the ATAS software are:

1) Initialize the system to its start state (lamp settings, tap position, etc.).
2) Read the frequency keyed in by the operator and check it for validity. (The keyboard is checked for possible input at every Real Time Clock interrupt. A SELECT key indicates the start of the frequency select mode, and an EXECUTE key indicates that frequency key is complete.)
3) Coarse tune the antenna to the selected frequency, by interpolation into an in-core table of frequencies and corresponding tap positions. Enable RF after coarse tuning is accomplished.
4) Monitor tuning by regularly (every 10 Real Time Clock interrupts) sampling the Voltage Standing Wave Ratio (VSWR) and taking corrective action if it is outside accepted limits for a certain number of measurement intervals. (Corrective action involves moving the tap to minimize the VSWR.) If acceptable tuning cannot be obtained, the status lamps are set to indicate this, the RF is disabled, and tuning is discontinued. (At any time, the operator can depress SELECT and select a new frequency.)

The ATAS system introduces numerous programming concepts, including:
- interrupt handling
- foreground/background processing
- data aggregates (frequency-tap position table)
- mathematics (interpolation)
- complex flow of control (tuning adjustment)

The system is specified with a limited amount of RAM and ROM, and students are given budgets for their various subprograms. This is a realistic design constraint for microprocessor software, and encourages students to coordinate and make tradeoffs during implementation.

THE COURSE EXPERIENCE

The following subsections narrate our week-by-week experience in presenting the course for the first time.

Week 1—functional analysis and top level design

The course began with a review of computers and programming, in order to ensure a common starting point for discussion. This review included a description of various support software tools, such as operating systems, compilers, assemblers, linkage editors, and simulators.

The presentation of analysis and top-level design was then introduced by a discussion of the concept of total system (hardware and software) development. The role of a system engineering organization was presented, and the process of functional allocation between hardware and software was addressed. The class informally worked through the analysis of some example microprocessor-based systems (automobile

---

* UNIX is a trademark of Bell Laboratories.
cruise control, elevator controller). The ATAS system was then introduced, and aspects of the analysis of this system (performed by the instructor staff functioning as "system engineering") were presented.

Discussion then turned to the phases in the software development life cycle (analysis, top level design, detailed design, implementation, test and maintenance), and MIL-STD-1679 was introduced by describing the requirements it imposes on each phase. Students were given the ATAS Program Performance Specification (PPS) that the instructors had prepared, and were taught how to interpret and work from a PPS, and how to interact with systems engineering regarding problems or questions on the PPS. Further emphasis was then given to the role of the Military Standard, and to the other requirements it imposes on the analysis and top-level design phases.

The week also included an overview of various methodologies supporting analysis and top-level design, including SofTech's Structured Analysis and Design Technique (SADT™),** PSL/PSA, Structured Systems Analysis, HIPO, Jackson Design, and Warnier/Orr Design.

Finally, the class established a document library to be used for the ATAS project, and was introduced to the concept of the author-librarian-reader review cycle.

Week 2—Pascal as a PDL

Because the target computer for the course was to be programmed in assembly language, it was considered particularly important that students be exposed to a modern high-level language containing structured programming constructs for use as an initial design mechanism. Pascal was selected for this purpose, and was taught as a Program Design Language, stressing concepts rather than perfect syntax. As students generally had little programming experience prior to this, this week was oriented toward "how to design programs," with Pascal as the language for design. Top-down design was emphasized throughout the week, with several examples of problem decomposition (e.g., Eight Queens²). Programming exercises and examples were, in many instances, designed to present concepts needed in the ATAS exercise (e.g., multiplication by shifting and adding). This week allowed substantial time for students to work on exercises reinforcing the lecture material, and to discuss them afterward.

As a by-product of this unit, students learned to read and work with both BNF grammars and syntax diagrams. Also, though it had not been planned, student interest prompted some discussion of rudimentary compiler theory. The students' hardware background seemed to make them particularly aware of the apparent difficulty of this process.

Week 3—structured programming

This unit began introducing more formal aspects of structured programming, and continued the teaching of the programming process begun in Week 2. The theory of structured programming was presented, and the basic set of control structures was discussed. One exercise was the restructuring of an intentionally unstructured flowchart that appeared
in the ATAS PPS to describe the tuning adjustment algorithm. Structuring a program into a collection of functionally simple subroutines was also emphasized.

Various descriptive techniques for developing and expressing design were presented. These included the Data Structure Diagram for describing data structures, and the Calling Hierarchy for illustrating program decomposition. The Finite State Machine concept was introduced as a design technique, and its applicability to describing the states (frequency select, fine tuning, etc.) of the ATAS system was noted.

In the examples and exercises used during the week, new programming techniques and concepts were introduced. These included various data structures (linked lists, trees), searching and sorting algorithms, etc. Students would later select among these in designing the data structure and lookup algorithm for the ATAS coarse tune function.

Week 4—documentation, testing, and debugging; UNIX

This week involved two course units—approximately half was devoted to documentation, testing, and debugging, and half to an introduction to the UNIX system. The lecture material concentrated on the requirements of MIL-STD-1679 that had not been covered in Week 1. In addition to the other required project documentation, the week included lectures on project management requirements, configuration management, and software quality assurance. Students developed both a System Operator's Manual and a Test Plan for the ATAS system. Both documents were produced by developing an outline through class interaction and then assigning a section to each student. Though this resulted in rather unusual first drafts, it is comparable to the way documents are often produced in the real world. It was also the only realistic way to have the class produce actual documentation, as individuals could not be expected to produce entire documents. The students also developed a Software Trouble Report form to be used during ATAS development.

The testing and debugging material included both a discussion of current approaches to software development testing and debugging, as well as presentation of MIL-STD-1679 requirements for formal acceptance testing. It included a detailed discussion of testing under simulation, as this was the way ATAS (like many real microprocessor-based systems) would be tested.

The UNIX classes and labs concentrated on the basics of the UNIX file system and text editor, but provided an overview of more sophisticated features such as parameterized command files. The students used the UNIX editor and runoff (nroff) to produce their ATAS System Operator's Manual and Test Plan documents. Each typed in his section, and the instructor provided the title page and ran off the final document.

Week 5—reviews and walkthroughs; CCLF

Week 5 was also shared with the Host Computer Facilities unit. The Reviews and Walkthroughs unit introduced various concepts relating to egoless programming and peer review, including code reading and structured walkthroughs. The objectives, mechanics, and behavioral aspects of these were discussed, and exercises in each were conducted. Walkthroughs were held for both ATAS documents produced in Week 4, and each resulted in action items that were subsequently performed to produce revised documents. In fact, review of the System Operator's Manual led to a question about the correctness of the PPS in regard to the meaning of one of the status lamps. The instructor pointed out that it was necessary to go back to system engineering for a decision, and the class put in a call to the "system engineer" back at SofTech.

The lab sessions this week presented the Change Control Library Facility (CCLF), which would be used to maintain PDL and program components for the ATAS project. Students learned to create and manipulate Software Configuration Trees, the basic structure used to organize and control development within the CCLF framework.

Week 6—detailed design lab

This was a lab week with no formal instruction, devoted to development of detailed PDL design for the ATAS project. The instructor staff developed a Program Design Specification (PDS) documenting the top-level design to be followed. The PDS decomposes the system into twelve subprograms and two libraries of common subroutines (math routines and I/O routines), and fully documents their functions and interfaces.

The class was divided into two teams, each of which would develop a complete ATAS system. This was done for the following reasons:

- It seemed to be (and proved to be) about the right amount of work per person.
- It made the number of interfaces more manageable.
- Students had an opportunity for more self-management and real teamwork, as the instructor could not always be involved.

Teams elected chief programmers, who would generally be responsible for keeping the team organized (the instructor distributed "chief programmer job descriptions"). Individual assignments were determined by the teams, with input from the instructor as to the difficulty of the various subprograms.

The teams then separated to begin design. They were encouraged to produce early written design kits for review by other members of their team, and by the instructor. Designs were then refined based on comments. After this, each team held a walkthrough. The chief programmer served as the moderator, and the instructor recorded action items but otherwise generally attempted to maintain a low profile. After the walkthroughs, designs were updated again.

Throughout the week, students used the CCLF to create Configuration Trees of their PDL, eventually producing a tree with their final system PDL. The tree would then be
expanded to include parallel nodes for source, object, etc. during Module 4.

This week was particularly well-received by the students, and provided an excellent illustration of the values of thorough early design and peer interaction. With little real help from the instructor (other than nudges in the right direction) students went from faulty individual designs that they had little confidence in, to complete system designs that they were very sure of and that were in fact virtually faultless.

Week 7—microprocessor architecture and instruction set

Week 7 began the introduction of the target microprocessor, the Intel 8080. It combined a detailed presentation of the microprocessor's register and bus structure and its instruction set with an introduction to the microprocessor development facilities. As soon as the assembly language was introduced, students began using the assembler, link editor, and loader to run simple exercises.

TheXAS8 assembler does not process conventional Intel 8080 assembly language—it recognizes a more expression-oriented instruction syntax and provides the control structures needed for structured programming, such as IF-THEN-ELSE and DO WHILE. (The assembler generates appropriate tests and jumps.) Thus the students were taught to develop structured assembly language programs using the same principles they had learned with Pascal.

Week 8—advanced coding techniques

A major topic in Week 8 was mapping of Pascal (PDL) constructs to 8080 assembly language. This is particularly relevant in considering the mapping of more complex data structures and techniques for accessing them. This week also introduced coding "tricks" and time and space saving techniques, along with a discussion of when tricks should be avoided because of an adverse effect on program understandability.

Another subject addressed in Week 8 was the need for coding conventions in system development. Conventions for the ATAS project (e.g., subprogram linkage, register usage) were established and documented.

During Week 8 students began to use the lab sessions to convert their ATAS PDL to assembly language, mostly concentrating on getting the code to assemble correctly rather than on executing it.

Week 9—microprocessor lab exercise

This week was devoted to completion of the ATAS project. Students completed coding, and code-read one another's subprograms. They then used the instruction simulator to perform unit and integration testing in accordance with the test plan they developed in Week 4. During this week teams were encouraged to work with one another so that a team that had developed a working version of a particular subprogram could help a team that had not. This provided an opportunity to point out the benefit of defining subprogram interfaces clearly and then observing the specification, as one person's version of a subprogram could be directly substituted for the other's.

A final output of this week was a Program Description Document for each team's design.

CONCLUSIONS

As the course is just concluding as this paper is being written, it is not yet possible to evaluate the performance of the students as software engineers. However, our experiences in presenting the course have made us very optimistic about this. A substantial amount of material was covered and absorbed by the students. In addition, the emphasis on military standards, on real project experience, and on development tools actually used at ITTDCD, has equipped these students with skills that most new computer science graduates lack.

ITTDCD intends to follow up the course by observing and evaluating its success at producing effective software engineers. If the outcome is positive, the course will be presented to additional groups of students. As indicated previously, the course evaluation report prepared by Softech will recommend possible modifications based on the initial experience.

The modular nature of the course makes some modifications quite straightforward. For example:

- An introductory module could be added for students with no prior programming background.
- Most of Module 3 could be used stand-alone to teach modern programming practices to students who are already programmers.
- A different course project could be substituted.
- Another military standard could be substituted.
- A different target computer could be substituted.
- A different host development facility could be substituted.

Thus the present course material forms a baseline that can easily be tailored to fit ITTDCD's changing needs in the future.

ACKNOWLEDGEMENT

The course described in this paper was conceived and designed by G. Sampson and Gen. J. Robbins of ITTDCD, and by Dr. R. S. Eanes, Dr. C. McGowan, Dr. L. Weissman, Dr. S. Shrier, and R. Thall of Softech.

REFERENCES


