Software Engineering Principles
July 11-15, 1983
with Dr. David L. Parnas

Although the construction of a useful piece of software is a very complex task, the complexity can be mastered if a few fundamental principles are rigorously applied. The software designs that follow from these very simple principles are often very different from those that result from intuitive approaches.

To define those principles and their implications, the principles will first be introduced in lectures. Students will then apply the principle to practical software design problems. After they have done their best a discussion period will allow participants to compare their solutions and discuss the implications of the design concept. Students will return to their jobs understanding both the theoretical concept and how it can be applied in practice.

Topics include:
- the use of precise and complete specifications in the design of software;
- the application of abstraction/information-hiding to the solution of complex problems;
- the design of abstract interfaces in software;
- the design and synchronization of cooperating sequential processes;
- hierarchical structures in software;
- the design of program families;
- design by means of documentation.

For Whom: Programmers and system managers with some experience in the production of software products. Knowledge of at least two programming languages, one of which must be an algorithmic language such as ALGOL-60, Pascal, FORTRAN, Basic, Jovial, CMS-2, SPL, HAL.

Instructor
Dr. David L. Parnas, currently at the University of Victoria in British Columbia, has been studying the problems of software design for more than 20 years. Many of the principles he has identified and published have been incorporated in the design of modern languages and have been applied to the design of practical software. Working together with the Naval Research Laboratory, IBM Federal Systems Division, Bell Laboratories and others, he has been closely involved with the application of these principles to practical problems.

Tuition: $850.00 Canadian funds

Software Design: Verification and Testing
July 25 - 29, 1983
Dr. Harlan D. Mills and Dr. Richard G. Hamlet

In any human activity we progress in understanding by reducing complexity to simplicity. In programming methodology, the story is quite familiar. As a result of work by many researchers, we now have systematic procedures for program design. As a result of work in functional semantics, we have systematic procedures for convincing ourselves and others of the correctness of a program.

As a result, we now understand many deep simplicities that were formerly shrouded in the mystery and complexity of programming lore. This course will require more mathematical maturity than typical short courses. The mathematics, per se, is not difficult. The stretch will occur in learning to apply the reasoning used in mathematics to programming. Such reasoning will enable programmers to keep their programming under better intellectual control.

It will simplify the elimination of bugs and getting programs to run. There will be extensive discussion of the application of this approach to the problem of testing programs.

Instructors
Dr. Harlan Mills is IBM Fellow at the Federal Systems Division of IBM. He is well known for numerous papers on mathematics and for his work on structured programming. Under his direction, Structured Programming and its mathematical foundations have been taught to all the programmers in IBM/FSD and his approach has become the official standard of that Division. Dr. Mills has been deeply involved with applying mathematical techniques to solve the problems of the real world. Dr. Richard Hamlet is Professor at the University of Maryland.

Tuition: $850.00 Canadian funds