Software Engineering: A Motivation

by Dan Ghica

Anyone who has ever taken a computer science course knows how to go about writing a computer program. When you are given a problem you first develop a solution concept. Then you translate this solution into a language the computer understands and type it in as a program. Then you compile and run the program. Most of the time it does not really work as well as you expected, so you start examining it, debugging it, and changing it bit by bit. You eventually make it work. The result of this interesting process, driven by imagination and creativity and sustained by pizza and coffee, is a piece of software. You may run the program a few times to demonstrate it to your professor or classmates. But chances are you will forget about it a few weeks later and never use it again.

So you know what it takes to write a program; it is a fairly straight forward process. But let us suppose the problem you have to solve is a thousand or a million times more complex. So complex that one person can no longer handle it and a team of hundreds of people have to develop the solution concept. This solution will need to be translated into a programming language, and will result in a program of tens of millions of lines of code. This program will not be thrown away in a few weeks, but will stay in use and require maintenance for ten or twenty years or even more. Also let us suppose that human lives, or the functions of an economy or democracy depend on the correct behavior of this program. If we make all these crazy assumptions then we move from the classroom into real life --- where computers not only count the numbers of words in a file or draw nice circles on a screen, but manage airplane traffic, control financial transactions, or count the votes after an election.

Pizza and coffee are no longer the fuel to keep such an extraordinary activity alive, although they are still essential ingredients. Significant economic resources, severe constraints, and strict deadlines now come into play, demanding from the software producer discipline and efficiency. The real life software creation process is so complex that creativity and imagination are no longer enough. They are important, as important or even more so than before, but something else is needed to make a coherent software production mechanism possible. What is necessary is a body of methods for efficiently producing quality software. *Software engineering* is the discipline that studies these methods.

Because the numbers of problems that need to be solved and processes that need to be defined are so large, software engineering is subdivided into several narrower disciplines. These disciplines include *requirements specifications methodology* (how to describe the problem and the solution), *design methodology* (how to construct the program), *validation and verification techniques* (how to determine whether a program is correct or not), *software process management* (how to put everything together),

and many more.

Being so young and lacking the unity and the methodological coherence of established engineering disciplines, software engineering is not always recognized as a genuine form of engineering. But given its methodological and practical nature as well as its social and economic importance, it is only a matter of time until this new discipline will acquire the maturity necessary to be fully recognized as a true profession.