Sidebox: Just what the heck is software engineering?

Wrestling with this question has sent me looking for our principals, I've gone looking for our roots, I've searching the shelves of engineering books looking for "Engineering 101" so I could relate engineering principals to software. I found books on chemical engineering, books on production engineering, electrical engineering, mechanical engineering, marine engineering but no "Engineering for dummies."

Like so many things in life you sometimes find these things where you least expect them, so what a surprise to find under my nose "Some Software Engineering principals" by Dave Parnas. Reprinted in *Software Fundamentals* (Hoffman, 2001) the introduction summarises the key idea as: "the connections between program parts are the assumptions that the parts make about each other."

As principles go, interfaces and modules have stood the test of time, but it needs re-emphasising and re-casting in the light of new developments. The principle itself is developing, it is not enough to consider just the interface, we must also look at the way it connects and interacts with other modules.

In a thread entitled "The quality without a name" on the Accu-General mailing list (May 2002), Kevlin Henney gave one of his criteria for judging software quality: "Spacing: This refers to boundaries, separations, connections, etc. This applies at the smallest level, e.g. indentation of source, as well as the largest, e.g. how interfaces are used to partition component "neighbourhoods" in a large system." Similar ideas where also proposed by Jon Jagger under the name "Connections" and myself with the name "Boundaries" – sounds like we're back to Parnas's thoughts of 1978.

Software engineering is a young profession and strong principals need time to emerge. Unfortunately in the 50 odd years that computer programming has existed it has been amazingly successful, one of our problems now is that we are in danger of becoming insular. In my search for "Engineering 101" I came to realise how much other fields have to offer us. Mathematics and architecture have long been seen as disciplines with something to offer software engineering, but so too do product and industrial design, management, psychology and countless disciplines I haven't thought of.

For me, economics is a discipline all software engineers should study. I'll give two reasons for this bold statement. Firstly, software is a product of the modern economic world: there have never been riots over software shortages, nobody has ever been nailed to a cross for their belief in brace conventions, and the US constitution has nothing to say about the right to own software. In reality only software that has a socio-economic role is important.

Secondly, the two disciplines have a lot in common. They are both disciplines of the mind, which deal with ideas but which have a very real effect on people's lives, they are both open to new ideas and are home to experienced practitioners who sometimes fundamentally disagree.

Anyone who has studied economics will have seen the question "What is economics?" for me the answer has always been given by John Maynard Keynes:

> "The theory of economics does not furnish a body of settled conclusions immediately applicable to policy. It is a method rather than a doctrine, an apparatus of the mind, a technique of thinking which helps its possessor to draw correct conclusions."

To the question "What is software engineering?" I would like to answer:

> "The discipline of software engineering does not furnish a body of settled conclusions immediately applicable to program development. It is a set of methods rather than a doctrine, an apparatus of the mind, a set of techniques of thinking which helps the possessor develop computer programs."

Bibliography

Hoffman, D., and Weiss, D., eds., 2001; Software Fundamentals : Collected Papers of David L. Parnas, Addison Wesley, 2001.