Software Engineering Economics

The dictionary defines "economics" as "a social science concerned chiefly with description and analysis of the production, distribution, and consumption of goods and services."

Here is another definition of economics that is more helpful in explaining how economics relates to software engineering.

"Economics is the study of how people make decisions in resource-limited situations."

This definition of economics fits the major branches of classical economics very well.

The two main branches of economics are:

1. Macroeconomics
2. Microeconomics

1. Macroeconomics: is the study of how people make decisions in resource-limited situations on a national or global scale. It deals with the effects of decisions that national leaders make on such issues as tax rates, interest rates, and foreign and trade policy.

2. Microeconomics: is the study of how people make decisions in resource-limited situations on a more personal scale. It deals with the decisions that individuals and organizations make on such issues as how much insurance to buy, which word processor to buy, or what prices to charge for their products or services.

Economics and Software Engineering

If we look at the discipline of software engineering, we see that the microeconomics branch of economics deals more with the types of decisions we need to make as software engineers or managers.

Clearly, we deal with limited resources. There is never enough time or money to cover all the good features we would like to put into our software products. And even in these days of cheap hardware and virtual memory, our more significant software products must always operate within a world of limited computer power and main memory. If you have been in the software engineering field for any length of time, you can think of a number of decision situations in which you had to determine some key software product feature as a function of some limiting critical resource.

Throughout the software life cycle, there are many decision situations involving limited resources in which software engineering economics techniques provide useful assistance. To provide a feel for the nature of these economic decision issues, an example is given below for each of the major phases in the software life cycle.

• Feasibility Phase. How much should we invest in information system analyses (user questionnaires and interviews, current-system analysis, workload characterizations, simulations, scenarios, prototypes) in order to converge on an appropriate definition and concept of operation for the system we plan to implement?

• Plans and Requirements Phase. How rigorously should we specify requirements? How much should we invest in requirements validation activities (automated completeness, consistency, and traceability checks, analytic models, simulations, prototypes) before proceeding to design and develop a software system?

• Product Design Phase. Should we organize the software to make it possible to use a complex piece of existing software that generally but not completely meets our requirements?

• Programming Phase. Given a choice between three data storage and retrieval schemes that are primarily execution-time efficient, storage efficient, and easy to modify, respectively, which of these should we choose to implement?

• Integration and Test Phase. How much testing and formal verification should we perform on a product before releasing it to users?

• Maintenance Phase. Given an extensive list of suggested product improvements, which ones should we implement first?

• Phase-out. Given an aging, hard-to-modify software product, should we replace it with a new product, restructure it, or leave it alone?

Source URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05010193