Software paradigms

The word "paradigm" is in current vogue in software circles but is rapidly becoming useless because it does not have a well-agreed-upon (let alone precise) meaning. Almost any dictionary will give "model, pattern, or example" as one definition. I use the word in that context when I say that a paradigm is a model or approach employed in solving a problem.

Software paradigms either induce or are induced by a class of languages, the programming environments that these languages are supported in, and the software engineering discipline one uses to produce systems within these environments. Each class places different requirements on its associated programming environment. This results in a tool set and a style of usage that is natural with the tool set.

Examples of language classes include procedural, nonprocedural, functional, visual, logic programming, and object-oriented languages. Software engineering refers to the set of rules and structured thinking that one applies within the environment to produce software systems or components.

It is not clear that the paradigms in use today are either the right match to current and imminent technology or the right match to current and imminent user needs. We must develop new programming paradigms that lie beyond the ones currently studied and used in some degree of detail.

New hardware technologies often force the development of new paradigms, just as all-points-addressable graphics displays generated icon-based environments. Conversely, some researchers argue that new paradigms should drive the creation of new technology solutions. For example, functional languages may lead to specific non-von Neumann architectures.

These languages may require both new software development environments and software engineering disciplines when used to design and implement complex software. The languages, environments, and software engineering disciplines associated with alternative single or multiple paradigms will undoubtedly require substantially new technology for high-performance implementations of the systems based on them.

There are exciting times ahead as we begin to make even modest headway in these issues. This special issue of IEEE Software provides several articles that should aid the reader in exploring a variety of topics in this interesting area.

Bruce D. Shriver
Editor-in-Chief