Editor in Chief: Steve McConnell ■ Construx Software ■ software@construx.com

What's in a Name?

Steve McConnell

One consequence of the youthfulness of software engineering is that we have not standardized our terminology. This might seem like a small matter of primarily academic interest, but the lack of defined terms has significant and unexpected consequences. The number of ambiguous terms in software engineering is practically limitless, and in this column I will focus on only a few: "requirements," "prototype," "specification," "architecture," and "analysis."

Term Limits

Problems with terminology begin as soon as a project gets underway. Suppose your customer asks you to write a "requirements" document. What do you write? I have found that different organizations might call any of the following a "requirements document":

1. A half-page software product vision
2. A two-page list of key features
3. A 50-page list of detailed end-user-level requirements
4. A 250-page exhaustive listing of every visual element on every screen, input-field-by-input-field descriptions of all possible input conditions, all possible system state changes, detailed descriptions of every persistent data element, and so on

For my purposes, I have gravitated toward calling Level 1 requirements a "product vision," Level 2 a "feature list," Level 3 a "functional-requirements document," and Level 4 a "functional specification," but my usage is by no means standard.

One technique used to pin down requirements is prototyping, which brings on a second wave of confusing terminology. Some people use "prototype" to refer to a functioning program that will evolve into a working program but is less robust than the fully functional program will be (I call this an "evolutionary prototype"). Other people use "prototype" to refer to a nonfunctional mock-up that illustrates how the fully functional system will look or operate but is not, itself, intended to be evolved into the working system (I call this a "throwaway prototype"). Still others use the term "prototype" to refer to low-quality software that is at an early prerelease stage of development.

At some point, most project teams create a "specification" or "spec," which muddies the waters still further. "Specification" refers most commonly to detailed functional requirements—the Level 4 requirements mentioned earlier. Almost as often, "spec" refers to what I have called Level 2 or Level 3 requirements. Some people use "spec" to refer to an architecture document, detailed design, or other work product. One company used the word "specification" to refer to a document that contained every piece of information relevant to a project—and the document didn't have any requirements information at all (thanks to Karl Wiegers for this example).

"Architecture" has varying meanings, too. Some people use "architecture" to refer to user interface design, especially to the flow among the different elements of the user interface that will be visible to the software's users, but not to technical implementation work. Others use "architecture" to mean partitioning a system into subsystems and defining the interfaces between them. Still others mean anything related to the functional design of the software-to-technical implementation work.

Finally, the meaning of "analysis" varies from one organization to the next. Many use it to refer to the early requirements activity of analyzing user needs. Others use it interchangeably with requirements elicitation and documentation—essentially synonymous with "requirements engineering." Others use "analysis" to refer to the activities that bridge requirements gathering to design work. Once again, no meaning is common enough to allow any single meaning to be declared an obvious standard.

Expensive Consequences

These differences in the ways common software engineering terms are used can have major consequences. I participated as an expert witness in a multimillion-dollar lawsuit in which a company sued its former vice president of technology for nonperformance of work duties. The company alleged that it had assigned the VP to develop a prototype and architecture, and that he had failed to do so. The VP was flabbergasted that the company could baldly assert that he had not created these work products when in fact the work products were available for all to see.

It turned out that the VP had developed an evolutionary user interface prototype and demonstrated it to the company's senior management. Senior management was enthusiastic about the prototype and encouraged the VP to "finish" the prototype. Senior management assumed that the prototype was a throwaway prototype, and remaining details could be added quickly as a precursor to the main implementation effort. The VP assumed that "finishing the prototype" meant evolving it to a level of robustness and reliability at which it could be released for commercial use. "Finishing the prototype" meant "finishing the product." This led to a critical miscommunication in which senior management thought the prototyping work could be finished in a matter of days or at most weeks, while the VP thought the prototype development would take many months, possibly as long as a year.

At issue in the same lawsuit, senior management had assigned the VP to create a "product architecture." The VP's prototype showed many details of the user interface design, flow among user interface elements, and so on. He considered this to be a full-fledged product architecture. The company's senior management expected the VP to develop a functional architecture that defined the software's decomposition into subsystems, class hierarchies, network architecture, and so on. As the company insisted that the VP work harder on the "architecture," the VP spent more time fleshing out details of the prototype. The more time the VP spent fleshing out the prototype, the more frustrated the senior management became that the prototype was taking so much longer than they expected, and the more frustrated they were that the VP refused to create an architecture.

Coming to Terms

Not every misunderstanding of software engineering terms ends up in court, but we as a profession face some serious implications of these undefined terms. The IEEE Computer Society is creating a test that will be used to award "Certified Software Engineering Professional"

IEEE Software

Coming in the Next Issue:

Recent Updates in Estimation
Barry Boehm and Richard Fairley, guest editors

The Personal Software Process
Watts Humphrey, guest editor

credentials. It's challenging to develop a multiple-choice question like "What should go into a requirements document?" (or architecture, prototype, spec, or analysis) when common usage varies so much.

The legal field has a useful notion called a "term of art." To lawyers, a term of art is a word that has been analyzed and discussed so much in court cases and statutes that it has a precisely defined meaning, a meaning that might be quite different from the common language meaning. Software engineering has taken a step in the right direction with IEEE Std 610.12, the "IEEE Standard Glossary of Software Engineering Terminology." (The sidebar "A Few Standard Definitions" defines the terms used in this column.) I hope that as the years pass, common usage of these critical terms will gravitate to specific, precise meanings and that IEEE Std 610.12 and common usage will be brought into alignment.

A Few Standard Definitions

The closest we have to a vocabulary standard are the terms defined in IEEE Std 610.12. Here are its definitions of the terms used in this column:

Architecture. The organizational structure of a system or component.

Prototype. A preliminary type, form, or instance of a system that serves as a model for later stages or for the final, complete version of the system.

Requirement. (1) A condition or capability needed by a user to solve a problem or achieve an objective. (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents. (3) A documented representation of a condition or capability as in (1) or (2).

Requirements Analysis. (1) The process of studying user needs to arrive at a definition of system, hardware, or software requirements. (2) The process of studying and refining system, hardware, or software requirements.

Specification. A document that specifies in a complete, precise, verifiable manner the requirements, design, behavior, or other characteristics of a system or component and, often, the procedures for determining whether these provisions have been satisfied.

DEPARTMENT EDITORS

Bookshelf: Warren Keuffel, wkeuffel@computer.org
Culture at Work: Karen Mackey, Cisco Systems, kmackey@best.com
Loyal Opposition: Robert Glass, Computing Trends, rglass@indiana.edu
Manager: Don Reifer, Reifer Consultants, dreifer@sprintmail.com
Quality Time: Jeffrey Voas, Reliable Software Technologies Corp., jmvoas@rstcorp.com
Soapbox: Tomoo Matsubara, Matsubara Consulting, matsu@computer.org
Softlaw: Larry Graham, Black, Lowe, and Graham, graham@blacklaw.com

STAFF

Group Managing Editor
Dick Price
Managing Editor
Dale C. Strok
dstrok@computer.org
Associate Editor
Dennis Taylor
Features Editor
Crystal Chweh
Staff Editor
Jenny Ferrero
Assistant Editors
Cheryl Baltes and **Shani Murray**
Magazine Assistants
Dawn Craig and Angela Williams
software@computer.org
Art Director
Toni Van Buskirk
Cover Illustration
Dirk Hagner
Technical Illustrator
Alex Torres
Production Artist
Carmen Flores-Garvey
Acting Executive Director
Anne Marie Kelly
Publisher
Angela Burgess
Membership/Circulation Marketing Manager
Georgann Carter
Advertising Assistant
Debbie Sims

CONTRIBUTING EDITORS

Nancy Mead, Ware Myers, Gil Shif, Pradip Srimani

Editorial: All submissions are subject to editing for clarity, style, and space. Unless otherwise stated, bylined articles and departments, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *IEEE Software* does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society.

To Submit: Send 2 electronic versions (1 word-processed and 1 postscript or PDF) of articles to Magazine Assistant, *IEEE Software*, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1314; software@computer.org. Articles must be original and not exceed 5,400 words including figures and tables, which count for 200 words each.

IEEE Software