Ambiguity and What to Do about It

Ben Kovitz
bkovitz@acm.org

Abstract

Abstract: A major concern of most software customers, managers, and requirements engineers is to remove ambiguity in communication of requirements and specifications. The most obvious solution is to try to anticipate all possible misunderstandings and write the requirements perfectly precisely. In practice, this doesn't work.

This talk explains why it doesn't work, and offers easy, inexpensive methods for removing ambiguity—methods that anyone can do. The fundamental principle is to add redundancy, especially redundancy relating to context. High-bandwidth, informal communication is always a necessary supplement to formal, mathematical expressions. As software development is in essence the creation of formal, executable descriptions for the informal domains where our intents lie, we explore many ways to break up this process into small stages, allowing programmers and customers to detect ambiguity through real-world feedback.

Summary of talk

Ambiguity is a relation between a description and reality in which the distinctions in the description fail to guide you when you meet the reality.

"Multiple choice" ambiguity: You have a known set of possibilities and the description allows for more than one.

Examples: "Response time must be fast." How fast?

"User selects A or B and C or D."

"Yawning void" ambiguity: Relevant things in the world have no place in your conceptual framework. Reality undermines what you thought your description meant, like djinni stories.

Examples: "Display the name of the road." Palomar Airport Road or County Route 12?

"Bill travel expenses to the employee's organization." The organization that the employee is a member of, or the organization that is paying for the trip?

To detect multiple-choice ambiguity, look at the description. Repair it by *ruling out* known alternatives.

Two ways to detect yawning-void ambiguity: notice your own feeling of confusion, notice surprises from reality. Repair it by *creating* new concepts (new distinctions) directly from relationships found within the reality.

Multiple-choice ambiguity is syntactic, and can be detected automatically. Yawning-void ambiguity is semantic; there is no way to systematically ensure that you have ruled it out.

A tempting solution is to try to write perfectly precisely, in a way that can be interpreted correctly by mechanically following *a priori* rules of interpretation. This fails for two main reasons. (1) It assumes a hostile, infinitely intelligent, and infinitely patient reader, and real human beings are not like that. (2) Reality is more complex than you can anticipate.

Yawning-void ambiguity dominates in real-world software development. We always start with human ideas, which can be vague or contain gaps or contradictions. The vast majority of software development consists of working out the inevitable ambiguities in those ideas.

A common denominator among a great many effective ways to counter ambiguity is to add redundancy relating to context.

Context is everything outside your description and its subject matter, that relates to it in any way.

- Examples.
- Rationale: "Why?" Domain knowledge: describe the problem. Etymology.
- Structure and emphasis.
- Confirmed implications.

Context gives meaning to descriptions by anchoring them in reality. Context puts the reader's intelligence to work for you in resolving ambiguity.

Requirements communication, to be effective, must work human-to-human. Formal notations, with their total lack of ambiguity, are the end point of software development, not the start.