



Cherishing Ambiguity

Neil Maiden

“WARNING: AMBIGUITY IS an enemy of the requirements analyst.” That’s what we’re told. You know the mantra: make your requirements precise; remove ambiguous statements. But most requirements are still written in ambiguous, natural language, and we’re still able to develop working software.

Ambiguity in specifications has never been fashionable. In this column, I seek to make the case for ambiguity in requirements work. Moreover, I argue that, in some cases, it can even be beneficial.

Ambiguous Requirements

I know that this might be construed as ironic, but let me provide you with some definitions of ambiguity to define what I mean by it (at least for this column). Francis Chantree described am-

Meyer defined an ambiguous specification as one that makes it possible to interpret a problem feature in at least two different ways.²

It has become a truism that it’s bad to specify requirements that are open to different interpretations. Meyer, in his case for more formal specification languages, makes ambiguity one of the seven deadly sins in requirements specifications, whereas Chantree treats requirements ambiguity as noxious, even damaging, because it can lead to misunderstandings.

Now, before I receive a bag of letters from advocates of formal specification techniques, let me clarify: I am not arguing against precision in requirements. There are many reasons to seek precision, especially in certain families of products and systems.

tem for pedestrian traffic lights.³ The specified features are required to reconfigure a set of traffic lights to support school crossings:

1. The warden control feature shall allow a school crossing guard (warden) to extend a nominated intergreen period (time that the traffic light is green) by means of an input device (which can be remote from the controller) without access to any other controller functions.
2. The input device shall provide a nonlatching input (for example, a push button or biased key) and must be mounted or protected so as to prevent unauthorized use.

I’m sure that most readers can understand what the stakeholders probably wanted from the controller. But consider the ambiguities—how people could interpret each in two or more different ways:

- The school crossing guard role could be a new position, one created with each installation of the system, or one fulfilled by responsible existing roles such as schoolteachers.
- The input device could be bespoke and delivered with the traffic light system, or it could be a separate existing device.
- The feature to allow the crossing guard to extend an intergreen period could also enable the guard to reduce that period.
- The input device could be mounted to the traffic light system or to some other secure structure such as a school building.

What I’m arguing for is a more tolerant view of ambiguity. Let’s not be too quick to remove it.

biguity in a requirements specification as when something can be interpreted in more than one way.¹ Bertrand

What I’m arguing for is a more tolerant view of ambiguity. Let’s not be too quick to remove it during our requirements processes, because it could be advantageous. Consider an example.

Requirements on a Traffic Light Controller System

The following requirements are taken from a specification of a controller sys-



See www.computer.org/software-multimedia for multimedia content related to this article.

These different interpretations and other ambiguities not listed here could make it difficult to deliver the system that stakeholders really wanted. But then again, each of these ambiguities could also trigger the exploration of new requirements. Consider the following examples:

- We might rethink the role of the school crossing guard in the socio-technical system (for example, the role could be allocated dynamically to different existing positions). So how might the new system be interacted with, and what new requirements might then emerge?
- We might rethink the means by which the crossing guard can interact with the system. For example, teachers could use a secure app on mobile devices to control the system, or traffic data from another source could be used to infer each optimum intergreen period.
- We might rethink the control of the traffic light periods to extend as well as reduce the intergreen periods in more flexible ways. For example, the crossing guard could also directly control the traffic light changes in response to observations about local traffic.
- We might rescope the traffic light system to be part of the infrastructure of the particular school that implements the system. For example, we could specify door sensors that detect school pupils leaving the school, which in turn changes the intergreen period.

All of these examples are, I believe, valid in a requirements process. They can lead to more effective system scoping, greater innovation, the discovery of new requirements, and, eventually, increased requirements precision by eliciting answers to the questions that ambiguities trigger.

Ambiguity Is Inevitable

We can't escape ambiguity. Understanding is a cognitive task undertaken by individual human beings. How each individual understands a requirement will depend on his or her domain knowledge, past training, and current skill levels—even when the requirements are specified in a formal language. The human in the loop makes requirements ambiguity almost inevitable.


So, rather than try and remove ambiguity from requirements work, perhaps we should think about when to encourage or discourage it. I'm reminded of the distinction between divergence and convergence in creative processes—there are times when you want to generate creative ideas by exploring possible solutions, and there are times when you want to hone in on one emerging solution. It can be the same with ambiguity. There are times when you want to discover new requirements from ambiguous ones, and there are times when you want to precisely specify the system to be implemented.

That said, requirements work won't necessarily benefit from just any old form of ambiguity. I suspect that requirements that are too ambiguous won't provide useful triggers to drive forward requirements work. For example, an ambiguous requirement stating "the system shall provide a nonlatching input device" is less likely to enable the discovery of new ideas and requirements than requirements 1 and 2 from the traffic light example because the ambiguous requirement lacks sufficiently detailed triggers with which to ask the right questions.

Application

One striking example of ambiguity in action is in agile development. At the start of a project, requirements are deliberately underspecified and therefore ambiguous. During the project, a more

precise understanding of requirements emerges from different forms of communication and collaboration between different stakeholders. The flexibility of the process lets the team exploit ambiguities and remove them at the right times. There are lessons we can learn from this.

Ambiguity in requirements isn't always a bad thing. In the right hands, it can be positively useful. Think about this the next time you're staring at a requirement open to different interpretations. 

References

1. F. Chantree et al., "Identifying Nocuous Ambiguities in Natural Language Requirements," *Proc. 14th IEEE Int'l Requirements Eng. Conf.*, IEEE, 2006, pp. 59–68.
2. B. Meyer, "On Formalism in Specifications," *IEEE Software*, vol. 2, no. 1, 1985, pp. 6–26.
3. *Specification for Traffic Signal Controller*, tech. report TR-2500, UK Highways Agency, 2005; www.ukroads.org/webfiles/TR2500A%20.pdf.

NEIL MAIDEN is professor of systems engineering and head of the Centre for HCI Design at City University London. Contact him at n.a.m.maiden@city.ac.uk.

**IEEE
Software**
FIND US ON
**FACEBOOK
& TWITTER!**

[facebook.com/
ieeesoftware](https://facebook.com/ieeesoftware)

[twitter.com/
ieeesoftware](https://twitter.com/ieeesoftware)