Ambiguity in Natural Language Requirements Documents
(Extended Abstract)

Daniel M. Berry
Cheriton School of Computer Science, University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
dberry@uwaterloo.ca

1 Introduction

This paper is an extended abstract of an invited talk at the workshop that I put together using material from other talks and from papers that I and colleagues have written. The purposes of this extended abstract are to summarize the talk and to allow the reader to find the source materials for the talk directly.

2 Natural Language Is Key in Requirements Engineering

An overwhelming majority of requirements specifications (RSs) are written in natural language (NL). Virtually every initial conception for a system is written in NL. Virtually every request for proposal (RFP) is written in NL [1]. However, we all know that NL is so ambiguous, and so inherently so. No wonder RSs are such messes.

There is an old tradeoff: A RS can be written (1) in a NL or (2) in a mathematics-based (MB) formal language (FL) 1. A NL has the disadvantage that it is inherently ambiguous, but the advantages (1) that there is always someone who can write with it and (2) that a RS written in it is always more or less understood by all stakeholders, albeit somewhat differently by each. A MB FL has the advantage that it is inherently unambiguous, but the disadvantages (1) that there is not always someone who can write a RS with it and (2) that a RS written with it is not understood by most stakeholders, although all that do understand it understand it the same.

A lot of research in requirements engineering (RE) is directed at solving the problem of ambiguous RSs by (1) convincing people to use MB FLs and (2) addressing the negatives of MB FLs, by making them more accessible [3], sometimes with the help of tools [4,5].

However, the reality is that there is no escaping NL RSs. Michael Jackson [6] reminds us that “Requirements engineering is where the informal meets the formal.” In order to write software, ideas, which are inherently informal, have to be converted

1 The term “mathematics-based” is used to distinguish the kinds of FLs I am referring to from other semi-formal notations, e.g., UML [2], that are often called “formal”.

somehow to code, which is inherently formal. There needs to be a transition from informal to formal somewhere along the way from ideas to code. That transition generally happens the first time ideas are written in an even informal notation, during RE. Therefore, NLs are inevitable, even if it is only for the initial conception.

Even if one moves immediately to FLs, the inherent ambiguity of the NL initial conception can strike as the transition is made. What the formalizer understands of the conception may be different from what the conceiver meant. The phenomenon of subconscious disambiguation strikes [7].

In subconscious disambiguation, the reader of an ambiguous phrase is not even aware that there is an interpretation other than the one that came first to his or her mind. The reader understands an interpretation and thinks that it is the only one. In fact, here is where it is most important to catch ambiguity: right up front, when the requirements analyst (RA) is getting raw information, be it goals, business rules, or requirements, from the clients and users. The RA must find each ambiguity and ask the clients and users what they mean with it. The flip side of subconscious disambiguation is subconscious ambiguity, the inadvertent introduction of ambiguity during writing by an author who believes that all readers will understand what he or she was thinking during the writing.

In a semi-formal language such as any of the UML notations, there are two sources of ambiguity. Ambiguity can still strike when going from the conception to a model, and the model itself is not uniguous [2].

Therefore, there is a group of researchers focusing on solving the problem of ambiguous RSs by trying to improve our writing, understanding, and processing of NLs.

### 3 Avoiding or Detecting Ambiguities

There are several approaches to avoiding the ambiguity of NLs:

1. Learn to write less ambiguously, avoiding those constructions that tend to create ambiguities [8,9,10,11,12].
2. Learn to detect ambiguity either manually [13,14,15], or with the help of tools [16,17,18,19,20,21,22]. Manual detection is helped by being able to recognize constructions that tend to create ambiguities. Someone who is aware of writing pitfalls can detect ambiguities manually more easily than someone who is not aware of the pitfalls.
3. Use a restricted NL which is inherently unambiguous [23,24,25] but may not be so natural.

### 4 Taxonomy and Definitions of Ambiguity

Figure 1 shows a taxonomy of the kinds of ambiguity that can be encountered in a NL RS [26]. Most of the tree is based on the traditional ambiguity-and-related-phenomena literature [e.g., 27]. The new portions of the tree are based on more recent literature about software-engineering ambiguity [28,29,30,31] and language-error ambiguity [32].

---

2 “Uniguous” means “not ambiguous”.