

Erroneous Requirements: A Linguistic Basis for Their Occurrence and an Approach to Their Reduction

Kimberly S. Hanks

ksh4q@cs.virginia.edu

John C. Knight

Department of Computer Science
University of Virginia
151 Engineer's Way
Charlottesville, VA 22904-4740
+1 44 434 982 2216
knight@cs.virginia.edu

Elisabeth A. Strunk

eas9d@cs.virginia.edu

Abstract

Poor communication of domain knowledge is implicated as a major threat to the validity of a requirements specification. It is hypothesized that careful examination of the mechanisms used for this communication can yield insight to motivate a response. We recognize that natural language is amenable to rigorous inspection and look to research results in linguistics to illuminate the nature of communicative breakdowns. From a better understanding of these breakdowns, we develop an approach, called the domain map, that through its structure and the process of its creation shows potential to aid in the systematic reduction of the incidence of communicative breakdowns during requirements. Essential domain semantics are better preserved in the conceptual model of the developers and thus the threat to validity of the model is reduced. Tool support enabling the creation and analysis of domain maps is also described.

1. Requirements is a Communication Problem

Software is often defective and defects cost; some cost more than others and result in loss of life, dollars, or the integrity of critical infrastructures. The approaches software engineering has developed to date in response to this problem have not been good enough, as evidenced by the continuing occurrence of disasters and other undesirable events resulting from defective software.

A particular stage of the development process, requirements, is implicated as the locus of introduction of most such defects. Lutz found that the majority of safety-critical

defects in the systems she studied derived from poor requirements [7], and the Air Force's Rome Laboratory found that the majority of *all* defects they observed derived from poor requirements [12]. Brooks has stated that "[t]he hardest single part of building a software system is deciding precisely what to build" [1].

This makes the issue of costly defects an issue of requirements validity more than anything else. The field has advanced techniques for verification; under appropriate circumstances, we are quite facile at proving that a program meets its specification. However, the real problem appears to remain in producing a valid specification to begin with.

Domain knowledge plays a special role in this problem, since a goal of requirements is domain knowledge transfer. This makes requirements a communication problem, as echoed by Hayhurst and Holloway [3]. Domain knowledge is the thing being communicated and domain experts and developers are the people doing the communicating. Without accurate conceptions of the real world semantics relevant to a system, developers are forced to rely on misunderstandings and invalid assumptions about the entities they model, often without realizing it. Propagation of these misunderstandings and invalid assumptions can lead to catastrophic failures. Lutz states "[i]t is not the internal complexity of a module but the complexity of the module's connection to its environment that yields the persistent, safety-related errors seen" [7]; to understand this connection, developers must acquire an understanding of relevant domain knowledge.

The problem, then, is to examine how this breakdown in communication of domain knowledge occurs, and, using insights gained in the process, to develop strategies to minimize the breakdown. To this end, we look to a field whose

primary object of study is communication. The goal of linguistics is to develop an understanding of the rules and heuristics by which the semantics conceived of by one person can be recreated in the mind of another. In particular, we will draw on a subfield called cognitive linguistics, and examine the domain knowledge communication problem from the standpoint of the theory of cognitive categories that cognitive linguists propose.

2. Cognitive Categories

To understand how semantics can be communicated, it is valuable to understand the form in which they are stored by any person. Results in cognitive linguistics and cognitive psychology show that the universe of semantics conceived of by any individual is organized into collections called categories, and that these categories have internal structure and abide by collective principles [8, 9, 11].

Categories are collections of entities experienced or imagined by a person that are judged by that person to be the same with regard to some dimension of reality. For example, we can discriminate among sparrows, chickens, and penguins, but for a certain slice of experience, they are all birds and thus instances of a *bird* category. They do not, however, have equal status within the category. Membership in cognitive categories is graded; some instances are more prototypical and others are less so, and some may be judged in or out of a category inconsistently between people and even by the same person as his experience changes. The topology of the categories in a given person's inventory is as idiosyncratic as his lifetime of experience; it derives from several factors, including the saliency and relative frequency of his observations of entities to be classified.

Among the collective principles obeyed by an individual's inventory of categories are *cognitive economy* and *hierarchical structure*. *Cognitive economy* is the ability of humans to associate with an entity a large number of attributes that might not be readily observable [8, 11]. This association is gained through the mapping of an entity to a category, and heuristically ascribing to the entity categorical attributes highly correlated with known instances of that category. These heuristics employ assumptions based on accumulated past experience, and have been tuned through evolution to be effective in everyday human communication.

However, communication of domain specific information between experts and non-experts is not the common case, and cognitive economy, which is a heuristic, is not suited to it. Since the attributes assumed by participants through the application of cognitive economy are idiosyncratic to their respective experiences, the potential for com-

municative breakdown exists whenever the categories in question have topologies that differ in relevant ways. If there are cues signaling the misalignment to either the sender or receiver, clarification can be undertaken. In the absence of such cues, a receiver interprets into a category a set of attributes that *do not match those intended by the sender*. He assumes that he does indeed understand the intended semantics when in fact he more likely has information that is erroneous and incomplete. Cognitive economy is useful in environments where interlocutors have had experience that renders their category inventories compatible. However, its reliance on assumption runs counter to the explication and precision necessary for the accurate communication of domain specific information from experts to developers with very different experience and knowledge bases.

The other principle we address, *hierarchical structure*, compounds the problem presented for domain knowledge communication by the heuristic application of cognitive economy. Further evidence from cognitive linguistics suggests that the categories we possess are organized into a hierarchy, and that a particular level of this hierarchy, labeled the *basic level*, has a special salience in our perception of the world [8, 9, 11]. This level is intermediate, it is neither a very general way to describe a thing nor very specific (for example *dog* rather than *mammal* or *retriever*), and its special role is evidenced by features particular to it in our acquisition and use of categories. The basic level is the level represented by the categories first acquired by children, the categories we first invoke when classifying a newly encountered entity, and the categories we use when introducing an entity into conversation. Importantly, analysis of empirical data has demonstrated a further property of the basic level that supports our intuition of its salience: *the basic level is that level of the hierarchy at which elements of any given category share the most features with each other and the fewest with members of other categories* [8, 9, 11]. This property is key in uncovering a stumbling block in the communication of domain knowledge.

We hypothesize that domain experts possess accumulated experience that allows them to associate many additional attributes with categories pertaining to their domain. These attributes provide more dimensions along which to collect and differentiate entities, resulting in certain of these categories bearing the above mentioned mark of the *basic level*: that level at which members of any given category share the most features with each other and the fewest with members of other categories.

This predicts that experts tend to see lower levels as basic, and therefore use them in ways that basic level categories are used. This usage translates to proportionally more opportunity for the miscommunication deriving from the application of cognitive economy, since the very cate-

gories that are likely to be more often and more severely misaligned between the interlocutors, that is, the ones that classify domain specific concepts for which the expert and the developer have varying experience, are also more likely to arise in these conversations. This occurs not just because the communication is *about* the concepts organized by these categories, but because the basic level status of these categories for the expert will result in a larger number of invocations.

The psychological heuristics of *cognitive economy* and *hierarchical structure*, which evolved to convey a volume of semantics efficiently and usefully in everyday communication, backfire in the context of domain knowledge communication precisely because of their exploitation of assumption. *It is not a part of human nature to get this right without serious and explicit intervention.*

3. A Solution Approach

Consider the case in which an expert and a developer are communicating regarding an entity and one or the other has in fact recognized that a breakdown is occurring as a result of a misalignment. This is not representative of the more dangerous situation of no breakdown being signaled, but motivates a strategy for preventing the breakdown from occurring in the first place. In the case where breakdown is recognized as it is happening, the usual course of action taken by the interlocutors is to execute clarification activities. These activities generally take the form of paraphrasing the offending term with another term or phrase for which the sender believes the receiver is likely to possess a more compatible category topology or topologies. If this paraphrase contains terms that also invoke misaligned categories, these terms can further be paraphrased, and so on. Comprehension is recursive; we comprehend a new idea when we can put it in terms of other ideas that we already comprehend. This insight provides a direction for dealing with the problem produced by reliance on assumption in communicating domain knowledge.

Our approach is to introduce into specification a highly structured mechanism called the *domain map*, along with a process for constructing the map during requirements acquisition and specification. The map stores definitions of domain-specific terms, and documents their recursive dependence on definitions of other terms for their comprehension. It is intended to provide a systematic and complete repository of relevant domain semantics built according to the principle of making the implicit explicit. Further, once constructed, it is to serve as the exclusive point of reference for such semantics, providing a consistent picture to developers.

Specifically, the domain map is to be constructed using

a starting point of some form of natural language requirements, for example, a written document, or recordings or notes from requirements interviews. This body provides a corpus representing an instance of the language used to talk about the domain in question and the constraints to be placed on it. In an iterative process, developers and experts cooperate to partition this corpus into terms identified respectively as *domain*, those that have domain-specific meaning, and *common*, those that are unlikely to invoke relevant differing assumptions between experts and developers. In a preliminary study, we undertook this step in an ad hoc, intuitive manner, but are pursuing active research on the specific criteria for membership in these sets and the process for generating them.

Once the initial *domain* set is constructed, each of its elements can be defined precisely, again in an iterative process executed by cooperating experts and developers. For example, the developer might make a first attempt, which the expert would then examine and revise if necessary. A stipulation placed on the process is that for each term, the parties must agree that they have converged on the same understanding, as interpreted from the definition, before moving forward. This implies that both parties must have the same understanding of *each term* in the definition, thus, that terms upon which the initial term depends must themselves be classified as *domain* or *common* and defined as necessary. This realizes the recursive nature of comprehension discussed above, and forces the parties to trace these dependencies.

In theory, this process comes up against the *symbol grounding problem*: the fact that within a closed system (e.g., the complete lexicon of a language), if any symbol, i.e., a word of the language, is only defined in terms of other symbols, then no meaning is ever accessible because the recursion never terminates. This is in fact the basis of some fundamental research questions in linguistic semantics.

While a solution to the general symbol grounding problem is not within the scope of this work (or this field, for that matter), a solution to a special case advances our cause greatly. A useful special case solution is one in which the bottom of the recursion is defined by design, and thus the trees representing term definitions are of finite depth. It is possible to construct such a special case by specifying a set of terms that are accepted without definition, and requiring that all definitions eventually bottom out with these terms. This is the purpose of the *common* set; by virtue of its construction, it consists of those terms deemed to represent knowledge common to those both inside and outside the domain, and its use is to provide the source lexicon in which all domain definitions must eventually bottom out. Thus another stipulation is necessary: no cycles are allowed in the chain of dependencies associated with any

term. A cycle would indicate that the recursion would never terminate, and thus that a common understanding would not be reachable. Again, as with recognizing and documenting the fact of dependencies, the structure of the domain map directs aspects of the process for its useful generation: if a cycle is produced, the semantics are not accessible. The parties must cooperate to rework the offending definitions.

The size of the common set is presumably inversely correlated with the level of integrity of the communication: the smaller the set, the more is explicitly explained. This flexibility allows adapting the size of the common set to the risks associated with the project. In our preliminary experience, however, we have achieved significant improvements in the preservation and propagation of intended semantics with even relatively large common sets. For example, from an excerpt of 1600 words from a natural language specification for a track control system for marine vessels, we generated, in cooperation with an expert, a *domain* set of 102 terms, smaller than what we might have generated with the application of stricter guidelines. The construction of a domain map for this specification yielded errors in the initial developer-generated definitions of 55, that is, over half, of the terms in the original *domain* set. This represents an initial miscommunication of semantics to developers via the original specification that was intended to be a form from which developers are expected to be able to create designs and programs. A particular error is quite revealing of the severity of the problem: the term *bearing* was initially incorrectly defined by developers. In addition, however, 38 other terms *depend on* a correct understanding of this term for their own correct understanding. This demonstrates that an incorrect understanding of only *one term* can invalidate the understanding of a large fraction of the domain knowledge in question, in this case, nearly 40%. Indeed, semantic dependencies are a major factor in communication.

The process of construction of the domain map forced the parties involved to recognize these hidden dangers and respond to them. The stipulations that there be no cycles in the map and that the parties converge to agreed understandings before progressing further forced the uncovering of unconsidered relationships and unquestioned assumptions. The process forced much of the implicit to be made explicit, and resulted in considered and converged-upon definitions, semantically accessible to those without expertise in the domain.

Several other authors (e.g., [4, 13]) as well as many practitioners have suggested that important domain terms need to be defined in specifications. However, no theoretical basis has been provided in the past for these suggestions, nor has a detailed structure or systematic process derived from available insights generated by a theory been

proposed. In addition, while the use of formal notations such as Z [10] and PVS [2] has been demonstrated to reduce certain types of error in specifications, it does not apply to the errors being discussed here. Formal specifications are written in notations that contain no real-world semantic information; they are formal refinements of the symbols used to represent semantics informally. However, if the semantics in question are not those intended in the first place, the best that formal notations can do is to formalize invalid representations. Effectively communicating domain-specific information from experts to developers is about generating *valid* representations. Domain maps help developers to *build the right thing*.

4. Tool Support

Zeus is an experimental toolset in development at the University of Virginia to support the creation of formal specifications [5, 14]. It allows the manipulation and analysis of specifications in the Z language and complementary natural language simultaneously. We have extended Zeus to enable the creation and analysis of domain maps, as well as their integration with subsequent formal specifications, should they be undertaken. In particular, from a plaintext or FrameMaker natural language file, Zeus allows the extraction of terms to construct the *domain* and *common* sets. It provides facilities for generating a definition database, and generation of a tree structure to relate terms to each other. The tree structure can be displayed for reference, and from this view, all dependencies are shown and all definitions are available by right-clicking on terms. The structure also indicates the existence of cycles so that the involved parties can address them. In addition, Zeus enables various forms of analysis of domain maps, for example, the computing of the distribution of tree heights for a given map. Such data when compared with the analogous data from other maps can reflect an intuition of lexical complexity for a given domain, that is, if the average number of levels of dependency is high, the concepts particular to this domain might be more difficult for a non-expert to grasp. This information can aid in decisions regarding the allocation of time and resources to areas of a project, since a less tractable domain represents a higher risk.

5. Beyond the Requirements Stage

Domain knowledge is both acquired and used beyond the requirements stage. In every phase of the software life-cycle, it is necessary to make decisions related to development that are influenced by domain knowledge. The particular algorithms used in implementations are based on

specifics of the application domain, as are details of test plans, and so on. Thus as development proceeds, the information contained within the domain map is used extensively and can be extended to reflect additional domain semantics upon which the validity of the project depends.

With this in mind, one can consider enhancing the role of the domain map to serve a purpose throughout the lifecycle. This is not essential—the domain map and the associated theory can be applied during requirements specification as described and thereby achieve the anticipated benefits of reduced requirements errors. However, the possibility of integrating the domain map with activities occurring later in the lifecycle led us to develop a new software structure, the program+, that is designed to link the formal and informal components of a desired software artifact as it develops. These components are complementary; neither duplicates the other because they serve fundamentally different roles. The formal component, in particular a formal specification if it is undertaken and the resulting programs that implement the design, serves to define the discrete mechanics by which the necessary algorithms are realized. The informal component, including the domain map and other forms of natural language documentation, serves to connect the symbols of the formal component to meaning in the real world. It is only by virtue of this connection that 1) the software artifact can have any value, and 2) its value, that is, how well it accomplishes its purpose, can be evaluated.

The domain map and the program+ structure force this connection to be explicit and to have a well-defined form. The program+ also requires the mapping to be maintained throughout the lifecycle. The only time when the formal and informal parts are separated is when the formal part is in the form of a bit string that is placed in the memory of a computer.

Validation is usually a side effect of system testing because it is assumed that discrepancies can be detected by observing the software's behavior. In fact, validation is not a property of a self-contained software artifact; rather it is a property of how that artifact functions within an environment or domain in the real world. The domain map and its dynamic form within the program+ enhance our ability to

understand the relationship of a formal model to that real world.

6. References

- [1] Brooks, F.P., "No silver bullet: essence and accidents of software engineering", IEEE Computer (April 1987).
- [2] Crow J., S. Owre, J. Rushby, N. Shankar, and M. Srivas, *A Tutorial Introduction to PVS*, Workshop on Industrial-Strength Formal Specification Techniques, Boca Raton, Florida (April, 1995)
- [3] Hayhurst, K., C.M. Holloway, "Challenges in software aspects of aerospace systems", IEEE Software Engineering Workshop, Greenbelt MD (Nov 2001)
- [4] Heninger, K., *Specifying Requirements for Complex Systems: New Techniques and Their Application*, IEEE TSE, SE-6 (Jan 1980)
- [5] Knight, J.C., K.S. Hanks, S.R. Travis, *Tool Support for Production Use of Formal Techniques*, International Symposium on Software Reliability Engineering, Hong Kong (Nov 2001)
- [6] Langacker, R.W., *Concept, Image, and Symbol: The Cognitive Basis of Grammar*, Mouton de Gruyter, Berlin New York (1990)
- [7] Lutz, R. Analyzing software requirements errors in safety-critical, embedded systems. *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 126-133. IEEE Computer Society Press (January 1993).
- [8] Rosch, E., B.B. Lloyd (eds.), *Cognition and Categorization*, Lawrence Erlbaum Associates, Hillsdale (1978)
- [9] Rosch, E., C. Mervis, W. Gray, D. Johnson, and P. Boyes-Braem. Basic objects in natural categories. *Cognitive Psychology*, 8:382-439 (1976).
- [10] Spivey J., *The Z Notation A Reference Manual*, Prentice Hall International (1992)
- [11] Ungerer, F., H.J. Schmid, *An Introduction to Cognitive Linguistics*. Longman, London New York (1996)
- [12] USAF Software Technology Support Center, *Guidelines for Successful Acquisition and Management of Software Intensive Systems*, version 3.0 (May 2000).
- [13] Zave, P and M. Jackson, *Four Dark Corners of Requirements Engineering*, ACM ToSEM, Vol. 6, No. 1, (Jan 1997)
- [14] <http://www.cs.virginia.edu/zeus>