A Linguistic Analysis of Requirements Errors and Its Application

Kimberly S. Hanks John C. Knight Elisabeth A. Strunk
Department of Computer Science
University of Virginia
151 Engineer's Way
Charlottesville, VA 22904-4740
+1 44 434 982 2216

ksh4q@cs.virginia.edu knight@cs.virginia.edu eas9d@cs.virginia.edu

ABSTRACT

A significant number of requirements errors can be characterized as failures to adequately take into account the system context of the software to be built. It has been shown that poor communication of domain knowledge pertaining to the system is a major source of such errors. The pervasive medium for this communication, natural language, is widely accepted to be problematic for high-precision communication because of its characteristic ambiguity and informality. However, it is nevertheless amenable to rigorous inspection and possesses its own body of research results. We analyze the domain knowledge communication problem from the perspective of current cognitive linguistic theory, and we describe insights deriving from this analysis. These insights are exploited to motivate a technique, the construction of a domain map, which allows the recording and propagation of real-world semantics essential to the production of software elements that have validity within their system context. The domain map construction and form help ensure the integrity of the recording and the effectiveness of the propagation. Our technique is demonstrated on parts of a real industrial requirements specification, and an evaluation is presented which takes into account both quality and cost.

1. INTRODUCTION

Mistakes in requirements are a significant source of software defects. In one study, Lutz found that the majority of safety-related errors that were found in the systems she studied derived from poor requirements [10]. Results such as this suggest that substantial reductions in software defects might be achieved if mistakes in requirements could be reduced.

The problem is a subtle one in that there are many sources of requirements errors including incompleteness, inconsistency, ambiguity, and so on. The core of the problem lies in the fact that the requirements specification for a software artifact is a statement that has to convey information from the domain experts who want the system built to software experts who will build it. The information cannot be stated in the familiar language of either, since both have to understand and work with it.

Software does not operate in a vacuum. It is meaningless outside the context of the complete system in which it operates, and the requirements for a software system derive from that system context. Domain experts have the domain knowledge that allows them to understand the system context thoroughly, but it is bound to be the case that software engineers usually do not. It is our hypothesis that failure to effectively communicate essential domain knowledge to software engineers is a major cause of requirements errors.

In this paper we introduce a linguistic analysis of the domain knowledge communication problem. We show that the way in which humans innately use language is not conducive to effective communication between domain experts and software engineers (or any other pair of experts in different domains, for that matter), and that this is a major source of requirements errors. It has been shown, however, that no solution in which communication is based entirely on formal notations is possible because natural language is, in fact, essential to provide real-world meaning to any formalism.

Using the analysis, we introduce a concept called the *domain map* and we argue that it can be used to document and structure the essential domain information in a requirements specification. We describe some simple tools for building and using domain maps, and we present a feasibility study of the use of a domain map.

2. LINGUISTIC ANALYSIS

We are concerned with maximizing the integrity of the communication between experts in a given domain and developers working on a unit of software that will function within that domain. The root objective of any attempt at communication is to recreate the particular semantics conceived of by one person in the mind of another. We examine this process from a cognitive linguistic perspective, with the goal of better understanding the problems particular to communicating domain-specific semantics. This yields a model of domain knowledge communication that explains precisely the nature of communicative breakdowns between domain experts and developers and that motivates particular techniques expressly designed to address them.

In this section, we introduce the main elements of this model,

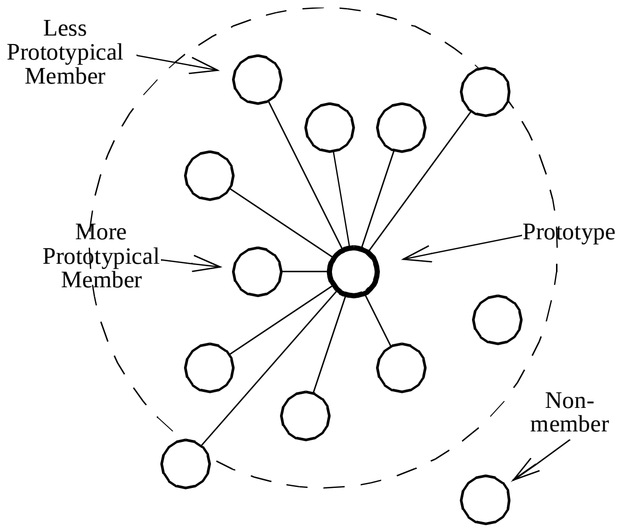


Figure 1. Cognitive categories.

cognitive categories, their internal structuring mechanisms of prototypicality, peripherality, and extension, as well as the principles that they collectively obey, cognitive economy and hierarchical organization. An understanding of these elements and the linguistic model they comprise motivates insights allowing us to approach the domain knowledge communication problem in controlled and directed ways.

2.1 Cognitive Categories

In order to understand how the semantics conceived of by one person may be recreated in the mind of another, it is first useful to have a notion of how these semantics are stored in a mind in the first place. Much work in linguistics and cognitive psychology has demonstrated that the universe of semantics understood by any person is organized into highly structured *categories* possessing certain properties [12, 11, 17, 7]. For our purposes, cognitive categories can be defined as follows:

Cognitive categories are collections of mental representations of entities encountered or imagined by an individual that are judged by that individual to be sufficiently similar to each other to count in some partitioning of reality as being the same.

Since there are many possible partitionings that are useful to us in our interaction with the world, an entity may be a member of many categories depending on the factors considered to be salient for the task or experience at hand. For example, a tree stump might be what is left of the trunk of a tree that has been cut down, but it may also be a welcome seat for a tired hiker.

Categories, in addition to being collections, have internal structure (see Figure 1). They can be visualized as radial arrangements of instances that bear degrees of resemblance to central prototypes. Instances closely clustered around the prototype bear stronger resemblances to it and instances further away are less prototypical. The tree stump above may indeed be a member of the *seat* category, but for most individuals it would be a non-prototypical member, whereas a chair such as is found in an office or at a dining table would be more representative.

Determination of which instances are prototypical or peripheral for a given category is an individual judgement call, based on accumulated past experience of the category possessor. In fact, membership itself is an individual determination: peripheral members of a category may be so much so that they might be judged as in or out of the category inconsistently by different individuals or even by the same individual as his experience changes. He may even be unable to decide at a given instant, allowing categories to overlap. Membership in cognitive categories, unlike membership in formal sets, is not binary.

Another dimension of internal structure that some categories may have is one or more layers representing semantic extensions to the category. Extension is essentially the selective appropriation of a subset of category semantics for use in classifying another group of entities. For example, before computers existed, the category *memory* was arguably restricted to representing a faculty of sentient life forms; its use in computer terminology abstracts certain, but not all, of its prototypical properties and exploits their commonality with aspects of human memory to allow an intuitive understanding of what computer memory does. Computer memory is not a peripheral member of the base category *memory*, rather, it is more like a category in its own right with a prototype and peripheral members (consider how *core* has gone from one to the other over time), but with a semantic origin and continued mapping to a preexisting category. This is essentially the cognitive view of metaphor. We will demonstrate that an understanding of extension, as well as of prototypicality and peripherality, motivates certain insights regarding domain knowledge communication.

2.2 Cognitive Economy

In addition to the internal structure that organizes each category individually, the collective set of categories possessed by an individual has been shown empirically to obey certain principles. Among these is *cognitive economy*.

Cognitive economy is the ability of humans to quickly associate with an entity a large number of attributes that might not be readily observable [12, 17]. This association is gained through the linking of an entity with a category based on the category name and possibly some prototypical attributes. This link leads to the heuristic ascription to the entity of additional attributes highly correlated with other known instances of that category. For example, a message receiver, upon receiving a message with the name *nest* in it, would link that name to the category represented by it in his inventory. Heuristically he would then ascribe to the instance attributes that might not be apparent but are highly correlated with other nests he has experienced, for instance that it is made of twigs and dirt, and that it might contain eggs or baby birds. If the message was, in fact, “Look at that bird’s nest!”, the receiver will likely look up, possibly into a tree, because those locations highly correlate with the locations of bird’s nests he has previously experienced. The receiver will already know what to look for and where; the message originator need not have indicated any description or direction. In this way, the individual on the receiving end of a message is able to conceive of rich and usually very accurate semantics while the message size is kept much smaller than it might otherwise need to be.

The heuristics used employ assumptions based on accumulated past experience of the message originator and receiver. These assumptions directly affect the structure of the categories that those involved in the communication possess, and they have been tuned through evolution to be effective in everyday communication.

2.3 Problematic Issues with Cognitive Economy

Since heuristic methods are the means by which the efficiency of accurate semantic transfer is increased, there are of course pathological cases that break them, i.e., that compromise either the efficiency or the accuracy of the heuristics. Communication of requirements from domain experts to developers is rife with such incidents.

One form of pathological case that defeats the purpose of cognitive economy is the event in which the original linking to a category is invalid. This occurs when a name signifies more than one category, i.e., there are name collisions. For example, a *monitor* can refer to the display on one's desk, a synchronization primitive, a minimal operating system, or even the child elected to manage traffic in the school hallway. While these are all distant extensions of a general *monitor* base category, each independently possesses internal categorical structure. If there is insufficient context to aid the message receiver in discriminating the intended extension, or worse, if he simply does *not possess* certain extensions, it is possible or even likely that he will ascribe to the entity in question a whole host of attributes that do not match those that were intended. The message receiver's ensuing behavior will be conducted in the context of this *incorrect* understanding, unless and until the error is recognized and corrected. Decisions and actions dependent on correct communication of the message will be invalid.

Another form of pathological case that defeats the purpose of cognitive economy is only subtly different than the above, but far more insidious. In this case, the original linking to a category is valid, that is, the name has the same basic referent for both interlocutors. But either the message originator or receiver has accumulated past experience that renders his version of the category more dense with features, and their relative correlations with the prototype more tempered and time worn, and therefore less open to adaptive reinterpretation. This difference in quality and density of experience causes the corresponding versions of the category possessed by two different people to have different topologies, i.e., different loci of prototypicality and peripherality, and different relations and distances between members. Any time two such people use this category, each assumes the topology and constellation of attributes consistent with his experience. Depending on the direction of the transfer, the result is that the message receiver constructs an understanding that is both qualitatively misaligned as well as either over- or under-specified, relative to that which was intended.

This phenomenon is a great danger to the accurate communication of domain specific requirements. Very many such categories exist where the denser category is possessed by an expert in a domain and a less constrained version is held by non-experts in the domain, notably, software developers. Consider, for example, use of the name *energy*: in physics, this category is constrained and well defined, with a precise set of attributes, whereas in lay usage, it is a rather amorphous thing. Context and other factors may or may not allow one or both participants to realize that something is amiss. Misunderstanding may or may not drive behavior that leads to errors. Errors that are made may or may not be caught. And since *energy* is only one of many thousands of such categories, spread over very many domains with which software developers interact, the probability that a very dangerous or expensive error will make it into production software is quite unacceptably more than zero.

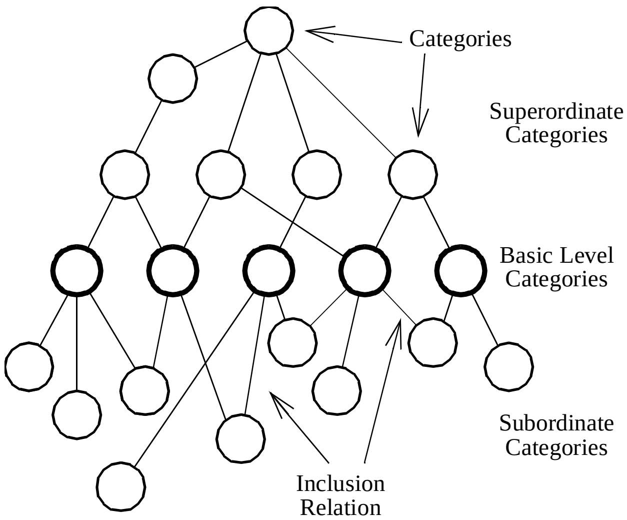


Figure 2. Hierarchical structure.

2.4 Hierarchical Structure

Further complicating the state of affairs resulting from the existence of pathological cases that flout the benefits of cognitive economy is a second principle: hierarchical structure. This principle is also obeyed collectively by the set of categories possessed by an individual.

Empirical evidence suggests that the categories we possess are organized into a hierarchy of specificity, with more general categories at higher levels and very constrained categories at lower levels. This hierarchy is one of inclusion, meaning that many low level, highly constrained categories are collected under the umbrella of a more general category, several of these more general categories are collected into still more general categories, and so on (see Figure 2).

Evidence also suggests that a particular level of this hierarchy has a special salience in our perception of the world [13, 12, 17]. 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. It has been termed the *basic level*, although it is not at the base of the hierarchy. It represents the categories first acquired by children, the categories we call up first when classifying a newly encountered entity, and the categories we use when introducing a new category into conversation.

Importantly, analysis of empirical data has demonstrated a further property of the basic level that supports its apparent 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* [13, 12, 17]. This property of the basic level further complicates the intact communication of domain knowledge.

This added complication derives from the nature of the domain specific categories used by experts. Domain experts have accumulated experience that results in their association of more attributes with certain categories (specifically those in their domain) than a non-expert would associate with the same categories. These attributes provide more dimensions along which potentially to collect and differentiate entities. This results 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.

The implication is that experts often tend to see, within their domain, lower-level, more constrained categories as basic, and therefore use them in ways that basic-level categories are used. On being presented with a new entity to be classified, provided the entity has some role within his domain, an expert is likely to associate it with a more constrained category than would a non-expert. Similarly, on introducing an entity into conversation, he is likely also to invoke a more constrained category.

This means that in addition to experts and non-experts possessing more and less constrained versions, respectively, of certain categories (as in the *energy* example above), the denser expert versions are more likely to come up in discussions relative to higher level umbrella categories because, to the expert, they are at the basic level. This translates to proportionally more opportunities for misalignment between the categories passed between originators and receivers than would occur because of the backfiring of cognitive economy alone.

The two pathological cases flouting the benefits of cognitive economy, name collisions and qualitative and density differentials, combined with the added complications of the interaction of domain knowledge and the basic level, in fact constitute the linguistic aspect of domain specificity in general. Communication across a domain boundary, communication that is essential in the correct preparation of a requirements specification, embodies exactly the properties that cause our natural machinery to fail. *It is not a part of human nature to get this right without serious and explicit intervention*.

2.5 Elicitation vs. Propagation

Finally, it must be noted that effective communication of domain knowledge is an issue throughout the lifecycle. The transfer of domain knowledge does not occur exclusively at one point in the development process; rather, it is negotiated back and forth in both directions at at least as many junctures as there are steps in an industrial lifecycle. And like an analog recording, it is subject to degradation at every transfer, as a specifier passes his best-effort, but not perfect, understanding to a designer, and so on to programmers, testers, and maintainers, and through all of the loops and switchbacks that an industrial development process follows.

The challenges provided by the domain-knowledge communication problem derive from far more than the goal of intact *elicitation* of domain knowledge. Elicitation is just the beginning. The challenges provided by the domain-knowledge communication problem derive at least as much from the goal of intact *propagation* of domain knowledge.

It is this recognition that drives this work. Although we are also conducting a related line of research on the linguistic issues involved in initial elicitation, we are concerned here with the recording of acquired semantics such that they can be *propagated* with the highest possible level of integrity, minimizing noise and degradation of the message over the duration of the process. Our linguistic analysis demonstrates the ways that a message can be degraded, and our knowledge of industrial processes indicates that the problem is recreated over and over in what are essentially new elicitation rounds every time a message must be passed.

These insights dictate that the shape of a solution should be one that enforces the placing of elicited information in a durable and consistent form that is cognitively accessible to all those who might have need to use it, and in so doing, does not recreate the elicitation problem and multiply its attendant noise and degradation further along in the process. We now turn to the problem of determining the medium of this recording.

3. ON THE NECESSITY OF NATURAL LANGUAGE

Many approaches to removing ambiguity and increasing the precision of a software specification advocate the practice of converting its representation to a formal one. The motivation to do so is correct for several reasons, the most familiar being that formal languages are inherently and entirely rule-based, unambiguous, and analyzable.

However, every software artifact to be built begins, by its very nature, as an informal cognitive concept in the mind of a client. Our assertion is that while any useful solution must eventually be formalized in order to be executable, formalization that is attempted too early or is overly relied upon as the exclusive or main representation of a developing software artifact is flawed. We assert that the dynamic representation of a software artifact, as it develops from an informal cognitive concept into an executable piece of code, must be accompanied by a complementary element in natural language, which takes a certain form, and which records the domain knowledge to be propagated. This recording then itself becomes an inextricable component of the specification, providing information that no formal specification can by itself confer, and serving as genuinely useful documentation.

This assertion begins from the recognition that the errors that can be caught through analysis of a formal specification are only those that are possible to catch within the closed world of the model that is being analyzed. Analysis of formal specifications can demonstrate internal inconsistency and faulty logic, but it cannot demonstrate completeness of the model, or validity of the elements represented. Formal representations carry and confer no meaning [18]; they only *represent symbolically* entities and relationships that people have decided to model. Formal representations are therefore incapable of providing the recording format for the propagation of access to essential domain specific semantics that is sought.

To summarize, the following issues arise in choosing the medium of the recording for domain knowledge:

- every representation starts informally in the domain,
- explicit and considered access to essential semantics must be propagated to all those who might need it throughout the development cycle,
- this explicit and considered access to essential semantics cannot be passed with accuracy through a lossy channel of non-expert relays, and

- formal language is not capable of carrying or conferring meaning.

Therefore, the recording format for the propagation of explicit and considered access to essential semantics must and can only be natural language.

It is not an accident that natural language is the medium of context for the entire process and the bootstrapping medium that allows any other kind of representation to be negotiated and constructed. When the utility of any other kind of representation breaks down, the discussion defaults to natural language for a reason: “pretty good” heuristics that deliver are better than “perfect” solutions that cannot be made to work. We know natural language has weaknesses, but we also know that it is all we had before we learned any formal representations, and we learned to manipulate them through transfer of ideas carried by natural language. When all else fails, return to first principles.

3.1 On Mental Models

Noteworthy at this point is a parallel that can be drawn with the acquisition and use of mental models. Leveson discusses some of the threats to safety that can arise when manual tasks are automated [9]. Operators, who once accomplished these tasks directly, are recast as outside monitors of the system that now performs the labor algorithmically. She cites Brehmer, who argued that this conversion of the operator’s role results in his comprehension of the system changing from a concrete to an abstract one, increasing its complexity and reducing its intuitive tractability. This was shown to lead to errors in system comprehension and the decisions based on it [1].

In a similar way, a conversion from a natural representation to a formal one, as we assert is often done too soon or relied upon too exclusively, removes semantics and relegates all relationships to be entirely syntactic and disconnected from the real world. Our techniques have the potential to reinstate some concreteness and intuitive tractability to a system model by expressly providing a mapping back to more common, familiar, and semantically accessible entities.

In addition, the cognitive construction of mental models suffers from the flaw that once an assumption motivates a particular interpretation, this interpretation is difficult to change [9]. Users of a model, correct or incorrect, are more likely to accept new information that is consistent with the model they have constructed and to discount information that is not. The implication is that it is in the best interest of a project to provide access as soon as possible to essential semantics that are as correct as possible. By doing so, erroneous assumptions, such as those that can enter in ways already discussed, are fended off, and incorrect mental models are not allowed to become fixed. For example, it should not be possible for a developer to walk away believing that his notion of *bearing* is what was intended by a client using the term with a very narrow and specialized meaning within his domain. But, as we will discuss, exactly this happened during our feasibility study before the application of our technique, and it had cascading effects which invalidated dependent aspects of the model.

3.2 Refining Ignorance

Several years ago, Berry advocated the inclusion in any requirements team of a person to fulfill the role of a “smart ignoramus” [3, 2]. This smart ignoramus is defined as someone who is: (1) generally intelligent and, in particular, (2) has a demonstrated facility with the use of language, but (3) who does not have any knowledge of the domain of the system for which a piece of software is being developed, and (4) is not afraid to ask questions. The point of including such a person derives from Berry’s personal experience. More valid system models resulted when he happened to serve in roles where the above properties held, and less valid requirements resulted when, for example, he knew more about the domain but was still not an expert. The implication is that he had flawed mental models that had become fixed, came up with fewer questions because he assumed he knew more than he did, and failed to probe for more information.

Berry’s approach is an excellent start, under which we place linguistic foundations. Our hypothesis is that the smart ignoramus is somewhat successful because he, through possession of certain properties, is more able to attend intuitively to potential linguistic breakdowns of the kind discussed earlier. The smart ignoramus is more capable of stepping outside of the discussion proper and realizing when a name might be matched to the wrong category, or have a denser or qualitatively different category topology for one person than for another. The smart ignoramus does not, however, know or understand this explicitly, and so his recognition of such situations is incomplete and his handling of them is ad hoc. The next step, based on the analysis we have provided of the cognitive and linguistic phenomena involved, is to control and direct this facility for comprehending and making available the intended semantics with more consistency, integrity and precision.

3.3 On the Scope of Natural Language

Ryan argues essentially that approaches that seek to invoke forms of natural language processing to automate understanding of a client’s needs and thereby relieve the requirements bottleneck are ill-conceived [14]. We do not disagree. However, he goes on to conclude that because automated natural language understanding is a feat not likely to be accomplished usefully, techniques exploiting and focusing on natural language for requirements are of limited and peripheral use. One of his premises is that while the syntax and specific semantics of a specialized (i.e., domain-specific) language are a factor in what he terms the communications gap, there are issues outside of natural language which are of greater significance, including “...unstated assumptions that reflect the shared (‘common sense’) knowledge of people familiar with the social, business, and technical contexts within which the proposed system will operate” [14]. The implication is that these issues are not approachable with natural-language techniques.

Our assertion is that these issues, rather, are very definitely within the purview of potential natural-language techniques, because linguistic and psychological research has generated results demonstrating that all of this knowledge is tied up together in one mental representational mechanism to which natural language is the most transparent window. Language comprehension is far deeper than the decoding of syntax and the mapping of explicit semantics; the entire structure of categories and the governance of their communication by cognitive economy demonstrates that most of semantic transfer is actually *implicit*. Very much surrounding the issues of assumption and tacit knowledge *can* be approached from a linguistic perspective, motivating the shape of techniques employing natural language. Berry’s smart ignoramus has demonstrated so informally, and our work has the goal of producing something more refined, controlled, and directed. Language is far more than explicit syntax and semantics—it is pragmatics, and context, and assumption, and implicature. Armed with results from linguistic research, we can absolutely attack some of these issues linguistically and rigorously in the requirements realm.

So, having now: (1) an analysis pointing to breakdowns deriving from the heuristic use of assumption in communicating semantics; (2) an argument that explicit and considered access to essential semantics must in fact be provided in natural language; and (3) given some attention to some specific points of relation to other work, we move next to specifying the form of the structure in which essential semantics can be recorded and propagated without degradation.

4. THE DOMAIN MAP

The domain map is a structure designed to allow explicit and systematic access to essential semantics in a form that is meaningful to those who do not possess expert knowledge of the domain in question. The idea that domain knowledge needs to be documented in a specification has been proposed by others, e.g., the *text macros* of Heninger [4] and the *designations* of Zave and Jackson [18], but without a theoretical basis.

We have demonstrated that developers, who are non-experts with respect to the domain in question, are unable to access the intended semantics directly from domain experts because the lexicon of the domain expert is partially incompatible with that of the developers. Even if they could access the semantics this way, it would be impractical for a developer to drop what he was doing to find a domain expert every time he needed an explanation. This itself might not be possible, because access to domain experts might be limited or non-existent after a certain time. Furthermore, since domain experts' notions themselves might not be totally consistent, a practice of finding an expert every time an explanation was needed allows the possibility of inconsistent expert explanations making their way into the model.

All of these issues demand that the information instead be put in a centralized form that is both cognitively accessible to developers and agreed with and among domain experts. This need not preclude dynamic requirements, which are a reality with which we have to deal. Rather, it precludes inconsistency of the currency of the description and unavailability of a mapping of that currency into a form comprehensible to developers. A requirement might change, but the language available to describe it should not, and the description should be meaningful to all who need to use it. This necessity is analogous to that of having common units in which to compare measurements: a common representation allows the detection and representation of possible changes to be more apparent, and allows access to the semantics of the description whether there are changes or not.

What we seek, then, is ideally a one-time conversion of the domain specific representation of essential semantics to a form accessible to those with a common base set of representations. This entity must in effect map domain specific representations to representations built out of common terms and phrases. For these purposes, we define *common* and *domain* narrowly as follows. *Common* refers to that set of terms for which the association between a particular term and its semantics is sufficiently similar among interlocutors that relevant miscommunication is highly unlikely. In other words, a term is common if the associated cognitive categories possessed by any two people within a project have essentially the same internal structure with regard to prototypes, peripheral members, and extensions. *Domain*, then, refers simply to all terms which are not *common*, indicating that if a term has a domain specific meaning, it is associated with a category that is potentially denser, claims a different prototype, variations in the choice of peripheral members, or one or more extensions not included in the category organization of a non-expert. This situation is the basis for all miscommunication as analyzed in section 2.

These definitions and overview of purpose motivate a structure, which we call the *domain map*, having the following properties:

- A specification for a software system includes a domain map.
- All domain specific terms that are relevant to the development of the specified software system are associated directly or indirectly with definitions consisting of exclusively common terms.
- No cycles are permitted in the use of definitions within definitions.
- The domain map and the documents to which it points are the only sources of domain-specific definitions to which developers can refer.

The original sources of information for the domain map are a wide variety of domain-specific documents. Many will be natural language statements of requirements for a software artifact that is to function in some greater system. Such statements are of flexible form. Ideally, a considered written statement is desired, as it is more likely that certain issues have been given attention simply by virtue of its construction. However, a tape recording or even just notes from a requirements interview would be useful in generating an analyzable lexicon, and further, this flexibility of source can be customized to suit the needs and resources of the client and the development organization.

Along with a statement of requirements for a software element, the greater system in which the element is to function must receive particular attention, since it determines the domain from which the correct, though still largely inaccessible, semantics derive. Some parts of the natural language statement will therefore have meaning only well understood by domain experts, but which must be communicated effectively to developers so that the implications of this meaning can be preserved in the model under construction.

Thus the technique first calls for a way of indicating terms and phrases in the natural-language statement that might have meanings unclear to developers. This is a hard problem in its own right. We defer specifically the issue of determining what is within the boundary of domain knowledge and what is not, i.e., determining whether the semantic organization of the categories in question are sufficiently similar between potential interlocutors, to a later discussion. We are researching this issue as well from a linguistic perspective.

An intuition-driven decision at this point, however, allows us to move forward to describe the rest of the technique. We elect, then, to first have a developer make a pass through the requirements statement, indicating words and phrases that he either outright does not understand, or has reason to believe might have a domain specific meaning of which he is unaware.

Next, a domain expert makes a pass through the same requirements statement, annotated with the developer's indications. The purpose here is for the domain expert to have a chance to validate the developer's indications as well as to add to them, without having to bear the entire time burden of doing it all himself from the beginning (since this is impractical and unlikely).

This trade of active role can be iterated as necessary for the domain expert, or several, and the developer, or several, to converge on a set of terms in need of semantic accessibility from outside of the domain. The choice of the number of experts and developers who have input at this point provides another dimension of flexibility along which the technique can be tailored to the needs and resources of the client and development organization. Furthermore, in practice, the number of iterations needed is small though it will be affected by the number of experts and developers who have input.

Next, each entry in the list of problematic terms and phrases is defined in a manner that paraphrases domain specific terms and phrases using, where possible, terms that are common[1]. It is recognized that this is not always directly possible, i.e., that a definition for a given domain specific term may itself contain domain specific terms. Thus the construction of the domain map is another iterative process in which each member of the set of definitions associated with the initial set of domain specific terms and phrases is itself subjected to an analogous inspection. Iterations beyond the first might in addition add terms to either the domain or common sets, according to determinations like those used in the original set construction.

A rule is defined by the technique whereby every term to be defined be mapped, indirectly if necessary, to a definition comprised of exclusively common terms. The definition of any term or phrase in the original set therefore describes a tree[2] in which the root and all of the remaining non-terminal nodes are domain terms, and all of the terminal nodes are common terms.

This domain-map tree structure can be subjected to analysis. In particular, we would like to know about several properties of the tree, and collectively about the set of trees resulting from the original term set. A simple check to determine whether the terminal condition has been reached, i.e., whether every domain specific term can be paraphrased directly or indirectly by exclusively common terms, indicates to the experts and developers whether the process is complete for a given term, i.e., whether there is a definition completely accessible to a non-expert. The depth of the tree gives some idea of the semantic complexity of a given term, and the maximum and average depths give an idea of the overall complexity of the domain lexicon, i.e., a measure of how far removed it is from everyday common language. These measures can be used to flag concepts at high risk for miscommunication, and to drive the amount of rigor applied in maximizing the validity of a model.

What the domain map provides is an organized and structured entity that documents essential domain knowledge. The procedure for building it accomplishes by design the goals set out in the earlier problem analysis. The domain map provides explicit and considered common access to the intended semantics, it remains in natural language, and it has permanence enabling it to become an element of the documentation for use later in the process by those even further removed from the domain experts. This mitigates the need for additional independent and potentially inconsistent elicitation rounds. It provides as well the flexibility to be adapted to the needs and resources of the client and development organization, through selective application to the areas at highest risk for miscommunication, and by allowing applications of various levels of rigor.

1. This strategy differs from that of Leite [8]. His Language Extended Lexicon (LEL) technique instead encourages defining domain specific terms in terms of each other. The problem with this is that definitions can be circular and thus there is no assurance that semantics can ever be accessed, i.e., the symbol-grounding problem is not addressed.

2. Technically, it describes a directed acyclic graph since some definitions will include the same terms, but, for our purposes, conceiving of the domain map as a tree is equally valid.

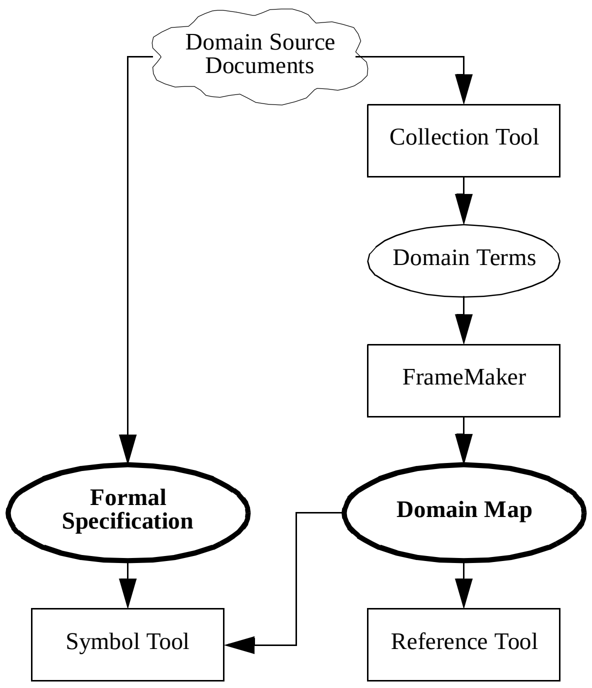


Figure 3. Support toolset.

5. TOOL SUPPORT

The domain map is a large and complex structure, but it is also an essential structure. Creating this map for any given application is both time consuming and difficult. The developers must ensure that all of the domain terms requiring definitions have been identified, that the definitions are correct, and that all the definitions are eventually grounded in common terms. For a large specification in a complex domain there might be a large number of terms with many definitions far removed from common terms.

Clearly, tools to support the initial creation of the domain map and to undertake the various forms of analysis that are possible is an attractive idea. To investigate the potential of tool support, we have enhanced an existing toolset, Zeus [6], to provide facilities for domain map creation and analysis. The tools and their relationship to the various artifacts is shown in Figure 3.

Zeus is based on the FrameMaker desktop publishing system and the Z/EVES verification system [15]. It provides comprehensive facilities for manipulation of both natural language and Z [16]. Its support for natural language is used in two ways. The first and primary use is to create and modify formal specifications that integrate formal elements written in Z with natural language. The second use is to manipulate project and domain-related documents in whatever format domain experts and developers require using all of the facilities of unmodified FrameMaker.

5.1 Collection Tool

In order to support the creation of domain maps, Zeus has been extended with a facility that collects domain-specific terms and phrases. As discussed above, the creation of the list of domain terms and phrases is undertaken by the developers and domain experts making various passes through source documents.

For natural language source documents, the analyst can invoke a *collection tool* that captures terms and phrases and stores the list in a file. Selection of a term by the analyst merely requires the text of the term be highlighted and a key struck.

The list can be manipulated as different passes are made over the source materials. The list is then formatted as needed for the domain map in preparation for the addition of the definitions. Natural language definitions are entered using the text manipulation facilities of FrameMaker.

5.2 Reference Tool

Use of the domain map is effected by a *reference tool*. The reference tool creates a graphic display of the domain map that shows all of the defined terms in alphabetical order. An example of the reference tool's display when applied to a sample specification is shown in Figure 4.

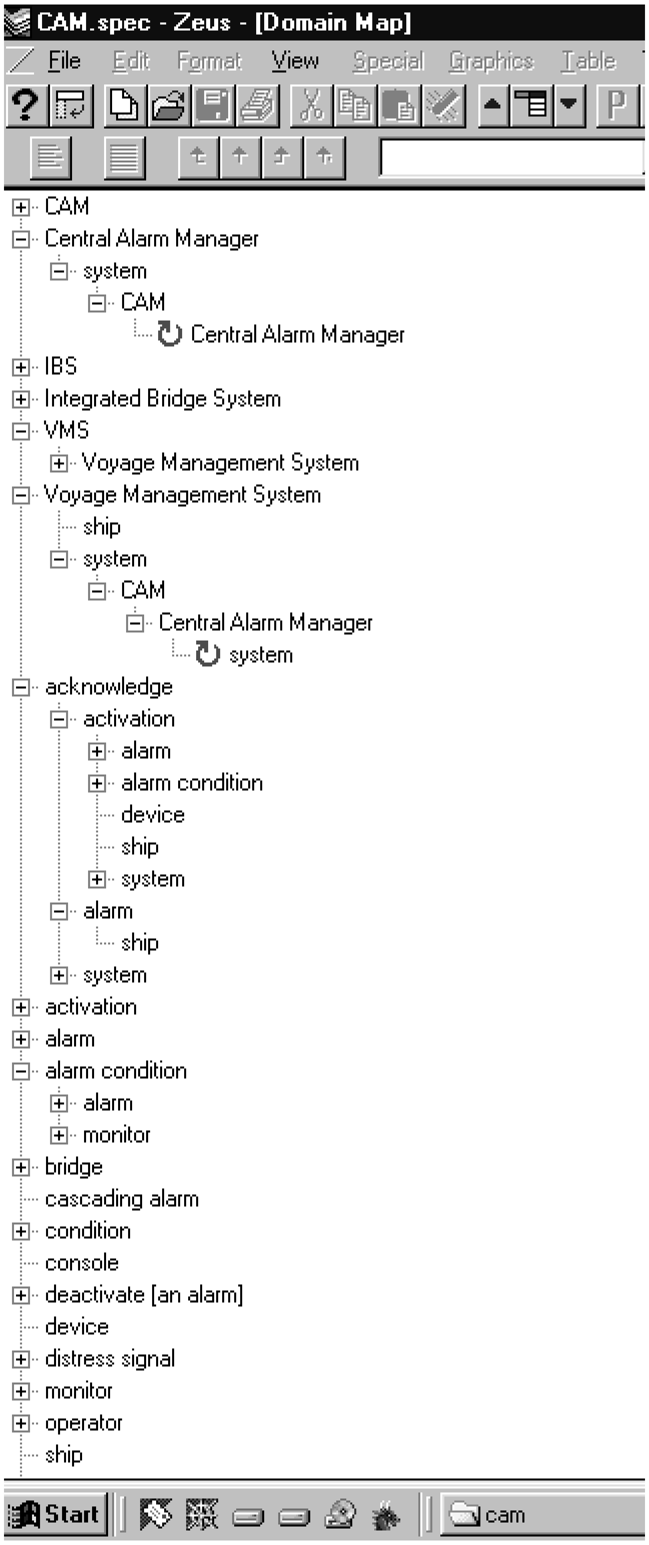
Since definitions are stated frequently using other domain terms or phrases, the reference tool displays for each domain term the set of domain terms upon which it depends. Finally, the reference tool displays the definition of any term if the right mouse button is clicked over it.

The reference tool performs a variety of analyses on the domain map as the display is built. For example, it checks for definitions that are either duplicates or empty, i.e., there is a placeholder for the definition text but no text has been entered. It detects cycles so that terms with circular definitions can be refined (note examples in Figure 4). It also checks the tree of definitions and ensures that all definitions are grounded in common terms.

Finally, the reference tool computes various metrics from the domain map including the distribution of the number of levels in the tree of definitions, the number of defined terms or phrases, and so on.

5.3 Symbol Tool

If a specification includes a formal component, the domain map can be tied closely to this component since symbols used in the formal component can be given meaning by the domain map. A third tool within Zeus, the *symbol tool*, checks the complete list of symbols used in the formal part of a specification and compares it with the definitions in the domain map. Any symbol used in the formal part for which no definition appears in the domain map is flagged.



```
⊞ CAM
⊟ Central Alarm Manager
    ⊟ system
        ⊟ CAM
            ↻ Central Alarm Manager
⊞ IBS
⊞ Integrated Bridge System
⊟ VMS
    ⊞ Voyage Management System
⊟ Voyage Management System
    ship
    ⊟ system
        ⊟ CAM
            ⊟ Central Alarm Manager
                ↻ system
⊟ acknowledge
    ⊟ activation
        ⊞ alarm
        ⊞ alarm condition
        device
        ship
        ⊞ system
    ⊟ alarm
        ship
    ⊞ system
⊞ activation
⊞ alarm
⊟ alarm condition
    ⊞ alarm
    ⊞ monitor
⊞ bridge
cascading alarm
⊞ condition
console
⊞ deactivate [an alarm]
device
⊞ distress signal
⊞ monitor
⊞ operator
ship
```

Figure 4. Sample output of the reference tool.

6. FEASIBILITY STUDY AND EXAMPLE

Use of the approach described in this paper is designed to help bridge the gap between the expert's knowledge of a domain and the knowledge of engineers who need to create software artifacts. Since this gap is responsible for many software defects, the long-term assessment of the approach has to be based on measurement of its effect on defects in industrial-scale development.

We are planning such assessment, but, in the short term, we have undertaken a small study to assess the overall feasibility and utility of the domain-map concept. This study is based on a large specification from the maritime control domain and we present it as an example to illustrate the application of the concepts.

6.1 Description

The starting point for this feasibility study is part of an existing, publicly available international standard for track-control systems for large, ocean-going ships. Specifically, it is the International Electrotechnical Commission's specification for track control systems on ships [5]. This is a natural-language standard document that defines many functional and safety properties that operational track-control systems must have. A track-control system allows an operator to enter a series of *waypoints* (intermediate or final destination points), and then the system steers the ship along those waypoints to the desired location. The standard includes test cases that are used to test compliance and the expected results of the test cases.

We have developed a version of this international standard in Z and natural language that is a complete rewrite of the original. It was written as a testbed for tools and techniques in the area of formal methods. The full standard (both versions) encompasses three categories of ships, differentiated by their ability to guide the ship through *turns* (where a ship has attained a waypoint and must change direction in order to proceed to the next waypoint). For simplicity in this initial study, we have limited our analysis to Category A ships, i.e., those that provide no assistance from the track-control system to the operator during turns. Within this category, we have further restricted the requirements to those that support entry of only one waypoint at a time.

An excerpt of the requirements section of the original standard served as our domain source document. After excerpting the relevant portion, we acted as developers and selected the set of terms we perceived to be domain terms. A domain expert in marine systems then inspected the document to choose any additional domain terms that we had missed. We then created an initial version of the domain map, including definitions of the domain terms derived from the natural language text and definitions of any new domain terms used when creating the domain map. The domain expert then inspected the candidate domain map, marking corrections where needed. Finally, we incorporated the corrections into the domain map and then analyzed the map for characteristics of the terms' definition trees.

6.2 Results

The requirements excerpt we used was 1600 words long and included a domain set of 102 terms. In the source document, some of these terms were defined and others were not. The source document [5] contained a glossary but those terms that were defined were defined in various places—definitions were not limited to the glossary. Of those that were defined, some were defined clearly while others were defined using domain terms. For example, "sway" was defined as "Athwartships component of ship motion (positive to starboard)." Some of the terms were ambiguous enough that the domain expert was unsure of their meaning. Use of the domain map revealed and resolved these issues.

The domain expert's initial inspection uncovered six domain terms that we, acting as the developers, had missed. His inspection of the candidate domain map resulted in revisions to 55 of the 102 candidate definitions (54%) and four additional domain terms generated from the map. These revisions indicate a higher level of understanding than might have been possible without application of the technique.

While many of the changes to the definitions were minor, some transformed the intended semantics significantly. For example, the definition of "bearing" was completely incorrect. It was used five times throughout the document, but perhaps more significantly, 38 words contained "bearing" as an ancestor in the domain map. Because it was misused in the definition of "heading", the semantics of most of those whose ancestry was indirect were not affected by the mistake, but this example illustrates the potential magnification of seemingly insignificant misunderstandings.

The updated version of the domain map contained 106 terms. Many of the terms had definitions accompanied by other explanatory text. Analysis of the domain map was conducted on the definitions only, and it revealed two cycles. They were removed by renegotiating the definitions to avoid the domain terms that caused the cycles.

The maximum height of the domain-map for a single term was 12, i.e., one or more definitions exist in which 12 layers of definitional refinement were required. The average height over all definitions was 5.0. The maximum number of references to other domain terms in any single definition was 8, and the average number was 2.3.

6.3 Evaluation

In this case, the creation of the domain map was fairly straightforward and the interleaved iterations of developers and domain experts was effective. The confusion over definitions was expected, but the potential for confusion brought about by severely erroneous definitions upon which many other terms depended was not. This is clearly a potentially significant source of subtle misunderstandings.

The data collected so far on the form of the domain map structure is encouraging. The structure is not trivial but it appears to offer a significant resource to the developer seeking to comprehend the domain.

The approach we have described does not require large amounts of resources in its application yet it has the potential for significant cost reductions in all lifecycle phases. It addresses a major source of risk, and, as a result, offers the potential for a large reduction of rework. It also has two dimensions of flexibility that allow the cost of application to be controlled. First, the technique can be applied to just the critical parts of a software artifact. Second, the rigor with which the technique is applied can be adjusted by varying the size of the domain set.

7. CONCLUSION

Numerous mistakes in requirements arise because of failure to take into account the context of the system in which the software operates. In many cases, this failure occurs because essential domain knowledge is not communicated effectively from domain experts to software engineers. We have introduced a linguistic analysis of this problem based on three important linguistic principles: (1) cognitive categories and their internal structure; (2) cognitive economy in the communication of categories; and (3) the particular properties of basic level categories. From this analysis, we have shown that the structure of our natural cognitive endowment is at least one of the reasons why domain-knowledge communication is fraught with the danger of misunderstanding.

Natural language is, in fact, necessary to provide the semantics not available in requirements specifications presented in formal notations. But while common natural language is not conducive to precise detailed communication, explicit and considered use that is directed by linguistic insight has the potential to be very useful. Our proposal consists of the construction and use of a structure called the domain map within the general notion of documenting specifications.

We note that the domain map is just a repository despite the fact that it is designed to deal with a major source of error. Once created, none of the developers (those in most need of a comprehensive understanding of domain knowledge) is required to use it. However, it is not merely a crutch for those who choose to use it. It is a crucial concept because it bridges the misalignment of structure between the everyday human model of terminology and a form more suitable for precise communication. Its use should prove attractive to software engineers, but its use could easily be mandated in a suitably precise software development process.

The tools we have developed to support the creation and use of the domain map provide immediate utility because they are capable of several forms of analysis as well as presenting critical domain information in an effective manner.

Finally, our feasibility study has given us an informal indication of the utility of the approach based on experiments with a substantial specification. We note that our feasibility study is very preliminary, and more extensive and detailed experiments in cooperation with industrial development groups is planned.

8. ACKNOWLEDGEMENTS

It is a pleasure to thank Brenden Schubert of the University of Virginia for his extensive assistance on the terminology of maritime systems and Sean Travis of the University of Virginia for his development of the extensions to the Zeus toolset. We also thank Jonathan Michel of Northrop Grumman for his assistance with the track control specification. This work was funded in part by NASA under contract NAG-1-2290.

9. REFERENCES

[1] B. Brehmer. Development of mental models for decision in technological systems. In *New Technology and Human Error*, Eds: Rasmussen, Duncan, and Leplat. John Wiley and Sons, New York, 1987.

[2] D. Berry. Formal methods: the very idea, some thoughts about why they work when they work. *Electronic Notes in Theoretical Computer Science*, 25, 1999. http://www.elsevier.nl/locate/entcs/volume25.html.

[3] D. Berry. The importance of ignorance in requirements engineering. *Journal of Systems and Software*, 28:179-184, 1995.

[4] K. Heninger. Specifying requirements for complex systems: new techniques and their applications. *IEEE Transactions on Software Engineering*, SE-6(1):2-12, January 1980.

[5] International Electrotechnical Commission, Maritime navigation and radio communication equipment and systems - Track control systems - Operational and performance requirements, methods of testing and required test results. Project number: 62065/Ed. 1 (2000-08-11)

[6] J. Knight, K. Hanks, and S. Travis. Tool support for production use of formal techniques. *International Symposium on Software Reliability Engineering*. IEEE Computer Society Press, November 2001.

[7] R. Langacker. *Concept, Image, and Symbol: The Cognitive Basis of Grammar*. Mouton de Gruyter, Berlin, 1990.

[8] J. Leite and A. Franco. A strategy for conceptual model acquisition. *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 243-246. IEEE Computer Society Press, January 1993.

[9] N. Leveson. *Safeware: System Safety and Computers*. Addison Wesley Publishing Company, Reading, Massachusetts, 1995.

[10] R. Lutz. 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.

[11] C. Mervis and E. Rosch. Categorization of natural objects. *Annual Review of Psychology*, 32:89-115, 1981.

[12] E. Rosch and B. Lloyd (eds.). *Cognition and Categorization*. Lawrence Erlbaum Associates, Hillsdale, 1978.

[13] E. Rosch, C. Mervis, W. Gray, D. Johnson, and P. Boyes-Braem. Basic objects in natural categories. *Cognitive Psychology*, 8:382-439, 1976.

[14] K. Ryan. The role of natural language in requirements engineering. *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 240-242. IEEE Computer Society Press, January 1993.

[15] M. Saaltink. The Z/EVES System. *Proceedings: 10th International Conference of Z Users*, Lecture Notes in Computer Science 1212, Jonathan P. Bowen, Michael G. Hinchey and David Till (Eds.), Springer-Verlag, Berlin, pp 72-85, April 1997

[16] J. Spivey. *The Z Notation: A Reference Manual, Second Edition*. Prentice Hall International (UK) Ltd, Hemel Hempstead, UK, 1992.

[17] F. Ungerer, H. Schmid. *An Introduction to Cognitive Linguistics*. Longman, London, 1996.

[18] P. Zave and M. Jackson. *Four dark corners of requirements engineering. ACM Transactions on Software Engineering and Methodology*, 6(1):1-30, January 1997.