

Tools Supporting the Communication of Critical Domain Knowledge in High-Consequence Systems Development

Kimberly S. Wasson, John C. Knight, Elisabeth A. Strunk, and Sean R. Travis

Department of Computer Science
University of Virginia
151 Engineer's Way, P.O. Box 400740
Charlottesville, VA 22904-4740, USA
{ksh4q|knight|strunk|srt3k}@cs.virginia.edu

Abstract. Predictably achieving requirements validity has proven extremely difficult because of the informal nature of this property, and poor communication of application domain knowledge is implicated as a main barrier to increasing this validity. In related work, we developed a methodology that exploits cognitive psychology research to improve this communication. In this work, we describe support for the methodology via a toolset that directs involved parties in the construction of artifacts defined by the methodology; these artifacts by their structure and content enable the spread of application domain knowledge among developers and throughout the lifecycle. The toolset in addition supports documentation, various forms of analysis, and integration with other lifecycle artifacts. We used these tools in collaboration with an industry partner to develop experimental documentation for recording and dissemination of domain knowledge relevant to the design of a medical device, and we here describe this experience.

1 Introduction

Erroneous or *invalid* requirements are implicated as a major source of defects in the software of high-consequence systems [14]. Achieving high-quality requirements necessitates high-integrity communication [7], but innate mechanisms of human communication are not well-suited to this goal [6, 5]. Researchers who study human communication have generated results that can be exploited for purposes of improving communication in the development of such systems. Linguistic knowledge about particular aspects of human communication can be embodied in an artifact that serves to organize and allow processing of content essential to the effective communication of critical domain knowledge [4, 6, 5]. This approach generates a large amount of natural language content that is highly structured, but difficult to manage effectively. In this work, we offer a proof of concept of tool support for acquiring, organizing and processing that material. The tools we describe automate much of the application of important results from linguistics to software engineering thereby providing the opportunity for significant improvement in the quality of documented requirements.

We begin with a brief examination of the relationship between communication and software validity. Validity refers to whether a software artifact intended for a purpose is “the right thing,” that is, whether it fulfills its intended functionality under intended constraints and without unintended consequences. Validity is necessarily an

informal property since it is a measure of mapping to human intent. It is thus difficult to achieve and demonstrate, since intent might change, vary among stakeholders, be incompletely considered by either developers or stakeholders, include inconsistencies that result from poor logic or ambiguity, or simply be inaccessible as a result of insufficient acquisition methods or lack of propagation of sufficiently acquired intent.

Natural language is the primary medium for the communication of intent, and herein lies much of the problem. Not only must this intent be isolated and captured in order to get the requirements right, but it must be propagated through the development process with integrity at any number of transfer points. Thus far, those developing high-consequence systems have rarely been able to get this right, and even then, not systematically or predictably. This fact is demonstrated by the numerous incidents, accidents, and other failures in high-assurance software, as well as in the schedule slippages and cost overruns that plague even non-safety-critical projects. By and large, these problems derive from communication deficiencies early in the lifecycle [7, 14].

In related work, we developed a methodology that exploits cognitive psychology research to improve this communication. In this work, we describe support for the methodology via a toolset that directs involved parties in the construction of artifacts defined by the methodology; these artifacts by their structure and content enable the spread of application domain knowledge among developers and throughout the lifecycle. The approach was brought into an industrial environment and used to develop experimental requirements and specification documentation relevant to the development of a medical device. This allowed us to demonstrate the feasibility of construction of a complex requirements artifact that is theoretically founded in a linguistic model using extensive tool support. The artifact is scalable in both representable complexity and processibility into new knowledge, and integratable with existing development processes. It provides a proof-of-concept that certain forms of breakdown that plague communication of application domain knowledge can be systematically overcome or avoided while keeping within the confines of a professional development process.

The structure of the paper is as follows: we first provide the context of related work, followed by a brief review of the approach. We then discuss the toolset and its enabled facilities. Finally, we report on our experience with industrial collaborators, which demonstrated the feasibility of construction of a theoretically-founded complex artifact and its manipulation and analysis on a realistic scale and in a realistic environment. Further, it provided insight that directed later development of the approach and tools in specific ways.

2 Related Work

A number of other researchers have investigated the application of knowledge about language to the problem of requirements validity. Goguen and Potts advocate forms of naturalistic inquiry as it is practiced in social and anthropological endeavors. Naturalistic inquiry involves a data collection process whereby an analyst observes participants in a community and pieces together a picture of their interaction through various techniques including interviewing and artifact collection [16]. The goal is generally to

expose the “...meanings and tacit understandings that participants in social contexts negotiate and derive from interactions with the artifacts and other participants there” [16]. Potts proceeds in a direction that uses observations of human interaction to motivate strategies to enhance communication. For example, he proposes a structured conversation model for communicating requirements based on what he calls the *inquiry cycle*, the patterns according to which humans iteratively request and process information in constructing new understanding [17].

The idea that domain knowledge needs to be documented in a specification has been proposed by others, e.g., the *text macros* of Heninger [8] and the *designations* of Zave and Jackson [19], but without a theoretical basis. Leveson recognizes the need to capture expert intent and proposes structures in which to record it, but applies psychology only at the level of compound, high-order activities, and does not suggest methods to connect meanings to the elements in a model [12]. Even so, this approach produces complex artifacts, and the complexity is difficult to manage without automated support. She has thus enhanced the SpecTRM toolset to support the approach [13].

Berry explicitly recognizes the need for both elements: attention to the cognitive mechanisms that allow semantic generation and comprehension as well as to the relationship of semantics to the forms in a model [1]. However, both his problem characterization and solution approach can be enhanced with the rigor available in psychological and linguistic methods.

Maiden and Hare have employed an element of psycholinguistic theory to improving the efficacy of certain activities that contribute to their work in problem domains. For example, they used a card sorting technique to elicit facts about the storage organization of their subjects in order to improve the modeling of domain information in a database of reusable domain models [15].

While others have thus applied psychology and linguistics at some level and to various elements of the requirements problem, our approach differs in that it is thoroughly based on psycholinguistic theories that account for the low-level mechanics of language comprehension and production. In previous work, we examined how the ways in which humans innately use natural language result in statements of requirements that are incomplete, inconsistent, and open to misinterpretation [6, 5]. To support this finding, we exploited results from cognitive linguistics that detail the ways in which humans organize and communicate conceptual information. We extended this model to account for the breakdown that occurs in communication of information across boundaries of domain expertise, breakdown that is implicated as a major limiting factor of the quality of large and complex software systems [2].

Our goal was to develop an approach that injected more rigor into getting from the initial idea for a software element to the eventual model of its structure and function, upon which implementations would be based. We wanted to be able to do so demonstrably, predictably and repeatably. Specific findings of the linguistic analysis dictated that our approach have certain properties. The approach developed entails the introduction of an abstraction called the *domain map* that can be used to organize and document for non-experts critical application domain information. The need to manage the complexity of represented entities within our approach motivated the work that led to the toolset described in this paper.

To review, others have applied psychology and linguistics to some extent to the requirements problem. Our approach seeks to do so comprehensively and rigorously, which, because of the complexity generated, motivates automated support. This support then enables not only practical usability, but experience that advances development of the complete approach.

3 The domain map

We have argued previously that developers, who are non-experts in a particular domain, 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 [6, 5]. This implies the need that this information be processed into a form that is both cognitively accessible to developers and agreed with and among domain experts. The domain map is a structure designed to allow explicit and systematic access to essential domain semantics in a form that is meaningful to those who do not possess expert knowledge of the domain in question.

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.

The linguistic analysis indicated at least three areas in which a structured approach built around an artifact held promise to reduce miscommunication. Based on the analysis, the domain map artifact was designed to provide mechanisms to do the following:

- reduce the ease with which a non-expert may make an assumption about the meaning of a domain-specific term, by providing directed explication,
- account for the recursive nature of understanding, by grounding explications of all domain-specific terms directly or indirectly in terms that are in the common lexicon, and
- provide characterization mechanisms that alone or in collection with metrics collected from other instances of the artifact can allow visualization of trends and aid in allocation of resources.

3.1 Artifact Definition

More specifically, then, a domain map is a structure that documents the conversion of a domain specific, explicit representation of essential semantics to a form accessible to those with a common base set of representations. This entity in effect maps 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 respective storage structures and content possessed by any two people within a project are essentially the same with regard to salient linguistic elements [6, 5]. *Domain*, then, refers simply to all

terms that are not *common*; if a term has domain-specific meaning, its storage structure and/or the content it contains differ for any two people from within and outside the domain, respectively.

To better serve its purpose of documenting this mapping, constraints are placed on the domain map such that the following properties are observed:

- A requirements 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.

A domain map, then, at the highest level, is a structure that, from a set of terms drawn from a natural language requirements source, documents a partition of this set into *domain* and *common* terms and explicitly relates members of these partitions to each other, supplementing with additional *domain* and *common* terms where necessary in order to construct a completely *common* representation for every *domain* term.

Further, once a domain map is instantiated, it can provide additional value, since the artifact described can be subjected to analysis. The definition of any term or phrase in the original set describes a tree of dependency in which the root and all of the remaining non-terminal nodes are domain terms, and all of the terminal nodes are common terms (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). 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 whether the domain map is legal, 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.

For further discussion of the domain map artifact, the reader is directed to [4, 6, 5].

3.2 Validation of the Artifact

As an example of the issues described, consider the following. In a small-scale study we undertook during the development of the domain map concept, developers were asked to work from an industrial natural language specification for a maritime track control system standard [9]. Among the quantities that were modeled was a ship's *bearing*, which was not defined within the specification, i.e., the original specification was, as they often are, incomplete. This left the researcher conducting the study to rely on her assumptions, based on her previous experience with things named

bearing, which turned out to be erroneous with regard to the system context. Upon construction of a domain map, however, not only was the term associated with a common language definition that corrected the developer's misunderstanding, but dependencies in which it participated were also made apparent. The domain map showed explicitly that the correct understanding of over 37% of the 102 terms defined in this case *depended on* a correct understanding of the term *bearing*. This demonstrates just how important it is not only to fully expand definitions to common representations, but to recognize that the misunderstanding of a single term can have a tremendous impact on a developer's conception of an entire domain.

In a more rigorous test in a controlled environment, the domain map was further demonstrated to contribute positively to higher-integrity communication of domain-specific semantics. 114 subjects participated in a 2-group true-random experiment in which the control group was provided with a set of industrial natural language requirements and the experimental group was provided with the same natural language requirements complemented by a domain map. All members of both groups took a diagnostic test of domain-specific knowledge pertaining to the requirements. The experimental group performed significantly better overall, scoring higher on 7 out of 8 questions, and by more than 50% on 3 [4].

What the domain map provides is an organized and structured entity that documents essential domain knowledge. 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, 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. 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.

While structures like the domain map that instantiate linguistically founded abstractions are conceivable and manually realizable, in practice there are severe limits on the value achievable through strictly manual means. These limits derive from size and complexity of the generated artifacts, but with automation, the representable complexity of such structures grows and further forms of processing can be performed. We next present a toolset that directs the creation of significant added value to and from such representations; this is value that is not predictably or systematically achievable without both the theoretical basis upon which the structures are founded as well as the automation of processes that the structures allow.

4 The Toolset

As illustrated in Figure 1, our goal for the toolset was to provide a support mechanism for applying the linguistic model, via the domain map, to the creation of early lifecycle artifacts such as requirements documents and specifications. Linguistically sound artifacts can be instantiated manually, but these instantiations are necessarily of limited complexity. Further, manually creating and analyzing even a relatively non-complex domain map is both time consuming and difficult. Developers must sift through all

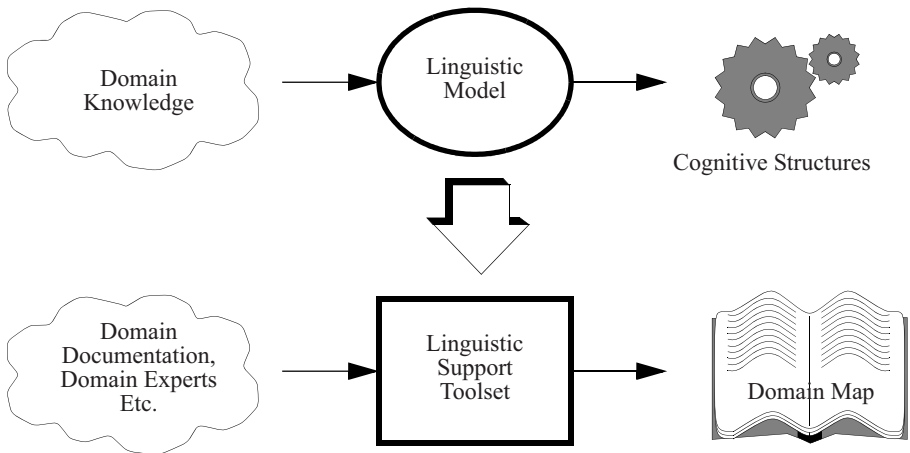


Fig. 1. Toolset goal.

material to ensure that all terms at issue have been identified, that they have each been provided with grounded and agreed-upon definitions, that all definitions are analyzed for legality, that overall organization and usability of the information is maintained, and so on. For a realistic requirements specification, this information and its organization can quickly become exhaustingly complex. Further, any manual analysis can be tedious and error-prone, and when done sufficiently rigorously to be valuable, it is likely to be prohibitively costly.

Clearly, automated support for the initial creation of the domain map and execution of the various forms of analysis that are possible is an attractive idea. More importantly, in addition to providing efficiency and reducing errors in processing, tools could allow the construction of representations that are more complex, as well as more realistically useful kinds of analysis capability than can be achieved manually. The following specific goals guided our support strategy:

- *Complexity management.*

A key requirement for the toolset is to permit the creation of a domain map in a manner that embodies critical components of the linguistic theory as they apply to the problem. Our analysis indicated that lack of explicit attention to facts of semantic storage structure allowed breakdown in communication. The tool must provide effective control and management of the data elements representing these facts and their relationships as the structure grows in size, as well as facile alternations between related but separate views.

- *Completeness and Consistency analysis.*

The domain map must be complete in the sense that all terms indicated to be domain-specific by the linguistic model are present and defined appropriately. “Appropriately” refers to one of three possible forms: (a) an explicit definition using only common terms, (b) an explicit definition using common and domain-

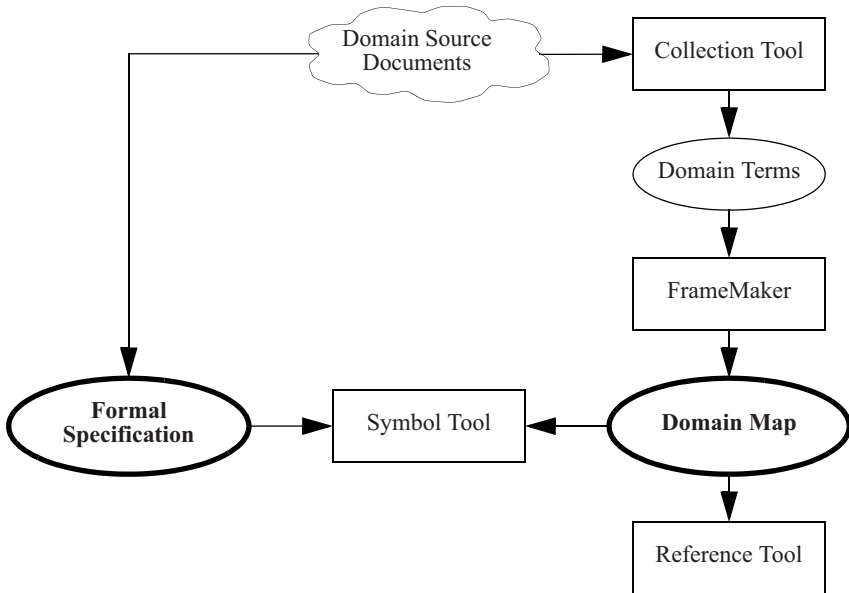


Fig. 2. Support toolset.

specific terms (themselves defined appropriately), or (c) a link to a domain-specific formalism. The tool must examine definitions for compliance with these forms. It must further examine for consistency the resulting graph structure that relates the definitions, checking for circularities and redefinitions.

- *Human Direction*

The tool must provide direction to users in creation, refinement, and analysis of the structure. For example, upon selecting a term for definition from source materials, the tool should create the appropriate fields and guide the user in completing them. Further, it should systematically organize sets of situations where human judgment calls are necessary, such that users are more likely to resolve such situations. For example, it should collect and display location information for all instances of a term so that a user can examine instances in context for consistent usage. The tools should also provide a method of enforcing that processes are carried out, possibly in the form of requiring confirmation before proclaiming the structure to be legal according to all embedded constraints.

To enable support for these activities, we have enhanced an existing toolset, Zeus [10], to provide facilities for systematic direction of the creation, refinement and analysis of domain maps, as well as possibilities for their integration with other lifecycle artifacts. The tools and their relationship to various entities are shown in Figure 2.

Zeus is based on the FrameMaker desktop publishing system and the Z/EVES verification system. It provides comprehensive facilities for manipulation of both natural language and Z [10]. Its support for natural language is used in two ways. The primary use is to create and modify formal specifications that integrate formal elements written in Z with natural language that is organized to exploit linguistic theory. It can also be

used to manipulate project and domain-related documents in whatever format domain experts and developers require using all of the facilities of unmodified FrameMaker.

4.1 Collection Tool

In order to support the creation of domain maps, Zeus has been extended with a facility that allows the collection of domain-specific terms and phrases. The creation of the list of domain terms and phrases is undertaken by developers and domain experts who make a number of passes through source materials. For example, a scenario we have exercised is one in which a developer makes the first pass through an original requirements draft, indicating words and phrases he either outright does not understand, or has reason to believe might have domain-specific meaning of which he is not aware. Then a domain expert makes a pass through the same document, annotated with the developer's indications. The domain expert thus has a chance to validate the developer's indications as well as add to them. This trade of active role can be iterated as necessary for the involved parties to converge on a set of terms in need of semantic accessibility from outside of the domain. Further, in the version of the tool under development, terms and phrases of interest, as well as descriptive information pertaining to them, can also be collected directly from structured artifacts that result from an elicitation process we have since defined.

The Zeus *Collection Tool* enables the above described passes through natural language requirements and the capturing of terms and phrases to be entered into the domain set. The mechanism is the selection (highlighting) of a term while in capture mode, followed by a keystroke, which creates a blank domain map entry for the term. The list of domain terms can be manipulated as different passes are made over the source materials, and natural language definitions, again negotiated by the involved parties, are entered for domain terms using the text manipulation facilities of FrameMaker. Thus the construction of an initial domain map abides by the necessary informality imposed by the development of ideas and their expression in natural language, but the Zeus tool guides navigation and systematic exploration of this informal space by providing the artifact structure and operations that allow explicit accounting for critical elements.

4.2 Reference Tool

Refinement and use of the domain map are supported by the *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 the left portion of Figure 3. 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. Further, it displays the definition of any term if the right mouse button is clicked over it (right portion of Figure 3), and if there are any additional domain terms in a definition, they are indicated with underlining.

The reference tool performs a variety of analyses on the domain map as the display is built, allowing refinement from a draft to a legal form. 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

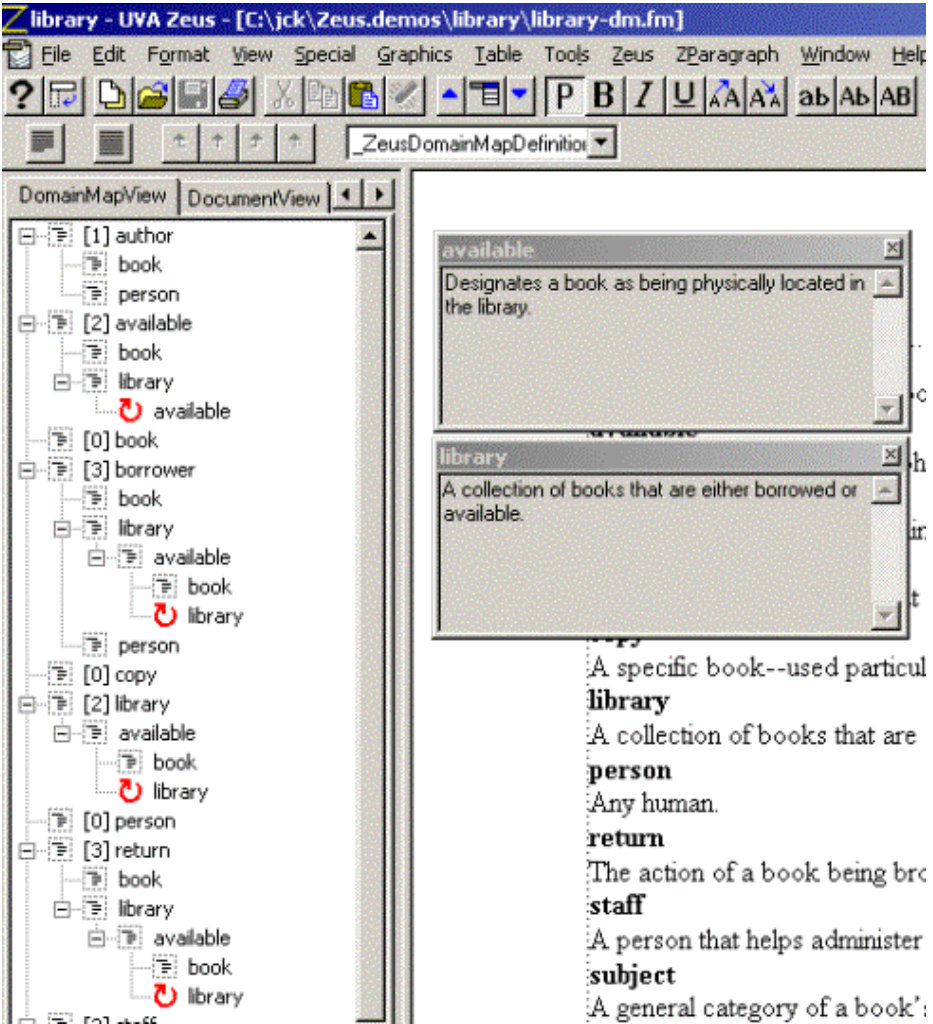


Fig. 3. Multi-framed view of dependencies, editing window, and pop-up definitions.

definitions can be refined to comply with the no-cycle rule (note examples in Figure 3 marked by circular arrows). Further, it checks the dependencies for all definitions and ensures that each is completely grounded in common terms.

Finally, the reference tool computes various metrics from the domain map, for example, the distribution of the number of levels required to ground domain terms (that is, the distribution of the depths of dependency) and the number of defined terms or phrases relative to the total word count for a requirements document. These metrics are useful in gaining a picture of the semantic complexity of individual terms and entire lexicons, allowing intuition regarding the most efficient application of resources.

4.3 Symbol Tool

If a specification includes a formal component, the symbols in that component can be associated with the contextual information provided by the domain map. The connection between formal symbols and their referents in the real world is complex [6, 5], but typically, symbols in a formalism are labeled with mnemonic identifiers that implicitly stand for entities in the world, and a user is left to interpret the identifiers within the limits of his own experience of the terms used. A third tool within Zeus, the *Symbol Tool*, can check whether each mnemonic possesses a corresponding, explicit entry in a domain map and is thereby intentionally grounded. Any identifier not linked to an entry is flagged. Zeus currently supports this function for Z specifications, and could be expanded to include other notations and design tools with formal elements, such as UML or Visio.

5 Feasibility Study

During development of the domain map approach and the supporting toolset, we undertook a number of activities to validate the progress being made. We were fortunate to have access to a set of natural language requirements describing a standard for maritime track control systems, which proved useful as an early testbed application. The results addressed in section 3 derive from this experience. However, the first case study was of small scale, performed primarily manually, and used to assess practicality of the approach and identify the necessary functionality of automated support. The controlled experiment assessed primarily the efficacy of the domain map artifact, but while prototype tools were used in the construction of experiment materials, the hypothesis under examination did not involve them. The functionality determined during the case study and prototyped for use in developing experiment materials was then implemented as a toolset, and to give the toolset a more realistic trial, we sought a more realistic environment.

This environment was provided for us by an industrial collaborator. Our collaborator allowed us access to development materials for a medical device, as well as access to development personnel familiar with the project. The purpose of the device we studied is to generate surgical planning information to be used by physicians in determining a course of therapy. We restricted our inquiry to a part of the system that calculates specific therapeutic effects. We began from the set of materials that constituted the requirements and specification for this element from which developers were expected to produce designs and implementations.

The Zeus toolset automated much of the construction and refinement of the domain map. In particular, it provided the necessary complexity management services thereby facilitating the efficient and complete development of the domain map for this application. The use of the completeness and consistency mechanisms enabled rapid checking of the form of the domain map and its essential properties. Zeus fulfills the goals stated in the previous section of complexity management, completeness and consistency analysis, and human direction. Using the toolset in an industrial context also generated ideas about how domain maps might be integrated with other lifecycle artifacts, in ways that were discussed earlier.

Notably, the case study forced out issues that were not fully apparent under the controlled circumstances of early approach and tool development. Of particular interest are the method of selection of terms for inclusion in the domain map, and the process by which definitions were constructed. The case study provided the clearest demonstration up to that time that both of these processes were underspecified and ad hoc; foundations were needed and new methods had to be added to the approach. In succeeding work, both of these activities have undergone rigorous (re)definition. In addition, a companion elicitation process, also built on linguistic principles, has been developed that supports and complements the domain map recording and propagation process. As of this writing, Zeus is undergoing enhancement to support these activities.

An issue that we would like to explore in future work is that of expert understanding stored as narrative or procedural memory versus that stored as hierarchical or ontological memory. While the domain map by its nature lends itself to the latter, we recognize that much expert or tacit knowledge is represented by the former. Linguistic theory concerning the semantics of procedural memory supports the existence of cognitive entities called *frames*, *schemas*, and *scripts* [11, 18] that could form the basis of an approach to representing procedural expert knowledge. Since these are compound entities that relate lower-level ones, such as the entities represented in a domain map, an approach to representing procedural knowledge using them could be integrated with the existing artifacts to provide a comprehensive model of the relationships that hold. Tool support would be expected to contribute to the practicality of such an approach.

Thus the experience illuminated both strengths and areas for improvement in both the approach and the tools. In particular, the tools proved to be a powerful aid to constructing and refining the domain map in an organized and efficient manner, and demonstrated that they allow an originally manual approach to scale to more realistic environments. Further, the experience gave the approach its first trial in a realistic setting, generating a number of insights that have since led to modifications in both the approach and the tools.

6 Summary

Effective communication of application domain knowledge among domain experts and software developers is crucial if the quality of requirements for high-consequence systems is to be improved. We developed a strategy to exploit systemic linguistic knowledge about particular aspects of human communication by embodying it in an artifact that can serve to organize and allow automated processing of meaning essential to the communication of critical application domain knowledge with integrity.

Earlier work addressed the cognitive and linguistic barriers to effective communication between domain experts and non-experts. Our approach exploits research in these fields to improve the communication of application domain knowledge. In this paper, we reviewed this approach and introduced tools built to support its deployment in real projects. While manual construction of our theoretically-founded artifacts is possible, there are limits to the complexity that can be represented as well as to the kinds of analysis that can be performed effectively and efficiently. The toolset was designed with the goal of automating not only what was achievable manually, but

enhancing that set of functionality to handle realistic complexity in organization as well as in processing. The toolset directs involved parties in the construction of artifacts that by their structure and content enable the spread of domain knowledge among developers and throughout the lifecycle. The main artifact, a domain map, and its several views, provides documentation of domain knowledge that by design is more complete and consistent than are ad hoc or intuitive attempts at recording this information. The tools further support various forms of analysis of domain maps, providing information that directs their refinement as well as metrics that provide indications of semantic complexity. Further, the tools allow domain maps to be integrated with formal specifications as well as natural language documents that can benefit from such a reference.

We used these tools in collaboration with an industry partner to develop experimental documentation for recording and dissemination of domain knowledge relevant to the design of a medical device. The experience illuminated both strengths and areas for improvement in both the approach and the tools. In particular, the tools proved to be a powerful aid to constructing and refining the domain map in an organized and efficient manner, and demonstrated that they allow an originally manual approach to scale to realistic environments. The exercise provided a proof-of-concept that the construction of such an artifact, one that embodies systemic linguistic knowledge about miscommunication, and provides for a systematic, repeatable, and predictable process, can be realized in an industrial environment and on a useful scale.

We thus conclude that the approach, originally applied on only a small scale, is feasible for realistic projects, and that the combination of a theoretical foundation to motivate a structure and an industrial-strength support tool to direct its instantiation and analysis provides value not before achievable in reducing the incidence and severity of miscommunication. Practical construction and refinement of artifacts, as well as analysis and integration, were made tractable. The result was a set of documents that held value, demonstrated by additions, corrections, and enhancements beyond the original materials, and recognized by members of the collaborating organization. Further the experience generated insights that led to improvement of both the approach and the tools.

Acknowledgements.

It is a pleasure to thank our industrial collaborator for access to their materials and personnel. This work was funded in part by NSF under contract number CCR-0205447 and in part by NASA under contract NAG-1-2290.

References

1. Berry, D.: The Importance of Ignorance in Requirements Engineering. *Journal of Systems and Software* 28 (1995) 179–184
2. Curtis, B., Krasner, H., Iscoe, N.: A Field Study of the Software Design Process for Large Systems. *Communications of the ACM* 31(11) (1988) 1268–1287
3. Goguen, J.: Formality and Informality in Requirements Engineering. *Journal of Systems and Software* 28 (1995) 179–184

4. Hanks, K., Knight, J.: Improving Communication of Critical Domain Knowledge in High-Consequence Software Development: An Empirical Study. Proceedings: 21st International System Safety Conference (2003)
5. Hanks, K., Knight, J., Strunk, E.: A Linguistic Analysis of Requirements Errors and Its Application. University of Virginia Department of Computer Science Technical Report CS-2001-30 (2001)
6. Hanks, K., Knight, J., Strunk, E.: Erroneous Requirements: A Linguistic Basis for Their Occurrence and an Approach to Their Reduction. Proceedings: 26th Annual IEEE NASA Software Engineering Workshop (2001) 115–119
7. Hayhurst, K., Holloway, C.M.: Challenges in Software Aspects of Aerospace Systems. Proceedings: 26th Annual IEEE NASA Software Engineering Workshop (2001) 7–13
8. Heninger, K.: Specifying Requirements for Complex Systems: New Techniques and Their Applications. IEEE Transactions on Software Engineering SE-6(1) (1980) 2–12
9. 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)
10. Knight, J., Hanks, K., Travis, S.: Tool Support for Production Use of Formal Techniques. Proceedings: International Symposium on Software Reliability Engineering (2001)
11. Langacker, R.: Concept, Image, and Symbol: The Cognitive Basis of Grammar. Mouton de Gruyter, Berlin (1990)
12. Leveson, N.: Intent Specifications: An Approach to Building Human-Centered Specifications. IEEE Transactions on Software Engineering 26 (2000) 15–35
13. Leveson, N., Reese, J., Heimdahl, M.: SpecTRM: A CAD Tool for Digital Automation. Proceedings: 17th Digital Avionics Systems Conference (1998)
14. Lutz, R.: Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems. Proceedings: IEEE International Symposium on Requirements Engineering (1993) 126–133
15. Maiden, N., Hare, M.: Problem Domain Categories in Requirements Engineering. Int. J. Human-Computer Studies 49 (1998) 281–304
16. Potts, C., Newstetter, W.: Naturalistic Inquiry and Requirements Engineering: Reconciling Their Theoretical Foundations. Proceedings: IEEE International Symposium on Requirements Engineering (1997) 118–127
17. Potts, C., Takahashi, K., Anton, A. Inquiry-Based Requirements Analysis. IEEE Software 2(11) (1994) 21–32
18. Ungerer, F., Schmid, H.: An Introduction to Cognitive Linguistics. Longman, London (1996)
19. Zave, P., Jackson, M.: Four Dark Corners of Requirements Engineering. ACM Transactions on Software Engineering and Methodology 6(1) (1997) 1–30