

Improving Communication of Critical Domain Knowledge in High-Consequence Software Development: an Empirical Study

K. S. Hanks; University of Virginia; Charlottesville, Virginia

J. C. Knight; University of Virginia; Charlottesville, Virginia

Keywords: requirements, natural language, safety-critical

Abstract

Poor requirements are implicated in a disproportionate number of defects in safety-critical systems, as well as in schedule and cost overruns that further burden resources. To a first approximation, the greatest improvement to safety-critical software quality and to the efficiency of the development process is to be gained if a substantial advance in requirements accuracy can be achieved. We discuss the role of domain knowledge in requirements accuracy, and describe an artifact that embodies properties called for by a linguistic analysis of miscommunication. This artifact is expressly designed to reduce the potential for breakdown in domain knowledge propagation by systematically compensating for normal variability in the ways that humans apply communicative heuristics. We further describe an experiment constructed to investigate the value of this artifact for improving requirements by increasing the potential for domain knowledge comprehension. Results of the experiment indicate that use of our artifact can be an effective strategy for the propagation of domain knowledge with higher fidelity, and thus for a reduction in the number of safety-critical defects and process inefficiencies that result.

Introduction

Errors introduced into software during the early stages of the lifecycle pose two significant challenges to the development of high quality systems. First, errors introduced early are correspondingly more difficult and expensive to correct (refs. 3, 15). In the worst case, they are left uncorrected and for some systems can result in significant loss. Second, there are more *of* them (refs. 3, 16): the requirements stage in particular is implicated as the locus of introduction of more defects than any other. Lutz found that the majority of safety-critical defects in the systems she studied derived from poor requirements (ref. 14), and the Air Force's Rome Laboratory found that the majority of *all* defects they observed derived from poor requirements (ref. 18). Brooks has stated that "[t]he hardest single part of building a software system is deciding precisely what to build" (ref. 2).

To a first approximation, then, the greatest improvement to software quality and to the efficiency of the development process is to be gained if a substantial advance in increasing requirements validity can be achieved. The field has advanced techniques for verification; the real problem appears to remain in producing a valid specification to begin with.

Application domain knowledge plays a crucial role in this problem, since a goal of requirements is domain knowledge transfer. Curtis, Krasner, and Iscoe implicated the "...thin spread of application domain knowledge" as a main limiting factor to software productivity and quality in the design of large systems (ref. 6). This makes requirements a communication problem, as echoed by Hayhurst and Holloway (ref. 10), with domain knowledge the thing being communicated and domain experts and developers doing the communicating. Without accurate conceptions of the real-world semantics relevant to a system, developers often rely on misunderstandings and invalid assumptions about the entities they model, without realizing it.

Propagation of these misunderstandings and invalid assumptions can lead to annoying, expensive, or even catastrophic failures. Curtis, Krasner, and Iscoe report that “[c]ustomer representatives and system engineers complained that implementations had to be changed because development teams had misconceptions of the application domain” (ref. 6), and Lutz states “[i]t is not the internal complexity of a module but the complexity of the module’s connection to its environment that yields the persistent, safety-related errors seen” (ref. 14). To understand this connection, developers must acquire an understanding of relevant domain knowledge.

We previously examined how this breakdown in communication of domain knowledge might occur, and proposed a theory to explain its mechanism based on results from cognitive linguistics (refs. 7, 8). Using insights gained in the process, we suggested the shape of an approach to coping with the challenges inherent in communicating domain knowledge; this approach embodies strategies specifically designed to minimize the breakdown. The process implementing the approach results in an artifact called a *domain map*, and in this work, we discuss the design and results of an experiment undertaken to investigate the value of domain maps for improving requirements.

The structure of the paper is as follows. First, we review the motivation and structure of domain maps. We then discuss the design of an experiment to investigate their value. Finally, we present the results of the experiment and their interpretation.

The Domain Map Artifact

In previous work, we provided an analysis of the domain knowledge communication problem using empirical results from linguistics and cognitive science to explain observed phenomena (refs. 7, 8). This analysis suggested that two organizing principles employed by human cognitive processing are at the crux of the problem of communicating domain knowledge with precision and accuracy. *Cognitive economy* and *hierarchical structure*, which evolved to convey a volume of semantics efficiently and usefully in everyday communication, backfire in the context of domain knowledge communication, because they exploit assumption based on common experience among speakers; this common experience is not present in communication across a domain boundary, for example, in the case where an application domain expert and a software developer must communicate regarding the requirements for a software artifact.

Formal methods are often suggested as a way to reduce ambiguity and increase precision in developing a requirements specification. They are attractive in this regard since they are inherently and entirely rule-based, unambiguous, and analyzable. However, software systems begin as informal notions, and before they can be formalized, developers must be confident that they understand just what it is they are formalizing. We assert that while any useful solution must eventually be formalized in order to be executable, the result is only as valid as the entities that were formalized to begin with. That is, formalization can result in higher precision representations, but these representations can still stand for quite the wrong real-world entities. Formal methods do not address the problem of communicating across a domain boundary.

Thus, the issues surrounding domain knowledge communication must be addressed before and whether or not formal specification is undertaken. To enable this, we proposed a structure called a *domain map* to complement requirements statements from which developers work (refs 7, 8).

The domain map is an artifact designed to allow explicit and systematic access to semantics in a form that is meaningful to those who do not possess expert knowledge of the domain in question. Curtis, Krasner, and Iscoe cited the necessity for better domain knowledge diffusion (ref. 6), and

Heninger's *text macros* (ref. 11), as well as Zave and Jackson's *designations* (ref. 20), were suggested as methods for its documentation. However, neither of these was proposed with a theoretical basis, and thus their structure and application was ad hoc. The structure of the domain map is motivated by specific insights from linguistic analysis, namely, that concepts likely to have high potential for miscommunication can be identified, and that explicit provision of semantics for those concepts can prevent or lessen recourse to assumption and reliance on prior knowledge, which varies with the individual. Systematicity allows domain maps to be both more comprehensive and more targeted in approaching this problem.

The purpose of the domain map is to document the conversion of a domain-specific representation of essential semantics to a form accessible to those with a common base set of representations. The entity must in effect map domain-specific representations to representations composed of common terms and phrases. For this purpose, 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 its storage structure and content possessed respectively by any two people within a project are essentially the same with regard to salient linguistic elements (ref. 7, 8). Domain, then, refers simply to all terms that are not common; if a term has a domain specific meaning, its storage structure and/or the content contained therein differ for any two people on opposite sides of the boundary. (ref. 7, 8).

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

- 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 number of sources can be used to construct the lexicon for a domain map and will include natural language statements of requirements, but are of flexible form. Ideally, a considered written statement is desired, as it is more likely that certain issues have been given attention by virtue of its construction. However, a tape recording or even 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.

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.

Specifically, each entry in the *domain* partition is to be defined in a manner that employs, where possible, terms that are common¹. It is recognized that this is not always directly possible, i.e.,

¹This strategy differs from that of Leite (ref. 13). 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.

that a definition for a given domain-specific term may itself contain domain-specific terms. Thus the path from a domain-specific term to a commonly accessible representation may embody a number of expansions of nested domain terms. To ensure the usefulness of the representation, however, the no-cycle rule enforces termination in a completely common representation. This complete path implements the explicit and considered access to domain semantics required by non-experts who must understand the system. In particular, it recognizes that there are semantic dependencies among elements in a domain that must be acknowledged and accounted for in propagating understanding to non-experts, and it systematically removes opportunity for the kinds of assumption motivated by cognitive economy.

Further, the domain map provides analysis capability not otherwise possible. The definition of any term or phrase in the original set describes a tree² 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. 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. We have extended an existing toolset to enable the construction and visualization of domain maps as well as these forms of analysis (refs. 7, 9, 19).

Experimental Design

Recall that the domain map is designed to provide an organized and structured entity that systematically documents essential domain knowledge such that it is both physically and cognitively accessible to those without domain expertise. In the design discussed below, we present a hypothesis regarding the effect of the application of this technique on the performance of subjects at a comprehension task. In addition, we describe a two-group true random experiment designed to assess the validity of the hypothesis. Finally, we provide an assessment of potential threats to the validity of inferences drawn based on the results of the experiment.

Design of the Experiment: Previous work addressed the linguistic issues that motivate the existence and structure of the domain map artifact (ref. 7, 8). However, its value for reducing the incidence of miscommunication, and therefore, for increasing the efficacy with which essential domain knowledge is propagated, has not yet been experimentally supported. In this section, we describe the design of an experiment intended to evaluate the merits of complementing a set of industrial natural language requirements with a domain map.

We hypothesize that the use of a domain map, in conjunction with other forms of requirements documentation, by a developer in various stages of a software development process can result in a better understanding of essential domain concepts by that developer, as compared to use of existing common methods alone, and thus higher potential for those concepts to be modeled accurately and with validity in the software. To test this hypothesis, we conducted an experiment

²Technically, 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 which we cast the technique as the independent variable and level of comprehension as the dependent variable. We then examined the value of the domain map for improving comprehension of a set of industrial natural language requirements relative to the comprehension achievable given the natural language requirements alone. The requirements used were obtained from an industrial collaborator and describe an international standard with which maritime track control systems must comply in operation and performance. This is a large document, and for practical concerns of the experiment, we extracted a reasonably self-contained excerpt. This excerpt focuses on functional requirements, and the domain knowledge under consideration was constrained to that directly associated with requirements addressed here.

It is necessary, then, to provide a method of measuring comprehension of these requirements; we associate domain-knowledge comprehension with performance on a diagnostic test of the domain-specific information intended to be understood given this particular set. The test was constructed in consultation with an expert in the application domain, familiar with the standard, in order that this intent be represented with validity. The expert confirmed that the questions included in the final version of the test did in fact concern application domain knowledge that any developer tasked with working on such a system should know in order to be able to model entities within the domain with validity and manipulate these modeled entities in a system design and implementation. The test consisted of a total of 12 questions, 8 of which were of closed form, in that there was a clearly correct answer, and 4 of which were open-ended, in that subjects were asked to describe situations that demonstrated integrated understanding. In this paper, we address only the closed form questions, since they provided the opportunity to extract quantitative data.³ The eight closed-form questions used in the experiment are as follows:

- What is an active track? How does it relate specifically to motion of a ship?
- Which factors must be taken into account in order to determine whether an approach maneuver to a track is safe?
- What information does the track control system need in order to calculate dead reckoning position, for example, in the case of position sensor failure?
- Which entities/quantities must be known and/or measured in order to accurately model a ship's bearing?
- What is provided by an EPFS? From where does it get this information?
- What is the system expected to do if a course difference limit is exceeded? For whom is any resulting information intended, that is, who is the user?
- What is the difference between an alarm and an indication?
- If a track control stopped alarm is initiated, to what kind of control does the ship revert? Briefly explain how this other form of control works.

The experiment conformed to a multi-group true random design as described in (ref. 17). Two randomly assigned groups together composed the set of students present in an undergraduate Software class on a particular day during a past semester. Because of their majors, we conjecture that these students are as likely as anyone, and perhaps more, to be in the position of developing software in the near future. 114 total subjects participated, and the two groups of 57 subjects each were physically separated to opposite sides of a lecture hall. All subjects took the same domain-comprehension test and all had at their disposal the same set of industrial natural language requirements. Additionally, members of the experimental group had at their disposal a domain map constructed in cooperation with a domain expert. The subjects were instructed to complete the diagnostic test as accurately as possible during the 45 minute experiment duration, and to use their respective materials in whatever manner and to whatever degree they found them useful.

³Anecdotal data from the open-ended questions will be addressed in an alternate format to be undertaken at a later date.

Conformance of the Experiment to Sound Design Principles: Social research theory sets forth a number of criteria for establishing the existence of a causal relationship between an intervention and an effect: the cause must temporally precede the effect attributed to it, the cause and effect must covary, and there must not exist any plausible alternative explanations for the effect (refs. 1, 5, 17). The third is generally the most difficult criterion to meet, but the experimental methods also provide a taxonomy of threats to the validity of an inference based on experiment. If the potential effects of these threats are accounted for within the experimental design or otherwise, inferences drawn from the results of the experiment are better supported (ref. 17).

The temporal precedence criterion necessary for establishing a causal relationship is met here by design: subjects had access to the materials and were able to use them to whatever degree they wished before answering any question. In addition, if a subject answered a question and later decided to change the answer based on newly discovered information, he was permitted to do so.

The covariance criterion necessary to demonstrate a causal relationship is met if the performance of the group with access to the domain map was on average better than the performance of the group without access to the domain map. This would demonstrate a link between the independent and dependent variables. In the results section, we argue that the performance of the experimental group was in fact better.

A number of alternative explanations for a potential effect, that is, threats to validity, are ruled out or mitigated by design. For example, a *diffusion* threat arises when members of one group learn details about the experimental circumstances of the other group and knowledge of this information affects their performance (ref. 17). If there were a break and members of the two groups were not prevented from discussing with each other the materials to which they had access, information provided within restricted materials might diffuse to those not supposed to have it. This could affect the answers subjects provide on the test. This threat was minimized through the enforcing a “no communication” rule between the groups for the duration of the experiment. Further, the groups were physically separated by a distance greater than that at which any content in one group’s materials could be discerned by a subject in another group. An enumeration of other threats and our response to them is available from the authors.

Thus, the design of the experiment described above is argued to be sound insofar as it is structured to indicate the presence or absence of a causal relationship between the use of a complementary domain map and better performance on a domain-information test, relative to no use of this artifact. The temporal precedence criterion is met by design, the covariance criterion is met if the experimental group performs better than the control group, and known threats to validity are minimized or non-existent.

Results

The structure of the eight closed-form questions was such that a complete answer included two main pieces of information. For example, a correct and complete answer to “What is the system expected to do if a course difference limit is exceeded? For whom is any resulting information intended, that is, who is the user?”, must include indication that an alarm is to be sounded and indication that the user in question is the officer of the watch on the ship. Thus, each question was graded out of a potential 2 points, with each subject receiving a 0, 1, or 2 as appropriate, according to the corresponding number of elements of the correct answer contained in his response. The test was graded by one person, one question across the entire subject pool at a time, for the purpose of consistency in scoring.

Table 1 contains the average raw scores (out of 16) and percentages for each group, for the complete test as well as per question. In addition, the percent difference in performance between the group averages is indicated in the final column. For our definition of comprehension, it is clear from the test data that the experimental group performed better, that is, the group that had access to a domain map in addition to the original natural language requirements was better able to correctly and completely answer questions that were confirmed by a consulting domain expert to be representative of domain knowledge necessary to perform tasks associated with development of this system.

Table 1 – Results

Averages (Raw Score and Percentage) Overall and Per Question for Groups Individually and Combined, and Relative Performance

	Control Group Performance	Experimental Group Performance	Difference (Experimental less Control)
Overall	7.193 (44.96%)	11.895 (74.34%)	29.39%
Q1	1.070 (53.51%)	1.877 (93.86%)	40.35%
Q2	1.561 (78.07%)	1.316 (65.79%)	-12.28%
Q3	.544 (27.19%)	1.596 (79.82%)	52.63%
Q4	.439 (21.93%)	1.526 (76.32%)	54.39%
Q5	1.018 (50.88%)	1.509 (75.44%)	24.56%
Q6	1.228 (61.40%)	1.421 (71.05%)	9.65%
Q7	.474 (23.68%)	1.526 (76.32%)	52.63%
Q8	.860 (42.98%)	1.123 (56.14%)	13.16%

Specifically, the overall performance of the experimental group was over 29% better than that of the control group, achieving an average raw score of nearly 12 out of 16 as compared to the control group's average of just over 7. Further, the experimental group achieved higher scores on 7 of 8 total questions, by greater than 40% in 4 cases and greater than 50% in 3.

We interpret these results to indicate that the domain map did in fact provide essential domain information more clearly and explicitly than did the natural language requirements alone, as opposed to leaving omissions and assumptions about developers' individual knowledge of the domain unaddressed. For example, to answer a question regarding the pieces of information necessary to calculate dead reckoning position, subjects in the experimental group had access to a definition of what dead reckoning meant in the domain in question, this definition was located alphabetically in a comprehensive list of domain terms, and other domain terms upon which its definition relied were indicated and themselves defined. No such definition was provided in the natural language requirements⁴, and comprehension of this essential information was left to observation of use of the term in context, and prior knowledge that the developer was assumed to possess.

⁴The requirements from which the test excerpt was taken did include a short glossary, but neither dead reckoning position, nor most of the other domain terms addressed by the domain map, were included. Further, dependencies among domain terms were not addressed. The excerpt provided to subjects did not include the original glossary, however, it included only one term that might have made a difference to the performance of the control group on one question, had the definition provided essential information. However, the definition was circular, and explicitly included none of the information necessary for a correct answer, that is, subjects would still have needed to resort to assumption or prior knowledge.

Indeed, many of the subjects in the control group did in fact get this answer completely correct, since they were able to use either the context, prior knowledge, or a combination, in deducing what must be necessary to calculate dead reckoning position. Several subjects indicated as much in their answers, using qualifiers such as “Presumably”, “As far as I understand”, and “I would think that”. The issue, however, is precisely the unpredictability of this activity. The fact that many subjects in the control group also showed very little comprehension on this question demonstrates the diversity of domain exposure and reasoning skills possessed by any body of potential developers.

This variety of exposure and skill, often estimated to be at least a 5-fold difference between extremes (refs. 4, 15) has been corroborated as a highly influential factor in software productivity and quality (ref. 6). A useful approach to coping with this variety and its attendant influence is one that in a systematic way reduces the reliance on developers’ prior knowledge and more importantly, on their assumptions regarding what terms mean. Recall that a main finding of the linguistic analysis was that the heuristic of cognitive economy, which works by exploiting the value of making assumptions in known domains, breaks down when communication is across domains, because assumptions naturally made become less and less valid. The domain map is designed to systematically reduce the need to make assumptions, thereby reducing the incidence of invalid ones. Our data support the assertion that use of a domain map in conjunction with this set of requirements reduced the need for subjects to make assumptions or rely on prior domain knowledge, in effect rendering the diversity among subjects less influential in their comprehension of essential domain semantics.

A valid statement from a devil’s advocate position might be, “The experimental group had more information. Of course they were going to do better.” However, first, this only serves to illustrate the point that industrial requirements statements can be incomplete and imprecise. The domain map concept is designed specifically to systematically reduce incompleteness and imprecision. Second, as illustrated by the data from question 2, the experimental group *can* do worse. On this question, the experimental group underperformed relative to the control group by over 12%. On reexamination of the experiment materials, we believe there is a plausible explanation. A correct and complete answer to question 2 was actually clearly and explicitly available in the natural language requirements. This by itself indicates that all subjects had the potential to achieve full credit for this question based on information explicitly provided, with no recourse to assumption or prior knowledge. However, the experimental group in this case was also faced with additional information, meaning more raw quantity to process before making a decision regarding the answer, and thereby the potential for a lack of clarity as to which information was most important or relevant to the answer. Worse, duplicate attention in the documents opens the door to the potential for conflicting information. We believe that the underperformance of the experimental group on this question can be attributed to these factors. Further, this interesting data point provides insight that will be useful in refining the domain map construction process. The domain map is intended definitionally to complement existing resources, not to restate them, and this characterization must be heeded; more is not better when it results in decreasing returns.

Summary

Certain human cognitive heuristics, which are otherwise effective, are implicated in our limited ability to communicate well across domain boundaries. This suggests that domain knowledge essential to producing valid systems would be propagated with more fidelity if these innate heuristics could be suppressed or circumvented. We proposed a structure called the domain map, which was expressly designed to embody properties that counteract such problematic tendencies.

Primarily, it provides explicit and considered access to essential domain semantics, systematically reducing the need for recourse to the unpredictable mechanisms of assumption and prior knowledge. We designed and executed an experiment in which 114 subjects were divided into a control group and an experimental group, and tasked with a diagnostic test of comprehension of the domain knowledge associated with a set of industrial natural language requirements. The control group had only the original requirements at their disposal, while the experimental group had the original requirements as well as a complementary domain map. The experimental group performed substantially better overall, scoring higher in 7 out of 8 questions, in three of them by more than 50%. While the question on which the experimental group scored below the control group indicates that there are refinements to be made to the domain map construction process, we believe this experiment provides strong evidence that systematic compensation for normal human variability can promote the propagation of essential domain knowledge with higher fidelity.

Acknowledgements

We wish to thank Brenden Schubert of the University of Virginia for his assistance on the terminology of maritime systems, and Jonathan Michel of Northrop Grumman for his assistance with the track control specification. We also thank William Greenwell and Elisabeth Strunk of the University of Virginia for their assistance in executing the experiment, as well as the Software students for their participation. This work was funded by NASA under contract NAG-1-2290.

References

1. E. Babbie. The Practice of Social Research. 8th ed. Belmont, CA.: Wadsworth Publishing Company, 1998.
2. F. Brooks. *No silver bullet: essence and accidents of software engineering*. IEEE Computer, 20(4):10-19. April 1987.
3. B. Boehm. *Software engineering*. IEEE Transactions on Computers. C-25(12):1226-1241, December 1976.
4. B. Boehm. Software Engineering Economics. Englewood Cliffs, NJ.: Prentice Hall, 1981.
5. T. Cook and D. Campbell. Quasi-Experimentation: Design and Analysis for Field Settings. Chicago: Rand McNally, 1979.
6. B. Curtis, H. Krasner, and N. Iscoe. *A field study of the software design process for large systems*. Communications of the ACM, 31(11):1268-1287, 1988.
7. K. Hanks, J. Knight, and E. Strunk. *A linguistic analysis of requirements errors and its application*. University of Virginia Department of Computer Science. Technical Report CS-2001-30. November 2001.
8. K. Hanks, J. Knight, and E. Strunk. *Erroneous Requirements: A Linguistic Basis for Their Occurrence and an Approach to Their Reduction*. Proceedings of the IEEE Software Engineering Workshop. November 2001.
9. K. Hanks, J. Knight, E. Strunk, and S. Travis. *Tools Supporting the Communication of Critical Domain Knowledge in High-Consequence Systems Development*. Proceedings of the 22nd International Conference on Computer Safety, Reliability and Security (SAFECOMP 2003). September 2003.
10. K. Hayhurst and C.M. Holloway. *Challenges in software aspects of aerospace systems*. Proceedings of the IEEE Software Engineering Workshop. November 2001.
11. K. Heninger. *Specifying requirements for complex systems: new techniques and their applications*. IEEE Transactions on Software Engineering, SE-6(1):2-12, January 1980.
12. 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)

13. 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.
14. 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.
15. Y. Mizuno. *Software quality improvement*. IEEE Computer, 16(3):66-72, March 1983.
16. B. Potter, J. Sinclair, and D. Till. An Introduction to Formal Specification and Z. London: Prentice Hall, 1996.
17. W. Trochim. The Research Methods Knowledge Base. 11 Mar 2001. <trochim.human.cornell.edu/kb/>.
18. USAF Software Technology Support Center. Guidelines for Successful Acquisition and Management of Software Intensive Systems. Version 3.0, May 2000. <www.stsc.hill.af.mil/stsc-docs.asp>
19. University of Virginia Computer Science Formal Methods Research Group. The Zeus Toolset. 28 February 2002. <www.cs.virginia.edu/zeus>.
20. P. Zave and M. Jackson. *Four dark corners of requirements engineering*. ACM Transactions on Software Engineering and Methodology, 6(1):1-30, January 1997.

Biography

K. S. Hanks, M.A., Doctoral Candidate, University of Virginia, 151 Engineer's Way, Charlottesville, VA 22904-4740, USA, telephone - (434) 982-2225, facsimile - (434) 982-2214, e-mail – ksh4q@cs.virginia.edu.

Kimberly Hanks is completing a Ph.D. in Computer Science at the University of Virginia. She received a B.A. in Russian from Syracuse University and an M.A. in Slavic Linguistics from the University of Virginia. Her research interests are in the application of rigorous theoretical linguistics to problems of communication throughout the software lifecycle for safety-critical and other high-assurance systems. She is particularly interested in the influence of human cognitive mechanics on requirements validity. In addition, she is active in forensic engineering, researching topics that address the influence of cognitive mechanics on failure and how it is investigated.

J. C. Knight, Ph.D., Professor, University of Virginia, 151 Engineer's Way, Charlottesville, VA 22904-4740, USA, telephone - (434) 982-2216, facsimile - (434) 982-2214, e-mail – knight@cs.virginia.edu.

John Knight is a professor of computer science at the University of Virginia. He holds a B.Sc. (Hons) in Mathematics from the Imperial College of Science and Technology (London) and a Ph.D. in Computer Science from the University of Newcastle upon Tyne. Prior to joining the University of Virginia in 1981, he was with NASA's Langley Research Center. Dr. Knight's research interests are in software dependability, and he is currently working on projects in requirements and specification, and the survivability of critical networked infrastructure applications.