

Engineering, Communication, and Safety

John C. Knight and Patrick J. Graydon

Department of Computer Science
University of Virginia
PO Box 400740, Charlottesville, Virginia 22904-4740, U.S.A
{knight|graydon}@cs.virginia.edu

Abstract

Accurate and complete communication between human stakeholders is critical to the successful development of any engineered system. This is particularly significant in the case of safety-critical systems, where incomplete or incorrect communication has the potential to cause great harm. There have been many attempts to address communication in engineering, including the development of formal specification languages and data dictionaries. No one technique is a silver bullet and all come at a cost. For each communication flow, developers must select and employ a combination of techniques that they can justifiably claim is adequate given their system's reliance upon that flow. In this paper, we discuss communication in the engineering process and introduce Assurance Based Communication, a development methodology that helps developers to understand how their systems depend upon communication so that they can select techniques appropriately.

Keywords: communication, assurance, safety case.

1 Introduction

Accurate and complete human-to-human communication between various parties is critical to the successful development of any engineered system. In safety-critical systems, errors in communication can result in crucial defects that lead to failures with serious consequences. In this paper we discuss the general problem of communicating information in software engineering and introduce a new approach to managing communication called *Assurance Based Communication*.

As an example of the very subtle yet serious problems that can arise in communication during software development, consider the Mars Climate Orbiter (Mars Climate Observer Mishap Investigation Board 1999). Amongst other things, the spacecraft's ability to navigate to Mars depended upon estimates of the forces on the spacecraft produced by angular desaturation operations. These forces were calculated by the ground-based SM_FORCES software and recorded in an output file used in trajectory modeling. The adequacy of the

complete system, including the spacecraft, supporting systems, policies and procedures, thus depended in part upon communication between the engineers who built, verified, and validated the SM_FORCES software and the engineers who undertook the trajectory modeling. These engineers needed to have the same understanding of the format and meaning of the data in the file generated by SM_FORCES, and they did not. Despite the existence of a written specification, the SM_FORCES software generated data in English units when metric-encoded units were expected, contributing to the loss of the spacecraft.

The mechanism of choice for most communication in software engineering is *natural language*. It is used for documenting requirements, specifications, development processes, and a myriad of other artifacts. Despite its popularity, the use of natural language brings with it the potential for errors such as subtle misunderstanding. This is well illustrated by the following incident. Software engineers building a controller for a batch chemical reactor were told to "leave all controlled variables constant" in the event that a gearbox oil sensor indicated a low oil level (Kletz 1982). The software they wrote faithfully carried out this instruction according to their understanding of the phrase "controlled variables", i.e., the outputs of the software. Subsequently, when the oil sensor reported a low oil level during operation, the computer kept its outputs to the reactor's actuators unchanged. Unfortunately, the intent of the requirement followed a chemical engineer's understanding of "controlled variables", namely the temperatures, pressures and flow rates of the chemical reaction. Because the controller did not actively manipulate its outputs to keep the reaction variables constant, the reactor overheated and discharged its contents into the atmosphere.

Any given software product has several *stakeholders*, each of which plays a different role, has different interests, and has a different background. Customers want a specific problem solved by a specific time at minimal cost; users want maximum utility, flexibility and ease of use; engineers want minimal development risk; regulatory agents attempt to safeguard the public interest, and so on. Yet all of these stakeholders have to communicate extremely effectively to exchange a wide variety of different types of information in order to ensure a successful outcome.

There are many techniques available to help engineers identify what information needs to be communicated and to reduce the likelihood of misunderstandings. It is understood that each technique brings a given set of

benefits at a given cost; safety-critical systems engineers must choose and employ techniques that reduce the risk of missing or erroneous communication to a level that is appropriate given the potential consequences of a system failure.

Choosing appropriate techniques, however, requires guidance. In this paper, we explore the role of communication in the engineering of safety-critical systems and present *Assurance Based Communication* (ABC), a new view of how that communication can be identified and defined. The basic thesis of this view is that a system’s assurance goals essentially define the criticality of the communication upon which safety relies and thus the necessary assurance together with prior development decisions can drive the selection of communications techniques that need to be used.

2 A Model of Engineering Communication

In order for each stakeholder to contribute to the overall goal of producing a safety-critical system with the correct functionality and adequate dependability, he or she needs to act with the correct information at each stage in a complex and sophisticated development process involving many shared sources of information. This raises the questions of what communication techniques to apply, where to apply them, and what effect each technique will actually have.

One might begin to answer these questions by developing a comprehensive model of the communication that takes place during development. It is easily seen that the various flows of information in a development organization constitute a directed graph which we refer to as the *communications graph*. In this graph, the nodes are the various stakeholders, the arcs link one stakeholder (whom we refer to as the *producer*) to another (whom we refer to as the *consumer*), and the arc labels identify the content of the information flow.

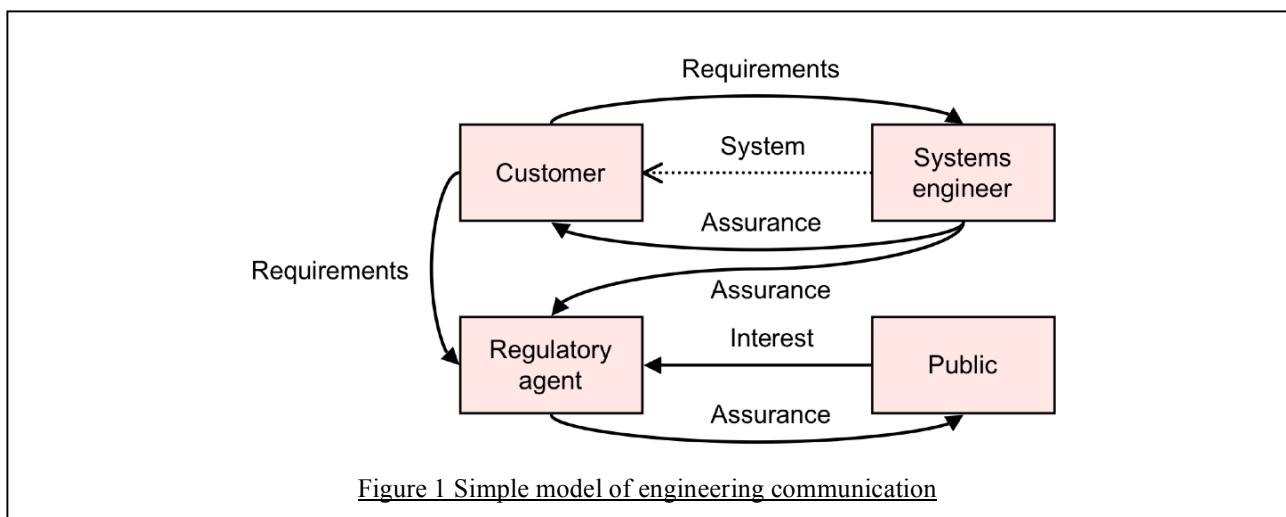
As an example of what the communications graph might look like, consider the very simple and highly abstract version of the communication involving a system’s requirements shown in Figure 1. The customer sends the requirements to a systems engineer who eventually returns the developed system together with various

artifacts (such as test results) which show that the system solves the customer’s problem. In many cases, both the customer and the systems engineer also send information about the system to a regulating agency that gets information from various sources to determine the public’s interest. In this case: (1) the customer, the systems engineer, the regulating agency and the public are all stakeholders in the system; (2) even at this high level of abstraction there are several crucial communications paths; and (3) any defect in any of the communications paths could have serious consequences.

Sometimes communication is planned and conducted carefully, and sometimes it is not. Information flowing in an unplanned manner may indicate an unanticipated yet important communication need. However, developers are unlikely to apply appropriate communications techniques to spontaneous communication flows. Without explicit attention, there is increased risk that the communication will be incomplete, inaccurate, and fragmented. Despite this, in practice developers rarely apply explicit control to help ensure that necessary information flows occur.

Information flow that is planned is still difficult to effect dependably. For many communications that do occur, different stakeholders effectively speak “different” languages. For example, the culture, experience, and jargon of the customer will be very different from those of the systems engineer (recall the incident involving “controlled variables” above). Such differences pervade the set of stakeholders even though there will be some overlap because of the common use of notations such as mathematics.

In practice, efforts to ensure that information is transferred accurately are ad hoc at best, and such efforts are never based on an understanding of the basic principles involved. To the extent that it is accurate in any specific case, the communications graph that we have described identifies the various flows and thereby permits attention to be paid at the level of the complete development process. But it is still not clear what communication techniques should be applied, to which communications they should be applied, and what their effects would be.



With the challenge of addressing these issues in mind, we present Assurance Based Communication in section 5. Before doing so, we examine some of the details of the arcs of the graph in more detail.

3 Threats to Communication

Given the serious impact that defects in communication can have, it is important to try to reduce to the extent possible both the occurrence of these defects and their impact. There are many reasons why the goal of precise communication between any two system stakeholders is not achieved, and in this section we review the various basic threats to communication.

Our goal in communication is to get some engineering information from the mind of the producer to that of the consumer. In trying to meet this goal, miscommunication occurs for a variety of reasons including the following:

- Either the producer or the consumer might misunderstand a term that is part of the *general* and usual lexicon for the domain. For example, a direction might be stated using just the word “North”, the intent being a reference to true North, the usual reference. One of the stakeholders who is not familiar with the convention might associate the direction with magnetic North.
- Either the producer or the consumer might misunderstand a term that is part of the *specific* lexicon of the subject application. For example, a pin number used for a particular output might be stated with the associated port number assumed, but each stakeholder assumes a different port number.
- The producer might omit important detail because he or she assumes that it is either obvious or common knowledge. Every American knows, for example, that there is no need to state that a down switch position is used for “off” in electrical systems. Most Americans are unaware that in some other places, the U.K. for example, the opposite is true.
- The producer might omit information because he or she forgets to include it.
- The producer might include information that is erroneous because he or she is unaware that it is erroneous.
- The representation of information might inhibit complete and precise understanding. For example, if related critical information is not stated in a single location, ensuring that all of it is seen properly by the consumer will be difficult.

Any of these threats could be addressed by casual, ad hoc means, and this is what has tended to happen in practice. However, there is no useful indication of the extent to which the threats would be mitigated by such treatments. Ad hoc approaches are likely to reduce the threats, but a greater understanding of the threats would permit a more comprehensive solution.

In a general sense, the field of *linguistics* is concerned with the use of language in human communication and

with the representation of information. *Cognitive linguistics* is the branch of linguistics concerned with the psychological mechanisms underlying, and the functional use of, language. Cognitive linguistics provides a useful model of the communication problem that we face and also a basis for developing approaches to dealing with the threats.

One model from cognitive linguistics that is especially important is *cognitive economy*. This is a mechanism that has evolved as part of everyday human communication to raise the effective bandwidth that is achieved in routine interaction. Intentionally, most of the terms we use carry a lot of implied information. The word *airplane*, for example, when used in speech (spoken or written) implies many characteristics of that mode of transportation, none of which is stated explicitly. Most people assume a fixed-wing, passenger, commercial transport operating on a scheduled flight when they hear or read the word “airplane”. If this is what is intended, all is well. But if the intent was to speak about a different type of aircraft, as might well occur in specialized domains, then the possibility of one or more serious misunderstandings is clear.

The mechanism of cognitive economy serves us well in informal communication. Without it, we would have to use a lot more words. Nevertheless, it does not serve us well in software engineering. In fact, it is actually the cause of many of the problems that we face. Since it includes assumption as the basis upon which it operates, cognitive economy forces us to make assumptions all the time when communicating about safety-critical systems. If any of those assumptions are wrong, then subtle flaws will have been introduced into one of our crucial communications and the flaws are both very difficult to detect and have unknown effects.

In the next section we discuss a range of possible approaches to dealing with the all of these threats. One of the approaches, CLEAR, is grounded in cognitive linguistics and includes as a fundamental element a systematic approach to dealing with cognitive economy.

4 Combating the Threats

The threats to communication have caused sufficient difficulty with safety-critical systems development that a wide variety of techniques have been developed to try to combat them. The techniques work on the media used for communication (usually documents), and they can be divided into those designed to avoid the introduction of defects into the communications media and those designed to eliminate defects from the communications media.

Simple but useful defect-avoidance techniques include the use of prescribed document formats, ontologies, and data dictionaries. Various forms of document review have been developed to facilitate defect elimination. In the case of documents written in formal languages, defect elimination can be aided by analysis of the document. Various forms of completeness can be checked, for example, thereby allowing mechanical checking of some forms of omission.

Detection and subsequent elimination of defects in communication often occurs as a result of other development activities. For example, it is sometimes the case that testing reveals defects in communicating requirements when unexpected or missing functionality is detected. It is during testing that the customer is likely first to see the system in operation and so noticing unexpected or missing functionality is to be expected. Unfortunately, discovering these problems at this point means that the wrong system had to be built just to discover flaws in the very first significant communication that occurred. Clearly, eliminating defects in communication by means such as this is very expensive because rework is inevitable.

In the limit, defects in communication come to light after a system has failed (perhaps with severe consequences), and the details of the defective communication have to be determined by an investigation. This is the worst possible way in which to detect such defects and provides significant motivation for dealing with the problem.

More sophisticated defect-avoidance techniques include: (a) the systematic and comprehensive use of formal languages; and (b) a technique based on cognitive linguistics called Cognitive Linguistic Elicitation and Representation (CLEAR). We summarize these last two techniques in the remainder of this section.

4.1 Formal Languages

Formal languages, such as Z, PVS, B and Larch, have been thought of as a solution to the difficulty of achieving accurate communication for a long time. It is hypothesized that basing a language on formal (mathematical) semantics will enable accurate communication because the language semantics will be completely defined and identical for all parties involved. For example, the notion of formal specification in safety-critical system in which the system specification is stated in a formal language has been thought of as a way to “get the specification right.”

When communicating a purely formal statement, a rare event, this hypothesis is true. However, it is *not* true for the vast majority of crucial communications that are needed in the development of safety-critical systems. It is important to appreciate that the issue here is not that we need better formal languages. The issue is that we cannot and never will be able to communicate purely using formal languages. It is impossible.

The reason for this impossibility is that formal languages are closed, i.e., they have no inherent meaning. Almost any formal language is merely a collection of symbols and a set of rules for manipulating them. Giving meaning to the logic requires that it be linked to real-world concepts and objects using the only available mechanism for this purpose, natural language.

As an example of this issue, consider propositions. Propositions only acquire a meaning when the associated propositional variables are either natural language text or pointers to natural language. From the perspective of the logic, the following two propositions are entirely

equivalent yet only one would be regarded as being useful in communication:

$$\begin{array}{l} a \qquad \qquad \quad \wedge \quad b \qquad \qquad \quad \Rightarrow \quad c \\ \text{is_raining} \quad \wedge \quad \text{depart} \Rightarrow \text{take_umbrella} \end{array}$$

This situation presents something of a dilemma. Natural language is known to be problematic in various ways and replacing it would be very desirable. Yet it cannot be replaced entirely. Many of its uses in communication can be better documented by formalisms, but the link between formal statements and the intended real-world meaning has to be stated in natural language, and that situation will not change. This means that the threats to communication listed in the previous section cannot be dealt with by eliminating natural language, i.e., using a purely formal approach. The exact mechanism by which a formal statement can be linked properly and completely to its real-world meaning is the subject of study. Nevertheless, at this point, it is necessary to settle for the inevitable use of natural language in communicating information.

4.2 Cognitive Linguistic Elicitation and Representation

Having been affected by erroneous and missing definitions, developers have adopted ideas such as the creation of a *data dictionary* to focus attention on the problem. A data dictionary is a list of all the external data items to which a system refers together with “definitions” of what those data items mean for the system. A data dictionary, though systematic in its treatment of the data items, remains an ad hoc solution to the problem because it does not apply any systematic approach to the form, content or creation of the definitions that it includes. The result is definitions that are sometimes circular, sometimes obscure, and often inaccurate.

Cognitive Linguistic Elicitation and Representation (CLEAR) is a technique that has been developed to capture the essential natural-language meaning of the real-world entities with which the system interacts, i.e., the system’s context (Wasson 2006). For example, electrical switch positions and the meaning of *up* versus *down* are captured as part of context. The technique employs results from cognitive linguistics to explain the mechanisms by which communication breaks down in engineering. It includes a process that provides procedural support for the elicitation and representation of context information in which specific threats to communicative fidelity are reduced.

CLEAR produces a *CLEAR Knowledge Base* (CKB) for a specific system that acts as a reference for all stakeholders involved in building the system. As such it provides detailed explications of terms and phrases that are essential to the understanding of how the system is built and how it interacts with its operating environment.

The process by which the CKB is built provides a high degree of confidence that many common problems that arise in natural-language communications have been reduced significantly. The CLEAR process is partly manual and partly automated. It begins by selecting existing project documents and then scans them to provide an initial set of candidate terms and phrases for

inclusion in the knowledge base. This is followed by various manual steps that include:

- assessment of the completeness of the set of terms and phrases by inspection;
- a categorization of the relative importance of the terms and phrases;
- the development of a category graph (a structure that psychologists believe represents the mechanism by which humans store and organize semantics); and
- the creation and analysis of the necessary explications.

The CLEAR process produces a set of explications in which some explications depend on others in a tree of dependencies. The depth of this tree varies with the application. The structure of explications, the process for their construction, and the support available from tools ensures that explications are free from circular dependencies, obscurity, and various other defects.

The leaves of the tree of explications are referred to as *common terms* meaning that they are assumed to be understood correctly by all stakeholders. Any application-dependent terms and phrases that are included in an explication can be replaced with their own explications, and this process can be applied until any explication is expressed entirely in common terms. The exact nature of these common terms depends on the application domain, the variety of stakeholder roles, the experience of the stakeholders, and so on.

Finally, CLEAR combats cognitive economy by seeking details of the various attributes (the things we normally assume) associated with all of the terms in the CKB and documenting these attributes explicitly even if they are perceived to be “obvious”. In this way incorrect assumptions about terms used in the system’s context description are significantly reduced because locating them is no longer an ad hoc activity.

5 Assurance Based Communication

If constructed carefully, a communications graph of the type that we introduced in section 2 provides a comprehensive picture of all the communication that is expected to occur in the development of a system. This graph will be similar from one project to the next, so it is likely that the graph for a new project could be developed from some standard template. The communications graph as presented so far has three significant limitations:

1. Even if the graph for a specific project were developed from a template, there is no way of being sure that all the necessary paths have been documented nor that all of the information needed by any particular stakeholder will actually be provided to him or her.
2. Much worse than omission of a link is the possibility that any specific communication will not succeed as needed. Our emphasis in the communications graph is to document the producer, the consumer and the entity used for transmission (typically a document of

some sort). There is no element of the communications graph that indicates the potential for a breakdown in the path from the document to the consumer’s understanding. Communicative fidelity is routinely assessed in disciplines such as education, where it is necessary to determine whether a student has properly understood the material presented by an instructor, and has even been measured in some software engineering research (Hanks et al 2003). Nevertheless, practicing engineers do not typically make an explicit determination of whether the consumer actually acquired all the information except in so far as the consumer generates his or her own questions.

3. The techniques discussed in section 4 are very useful and could be used to provide an implementation of the links in the communications graph, but each addresses just one element of the overall communications problem. A CLEAR Knowledge Base, for example, allows the context within which a system operates to be defined with high fidelity, but it provides no direct support for software development steps such as requirements elicitation.

Assurance Based Communication (ABC) is a technique for addressing the various limitations that we have noted in the communications graph. It is part of a comprehensive approach to safety-critical software engineering that we are developing called the *Assurance Based Lifecycle* (ABL).

We begin this section with a discussion of the rationale for assurance and its place in the ABL. A major component of the ABL is *Assurance Based Development* (ABD), an assurance-based technique for software development, and we summarize ABD in section 5.2. We describe ABC in section 5.3.

5.1 Assurance and Development

In many domains, engineering activities are governed by standards, many of which are *prescriptive*. The goal of such standards is to ensure that systems have desirable properties (such as being adequately safe) by prescribing a process for developing the system. The assumption underlying such standards is that all systems developed according to the prescription have the desired property. A system developer seeking to assure stakeholders that a developed system is adequately safe, for example, might thus present them with evidence that the system was built according to the prescription offered by an accepted safety standard such as IEC 61508 (2005).

The problem with this approach is that the underlying warrant—that all systems built according to the standard have the desired property—is rarely argued convincingly. Indeed, it may not be possible to argue that this is always true. An alternative approach that has gained particular favor in the UK is *assurance argumentation*. Developers of any system used in the UK defense sector, for example, are required to present a written argument along with their system which seeks to convince a skeptical audience that their systems are adequately safe to operate in the intended operating context. In case of safety-

critical systems the written argument is usually referred to as a *safety case* (Kelly 2004).

In previous work, we have suggested that the entire system development process should be driven by the system's assurance goals using a process called Assurance Based Development (ABD) (Graydon et al 2007). In ABD the choice of system development techniques is determined by the need for *assurance* that the system solves the problem it is meant to solve in the context in which it is meant to operate. We include all stakeholders' interests including the public's interest in the system as part of our notion of context, so that the probabilities of system failures that would endanger lives, the environment or expensive equipment must be shown to be adequately small.

Viewed in this way, the problem facing engineers is not "how do we build a system that solves the problem?" but rather "how do we build a system that we can convincingly argue solves the problem?" In the process of creating that assurance, engineers create systems that solve the problem.

5.2 Assurance Based Development

Safeguarding the public interest by requiring developers to produce assurance arguments rather than comply with prescriptive standards offers several benefits to developers. Developers of systems that are radically different from those that have come before, for example, are free to use whatever techniques they find most convenient for providing the necessary assurance; they are not bound by a prescription that may have been based on assumptions that are not true of their particular product.

One drawback of using assurance cases rather than prescriptive standards to guide development is the lack of process direction for developers. Developers who are bound by a prescriptive standard know precisely what they must do. When developers are free to provide assurance in any way they choose, however, they must pick each element of the development process with little guidance. This approach is often unfamiliar to developers, and it can lead to artifacts that do not meet the needs of certification.

Assurance Based Development (ABD) is intended to provide guidance to those obligated to produce both a system and its assurance argument. ABD is based on the premise that the assurance obligations incident on each part of the system can be used to guide process choices throughout development.

At each step in an Assurance Based Development, the evolving assurance argument (for a safety-critical system this will very likely be a safety case) presents the developer with a set of properties that must be assured. The developer then elicits potential system development choices that might result in the required assurance, evaluates them to determine whether they are likely to provide this assurance in his particular case, chooses one, employs it, and incorporates an argument fragment

associated with the choice into the overall evolving system assurance argument.

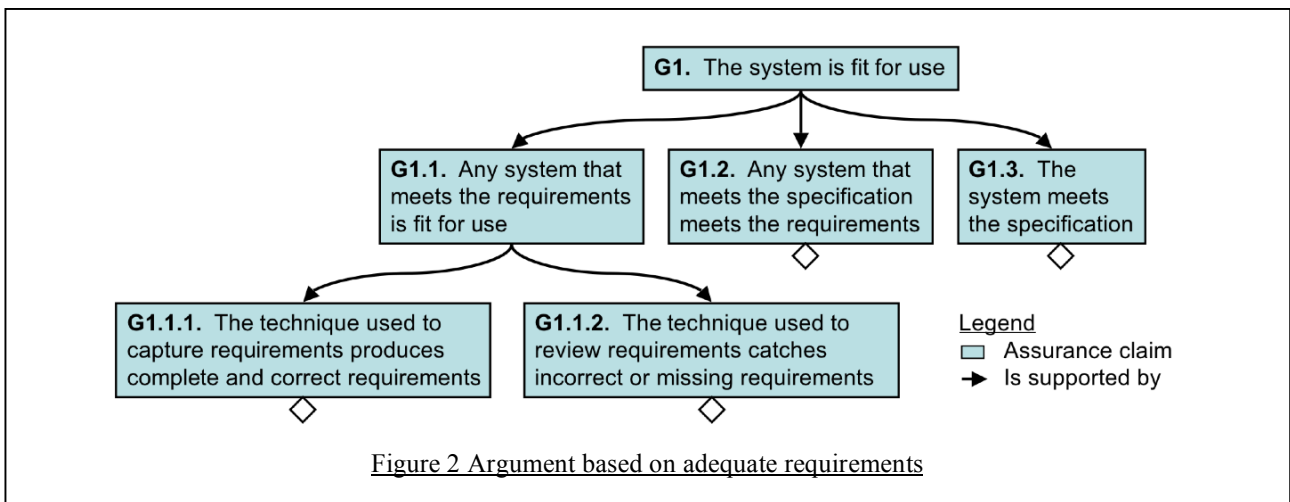
As an example, consider the early stages of a project that is developing a safety-critical system. The developer might consider alternative system architectures, but the selection will be of the architecture most likely to enable the assurance goals to be met. Once the choice is made, it is recorded in an architecture description, and the evolving safety case is modified to show how responsibility for meeting the system's safety (assurance) goals is divided over the components in the chosen architecture. By keeping the developer aware of the assurance obligations incident upon each part of the system and the effect of development decisions upon the assurance argument, the process helps the developer to avoid both development choices that do not provide sufficient evidence of dependability and development choices that provide superfluous evidence at extra cost.

5.3 Assurance Based Communication

Since deficient communication between stakeholders can threaten the ability of an engineered system to adequately solve the problem it is meant to solve, by definition the adequacy of communication *must* be addressed as part of the system assurance argument. Presently this is not the case. Although safety cases document a wide variety of issues, including inadequacy of artifacts such as requirements documents, typically safety cases do not include specific argument elements relating to human-to-human communication.

Assurance Based Communication complements Assurance Based Development by addressing the *quality* of communication between stakeholders overtly. The concept is to establish each communication link and the associated mechanism during development based on the assurance goals that have to be met by that particular communication. The communication graph begins as an empty graph and arcs are added as the need for the communication is derived from the assurance goal. Each assurance goal will itself have been determined by the *system's* assurance goals and that determination will be evident in the evolving safety case. Following determination of the required communication assurance, documentation of proper attention to communication is achieved by including communication assurance explicitly in the system assurance argument.

It is important to note both the way in which communication assurance is determined by system assurance and how it can affect system assurance. The validity of any goal in the system safety argument is shown by arguing using the solutions and evidence upon which that goal depends. If part of that evidence involves determination that a document has transferred certain information correctly between stakeholders, then the quality of that communication affects the validity of the argument for that goal. Thus the required quality of the communication, i.e., the assurance associated with the communication, is determined by the argument of which it is a part.



Given this argument structure, it is clear that erroneous communication can affect system assurance. But it is also clear that if communication fidelity is not included explicitly in the safety case argument, the effect of defective communication will not be apparent in the argument.

One might think that paying this level of attention to communication is not necessary because the qualities of the artifacts that result from communication are typically considered. However, as noted in section 5 this is not the case because defects arising from erroneous communication are often not detected until late in the lifecycle if at all. For example, many instances have been reported of defects introduced during requirements-related communication being propagated throughout system development (Lutz et al 1993, Lutz et al 2004).

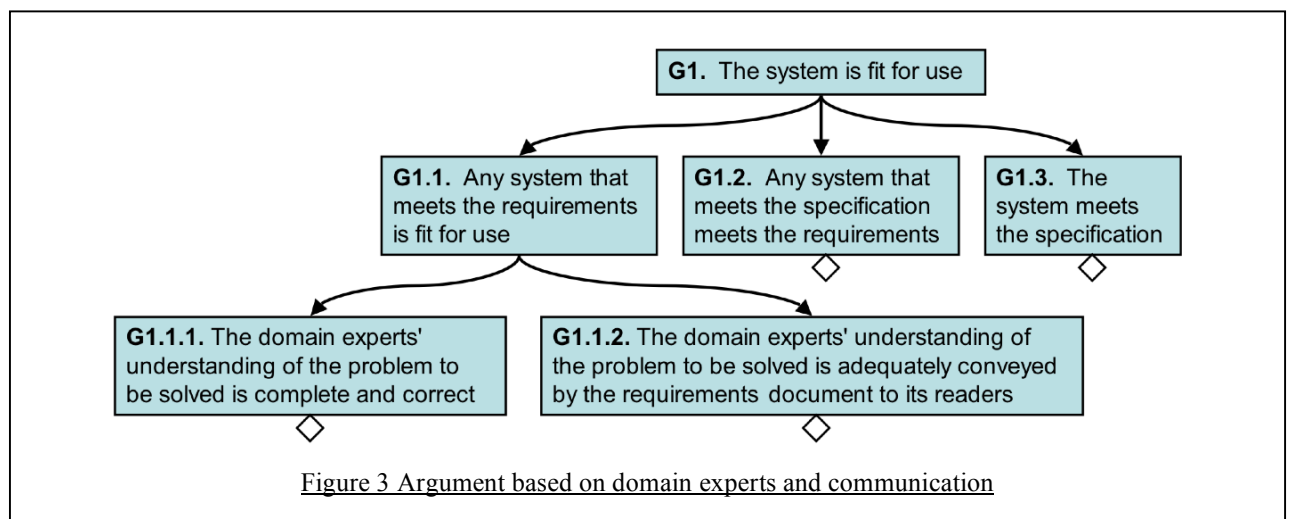
As an example of the need to include communication assurance in the safety case, consider the following scenario. Suppose that a developer argued in a safety case that her software was fit for use in a given context for the reasons shown in Figure 2. This argument fragment claims that the requirements capture technique produces complete and correct requirements and the requirements review catches any defects that may have slipped in during capture. Both techniques, however, involve communication, and if this communication is flawed because of, say, an invalid assumption on the part of

either stakeholder, the claim might not be true *although it is unlikely that either stakeholder will realize the mistake because neither will be aware of the erroneous assumption*. Although the requirements might have been carefully documented, there is no guarantee that the requirements have been properly and completely understood by the consumer as intended by the producer.

Any requirements capture technique is a process for producing a recorded statement of what the system must make true of the world. If the captured requirements do not give rise to the concept intended by the customer, then the developer may produce and have confidence in a system that in fact does the wrong thing. Likewise, the customer's review of the requirements cannot be counted upon to reveal requirements defects unless the customer's understanding of them is the same as the developer's.

Assurance Based Communication deals with the problem in this example in the following way. The developer, knowing that serious threats to engineering communication exist, treats the adequacy of communication as an explicit assurance obligation. This obligation is shown in Figure 3.

The developer would then be faced with the task of constructing a convincing argument in support of claim G1.1.2 in Figure 2. This might be done by arguing the use of a formal language to document the logic of the requirements coupled with the use of CLEAR to



document the context within which the logic operates. These items of evidence would then be included in the safety case and merged into the overall argument.

5.4 Verification of Communication

Having established the use of techniques such as formal languages and CLEAR to provide the mechanism for communication, the next challenge is to verify claims such as G1.1.2 in Figure 2.

In any typical development, the verification of documents such as requirements is undertaken by some form of review or inspection. However, the careful and explicit statement of the goals of communication that we are now including in the safety case rules out relying exclusively on a simple review wherein the domain experts read the requirements document and note any errors and omissions they see. The domain experts, having provided the information in the requirements document in the first place, are ill-suited to judge whether the document would give rise in the mind of other readers to the requirements they intend. Domain specific terms, for example, will almost certainly cause no problem for the domain experts and yet might be subtly misunderstood by the programmers who will write the software.

Consider instead a different form of review loosely based on the concept of Active Reviews (Parnas and Weiss 2001). We will refer to this form of review as a communications review. In a communications review, the domain experts pose questions to readers who must answer them with information gleaned from the document under review. If the readers are not already familiar with the document providing the communication and are representative of the set of people who must read the document as part of the engineering process—we may imagine a selection of programmers, test engineers, and the like—then such a review would go a long way to providing the needed assurance. In the case of our requirements example, the domain experts might ask programmers to read the requirements document and answer questions such as detailed “what if” questions about the spectrum of likely inputs, questions about the functionality that is required in different circumstances, and questions about what the consequences of failure would be if requirements are not met. To the degree to which these questions cover the range of information the producers intended to convey, the ability of readers to successfully answer them provides confidence in the adequacy of the reviewed document.

We note the similarity of both the problem and our solution to the challenge faced by instructors teaching courses. They typically determine the level of understanding of the attendees of the course by asking questions, namely by examination.

As with assurance of safety, the notion of defense in depth could be applied to increase our confidence in the completeness and correctness of communication. The developer could, for example, use a formal requirements notation in addition to requirements reviews. While the use of a formal language alone does not address all threats to the communication of requirements, use of a

formal language in addition to other techniques would provide additional assurance evidence and thus additional confidence.

5.5 Verification of Assurance

A safety case is a document that describes why a system’s developers believe the system they have built is adequately safe for use. It is, therefore, a mechanism of communication between stakeholders, and as such it is subject to the various concerns that have been raised in this paper. Specifically, we need to be sure that the argument presented in a safety case is valid, i.e., its creators built a logically correct argument, and that it communicates that argument correctly.

Because a safety case is an argument, the flaws in a safety case are *fallacies*. In order to provide a basis for discovering such fallacies, a taxonomy of fallacies in safety cases has been developed (Greenwell et al 2006). The taxonomy provides a list of possible fallacies and the forms in which the fallacies might appear in a safety case. With the fallacy taxonomy available, developers, reviewers and certifiers can use it as a check list with which to seek and remove fallacies from safety cases as part of a verification process.

6 Conclusion

Human-to-human communication is fundamental in engineering. In the development of safety-critical systems, we must achieve—and *know* that we have achieved—the best possible communication if we are to have adequate assurance that such systems will operate safely.

The introduction of the notion of a safety case has provided a framework for documenting our belief in the safety of developed systems, and we have argued that it can be used to guide many aspects of system development. The same approach that is used in Assurance Based Development can be used to address engineering communication. In Assurance Based Communication, developers base their approach to engineering communication on the assurance needs of the system being developed. As an Assurance Based Development of the system and its assurance case proceeds, the evolving assurance case provides the developer with both a description of the communication upon which assurance rests and a description of the needed fidelity. With this knowledge, the developer can choose appropriate techniques to address the threat of miscommunication.

Acknowledgements

We thank Kimberly Wasson for her insightful comments on this paper and on cognitive linguistics generally. This work was sponsored by NASA grant NAG1-02103.

References

Graydon P., Knight, J. and Strunk, E. (2007): Assurance Based Development of Critical Systems. *Proc 37th*

International Conference on Dependable Systems and Networks, Edinburgh, Scotland.

Greenwell, W., Knight, J., Holloway, C.M., and Pease, J. (2006): A Taxonomy of Fallacies in System Safety Arguments. *Proc. 24th International System Safety Conference*, Albuquerque, NM, USA.

Hanks, K., and Knight, J. (2003): Improving Communication of Critical Domain Knowledge in High-Consequence Software Development: An Empirical Study. *Proc. 21st International System Safety Conference*, Ottawa, Canada.

IEC 61508: 2005, Functional safety of electrical/electronic/programmable electronic safety-related systems.

Kelly, T.P. (2004): A Systematic Approach to Safety Case Management. *Proc. SAE 2004 World Congress*, Detroit, MI.

Kletz, T.A. (1982): Human Problems with Computer Control. *Plant/Operations Progress*, 1(4).

Lutz, R. and Mikulski, I.C. (2003): Resolving Requirements Discovery in Testing and Operations. *Proc. 11th IEEE Requirements Engineering Conference (RE'03)*, Monterey Bay, CA, pp. 33-41.

Lutz, R. and Mikulski, I.C. (2004): Empirical Analysis of Safety-Critical Anomalies during Operations. *IEEE Transactions on Software Engineering*, 30(3) pp. 172-180.

Mars Climate Observer Mishap Investigation Board (1999): *Phase 1 Report*.

Parnas, D.L. and Weiss, D.M. (2001): Active Design Reviews: Principles And Practices. *Software fundamentals: collected papers by David L. Parnas*, Boston: Addison-Wesley Longman Publishing Co., Inc.

Wasson, K.S. (2006): CLEAR Requirements: Improving Validity Using Cognitive Linguistic Elicitation and Representation. Ph.D. Dissertation, University of Virginia.