Using Occurrence Properties of Defect Report Data to Improve Requirements

Kimberly S. Wasson*, Kendra N. Schmid†, Robyn R. Lutz‡, John C. Knight*

*Department of Computer Science
University of Virginia
151 Engineer's Way
Charlottesville
VA 22904-4740
{kwasson|knight}@cs.virginia.edu

†Department of Computer Science
Iowa State University
226 Atanasoff Hall
Ames, IA 50011
schmidk@cs.iastaste.edu

‡Department of Computer Science
Iowa State University
Jet Propulsion Laboratory|Caltech
226 Atanasoff Hall
Ames, IA 50011
rlutz@cs.iastate.ed

Abstract

Defect reports generated for faults found during testing provide a rich source of information regarding problematic phrases used in requirements documents. These reports indicate that faults often derive from instances of ambiguous, incorrect or otherwise deficient language. In this paper, we report on a method combining elements of linguistic theory and information retrieval to guide the discovery of problematic phrases throughout a requirements specification, using defect reports and correction requests generated during testing to seed our detection process. We found that phrases known from these materials to be problematic have occurrence properties in requirements documents that both allow the direction of resources to prioritize their correction, and generate insights characterizing more general locations of difficulty within the requirements. Our findings lead to some recommendations for more efficiently and effectively managing certain natural language issues in the creation and maintenance of requirements specifications.

1. Introduction

Defect reports generated during testing of software systems provide information that can be used to direct improvement and maintenance of the requirements for those systems. This work investigates the potential for using defect report data to identify and manage some requirements communication deficiencies that threaten the validity of high-consequence systems.

High-fidelity communication is critical to ensuring the validity of a software system, since it is the mechanism by which the developers gain the necessary understanding to build the system from those who contract it. Often, the primary communication medium at the requirements stage and throughout the development process is natural language. The problems of ambiguity, imprecision, and inconsistency involved with informal specification of requirements in natural language are well-known. However, since natural language continues to be widely used, the need arises to manage it more effectively.

Defect reports and associated correction requests generated during testing provide evidence of miscommunication events that have occurred earlier in the process. Discovery of these events leads to a number of consequences, including the use of resources for correction of the sources of defects, as well as for rework on parts of the system whose design and construction might have proceeded on the basis of miscommunication.

While the individual instances of deficient language leading to the reported defects might be thus corrected, other instances of the same language throughout the remainder of the requirements documents might go unaddressed. In addition, deficiencies in related language, as well as communicative limitations of the requirements implicated by the defect reports, are rarely considered. This creates the likelihood that further, derivative faults will be discovered later in the process, or even in related systems that reuse these faulty development products. In the worst case, the deficiencies may not be discovered at all, leading potentially to failures during operation.

In this work, we investigate occurrence properties of phrases implicated during testing as problematic, and show how they can help detect a variety of communication deficiencies in the requirements. We here use "phrase" to indicate any string of one or more words, including acronyms and numerals, whose occurrence properties we are interested in examining.

We found that the occurrence properties suggested some strategies for efficiently detecting deficiencies

beyond those directly implicated in the defect reports. We also found that the occurrence properties of the problematic phrases indicated some guidelines for allocating resources for correction and maintenance to areas of the requirements at higher risk for miscommunication events. We thus establish a feed-forward path between problem reports generated during testing and requirements specifications in order to prevent some future requirements-related problems. This investigation is part of a broader effort to understand how requirements faults are created and how they manifest themselves in testing and operations.

Our results indicate that a variety of classes of communication deficiency in requirements can be systematically located using information contained in testing reports. These results provide a foundation for the development of methods that focus not just on fault correction and requirements maintenance, but also on fault prevention. In addition, the activities we describe lend themselves naturally to partial automation, suggesting avenues for low-cost integration with existing software processes.

The remainder of the paper is organized as follows. Section 2 provides an overview of related work in requirements involving testing and operations data, as well as work on requirements communication issues and the role of natural language. Section 3 presents theoretical foundations upon which the method is based. Section 4 describes the method and its execution using defect report data and requirements for a particular, real-world system. Section 5 presents the results and their interpretation, and Section 6 summarizes the contributions of this work.

2. Related Work

Incomplete requirements and requirements misunderstandings are the source of many testing and operational anomalies. Such requirements problems are of special concern because they also cause accidents. Hanks, et al. have implicated breakdowns in the communication of domain knowledge as a major cause of requirements defects in high-assurance systems [5]. Weiss, et al. have cited requirements misunderstanding in several recent disasters, stating, "software-related accidents almost always are due to misunderstanding about what the software should do" [15]. Similarly, in summarizing the results of a large defect-analysis study, Leszak, et al. stated that "domain and system knowledge continue to be one of the largest underlying problems in software development" [7]. Curtis, et al. have also implicated the "...thin spread of application domain knowledge" as a main limiting factor to software productivity and quality in the design of large systems [1].

Analysis of defect reports from testing and operations has shown that misunderstanding of requirements and their underlying rationales is a frequent cause of defects. Lauesen and Vinter, for example, looked at defect reports available a few months after a product's release. They found that about half of the defect reports involved requirements defects, with missing requirements being the most-frequent cause [6]. Requirements-related deficiencies were also found to be common causes of safety-critical anomalies during operations [8].

Defect reports have traditionally been used to evaluate software's readiness for release and/or its reliability. Fenton and Ohlsson have described some problems, however, in the use of defect reports as a measure of quality [2].

Another goal of analyzing defect reports from testing is to prevent recurrence of those defects during operations. For example, in a recent study a significant fraction of the anomaly reports turned out to be false-positives [9]. In these cases the software requirements were implemented correctly but the test teams misunderstood how the software was supposed to behave. Anomaly reports caused by misunderstanding of requirements are of concern to projects for two reasons. First, resolving them is costly in that it consumes human resources and test facilities that could be better spent elsewhere. Second, and more importantly, requirements that confuse testers may, if no action is taken to remedy the confusion, also confuse operators, thus adding risk to the deployed system. Remedial action sometimes recommended in these cases includes clarification of the requirements documents.

Analysis of defect reports has proven useful not just for the specific systems studied, but also in terms of insights into how to improve future requirements documents. For example, recent studies show that certain types of requirements appear to be especially prone to misunderstanding [9]. Linguistic analysis of miscommunication as in [4] offers remedial action not just in terms of the current system's requirements specification, but also in terms of avoiding miscommunication-prone language structures in the specification of future systems.

This work relates and extends the results of previous investigations to provide specific guidance in requirements fault correction and maintenance, as well as in fault prevention, based on analysis of information acquired in testing. The next section introduces the main theoretical foundations of the method presented here.

3. Method Foundations

Two areas provide foundations for this paper's analysis of problematic phrases in defect reports: linguistics and information retrieval. Linguistics provides rigorous characterization of a selection of communicative deficiencies, while IR provides a mechanism for visualization and analysis of patterns of occurrence of these deficiencies in requirements.

The *communicative deficiencies* in which we are interested are those that threaten the validity of a system by limiting the fidelity of the communication between customers, domain experts and developers. We are most concerned here with properties that characterize semantic accessibility; that is, the ease with which someone on the receiving end of a communicative transaction is able to arrive at the intended meaning of a phrase presented to him or her.

Software requirements specifications often include glossaries to aid this goal. However, the presence of a glossary does not in itself solve the problem. "An accurate definition should predict an appropriate range of use for a word"; in other words, a "reader should be able to trust that the definition is a reliable guide to how to use the word" [3]. For this purpose, most dictionaries and glossaries fail, as definitions are often too broad, too narrow, or do not otherwise accurately reflect semantic boundaries. Reliable interpretation is even less likely when explicit definitions are not available, as in the case when the glossary is missing or incomplete.

Characterization of a problematic phrase is organized in terms of the availability of meanings that map to it. There should be exactly one available meaning, and that meaning should be of pragmatic utility, that is, actually interpretable. Semantic accessibility is compromised by multiple available meanings, no available meanings, or an available meaning that is of limited pragmatic utility. The first case is exemplified by terminological overload and/or underspecification, the second by lack of definition (explicit or reasonably assumed), and the third by other structural faults in explicit or assumed definitions.

Structural faults in definitions are limitations that inhibit their interpretability, such as circularity or non-predictiveness. *Circularity* refers to a definition that uses one or more words or phrases that themselves are defined, directly or indirectly, in terms of the target word or phrase [3]. A circular definition is non-interpretable because it relies on another definition that is not forthcoming. An example is defining absolute counts and relative counts in terms of each other.

Non-predictiveness is marked by, for example, hedges such as "usually" or "etc." in the text of a definition. These devices prevent the absolute determination of the boundaries of a given concept, and thus render a definition not useful as a discriminator among concepts. An example is the project's phrase "errno". Each of the possible values of errno, which is of enumerated type, maps to a specific error that can occur. The errno is set when errors are detected to indicate which of the possible errors has occurred. However, the phrase was in this project defined only by examples, so was not well understood. This phrase was thus problematic and contributed to reported defects.

In addition to linguistics, information retrieval provides a foundation for analysis of documents with respect to the problematic phrases. In particular, we use the notion of *occurrence properties* as indicators of phenomena of interest. Common indicators include, for example, frequency and distribution of phrases within documents. In IR these indicators are used for indexing and search and retrieval of individual documents within collections [12].

In the work reported here, we are also interested in the occurrence properties of phrases within documents. However, rather than for search and retrieval, we are interested in what these properties can indicate about how the understanding of given phrases affects the quality of requirements documents. For example, do the occurrence properties of a phrase (such as the phrase "reporting period") in the requirements imply anything about how important the concept referred to by that phrase is to the validity of the requirements? Do the phrase's occurrence properties suggest strategies for allocation of resources for correction and maintenance? The specific occurrence properties in which we are interested and how they are used in this process are discussed in Section 4.

The work presented here integrates concepts from linguistics and information retrieval and applies them to better understand the relationship between testing and requirements quality. More specifically, it uses occurrence properties of phrases identified in testing are having linguistic deficiencies to guide requirements correction and maintenance. In the next section, we describe the step-by-step process used to implement this method.

4. Methods

This section describes the method for determining and analyzing occurrence properties of problematic phrases. It also describes the application of this method to the defect report data and requirements documentation of a particular real-world system. The section first presents the project artifacts that were used in the investigation. It

then describes the goals of the investigation. Finally, the section lays out the steps in the investigation process.

4.1 Objects of Investigation

We examined requirements and testing materials from a flight-software project to collect, process and format data for a space-borne, narrow-field ultraviolet telescope. The telescope had to accurately perform high-resolution, multi-wavelength observations for the mission to succeed.

The requirements materials examined consisted of two documents associated with the project. The first was a textual software requirements specification. It described the requirements for each of nineteen software modules mapped to components or other functional divisions of the system. The second was a detailed requirements traceability matrix containing 267 textual requirements, each traced to one of the nineteen modules.

The testing materials examined included a defect dataset containing 145 software defect reports generated during testing of the flight software system [10]. The defect reports included identifying information and textual descriptions of each defect. Further description, including information about the causes of the defect (sometimes speculative) and the corrective action taken to fix the defect, was available to us in the 37 change requests generated by the 145 defect reports. The change requests were also text-based.

The testing materials (defect reports and change requests) contained a number of fields of interest. In particular, they included a textual description of what was wrong, an indication of whether anything was fixed, where it was fixed, and if not, why not. This information provided the basis for asserting a gap between what was done in practice in response to a defect and what we argue could be profitably done given further analysis of the available information.

For example, text comments in a change request sometimes indicated a misunderstanding of a requirements concept by a developer, e.g., whether or not "spawned" tasks required floating-point support. Typically the effect of such a misunderstanding was the production of incorrect code. A problem report in this project was most often resolved by remedying the direct effect of the instance of misunderstanding. That is, the code was corrected or the tester's misunderstanding was documented, and the problem report was closed by the project.

Such resolution can leave the original source of the reported defect (the phrase(s) involved in the misunderstanding) intact in the requirements. Related phrases and other areas of difficulty that might be reasonably predicted given this evidence are thus not fixed, so remain as potential, future vulnerabilities.

If the tester rather than the developer misunderstands the requirement, a false-positive problem report is generated. In this case the code is correct but the tester misunderstood the required behavior [9]. An example from the application is that a tester erroneously reported a software defect when the code correctly threw an error condition. In a false-positive case such as this, there is the possibility that the misunderstanding can recur in operations. Even if the code is correct, an operator may experience the same misunderstanding that the tester reported. In a deployed system such communication deficiencies can add risk. For example, an operator responding to a misinterpreted signal from the system might execute an inappropriate and potentially harmful action.

These concerns lead us to establish specific goals in identifying potential sources of misunderstanding that might be reasonably predictable given evidence from testing materials. We introduce these goals below.

4.2 Target Phenomena of Investigation

Using analysis of the testing defect reports and change requests, we seek to provide an efficient method of accomplishing the following two main goal sets:

First, identify, and locate in the requirements, the following classes of phrases that have the potential to cause misunderstanding:

- Source instances of phrases directly implicated by testing materials, i.e., the location of an instance to which a problematic phrase is traced. For example, a report documenting the mistaken use of "relative counters" for "absolute counters" referred to the location where the word "counter" occurred that provoked the misunderstanding;

- Other instances of the same phrase. For example, if "housekeeping" is identified as problematic in testing materials, then all instances of this phrase in the requirements should be examined for potential misunderstanding; and

- Instances of related or qualified versions of the phrases that appear to possess similar potential for confusion. For example, if "commandable rate" is identified as problematic, then other "commandable" parameters, such as size, time and mode (all of which occurred in the investigation materials), are candidates for further inquiry. Once identified, these instances

can all be individually examined and, if necessary, corrected in requirements maintenance.

Second, and more significantly, we seek to provide a prioritization mechanism for allocating resources to both remedy the specifications and predict areas of difficulty from the information in the testing materials. Thus, in addition to the previous goals, we also seek to:

- Provide a foundation for judgments regarding which phrases in the first goal set offer the greatest return-on-investment if repaired, and

- Provide a foundation for the visualization of higher-order patterns in the requirements that may indicate areas of conceptual difficulty and miscommunication potential.

4.3 Process of Investigation

In order to identify instances of problematic phrases, as well as additional areas of difficulty, we applied a process consisting of the following steps:

1. Identify problematic phrases from the testing reports;

2. Locate all occurrences of those phrases in the requirements (both the instance directly causing the miscommunication as indicated by the testing report and other instances of the same phrase);

3. Use the context provided by the previous step to identify related and qualified versions of the phrases from the initial set and add them to the updated set;

4. Compute metrics to determine the occurrence properties of concern, that is, frequencies and distributions, for all phrases in the updated set; and

5. Analyze the occurrence properties. We distinguish below between Level 1 implications, which are those deriving from the occurrence properties of individual phrases, and Level 2 implications, which those deriving from correlations in occurrence properties of multiple phrases examined together.

We describe each of these steps and their application below.

4.3.1 Identify the Phrases of Interest

In order to construct the initial set of phrases of interest, we examined the testing defect reports to select phrases considered to be problematic. A problematic phrase was one that indicated deficient communication that was either central to or incident to the reported problem. Our initial analysis used manual inspection to identify which of the 145 defect reports appeared to be requirements-related (as opposed to indicating coding errors, etc.). We used the textual description of the problems in the associated change requests to help judge whether the reported defect was due to problematic phrases in the requirements documents. Section 5 lists several identified phrases.

The level of effort needed to identify the key phrases was dominated by the time it took to read through all the text-based defect reports and associated change requests; the additional effort needed to actually identify the key phrases during this reading was minimal, since the criteria for selection had already been defined.

Seventeen of the problem descriptions appeared to be requirements-related. We searched these problem descriptions for the phrases that appeared to be problematic for requirements understanding. Since these deficiencies were implicated in defect reports, they have potential impact on the validity of the system.

Of the seventeen problem descriptions selected, we judged five of the problem reports to be of high-criticality, i.e., those defects with the potential to cause system failure. An example of this category is that, due to a misunderstood requirement regarding timeouts, no events were flowing into the system in one scenario. Eight defects were of medium criticality, i.e., with potential to cause erroneous output data. An example of this category is that, due to an incomplete definition of multiplexer values, return values from one component were always invalid. Lastly, two of the defects were judged to be of low criticality, i.e., defects that do not have potential to cause failure or erroneous output.

We considered phrases to be problematic if:

- the defect report explicitly described misinterpreted concepts. The description given earlier of the false assumption that counters were relative rather than absolute, for example, was given in the problem report itself.

- the defect report described the use of a particular phrase that contributed to the problem reported. For example, in one report, misunderstanding of the phrase “repeated errors” clearly contributed to the problem reported, but this was not explicitly noted by the tester.

This second criterion was further elaborated using the

classes of limitation on semantic accessibility introduced in Section 3. That is, any phrase selected without direct implication from a tester's comments was based on evidence of either too many available meanings, no available meaning, or an available meaning that was either circular or non-predictive and therefore not pragmatically useful.

The phrases implicated according to the above evidence to be at risk for misinterpretation constituted the initial set of phrases of interest.

4.3.2 Locate all Occurrences of the Phrases

The second step of the process was to locate all occurrences of each problematic phrase in the requirements documentation. This addressed the first two categories in the classification in Section 4.2. This step provided guidance for correction of all instances of a phrase known to be problematic rather than just the single instance with which a defect report was associated. This step also allowed examination of context to help refine the phrase list before computing occurrence metrics. The refinements are described in the next section.

4.3.3 Identify Variations and Refine the Phrase List

The third step identified related and qualified versions of the phrases in order that the initial set could be refined to better represent the problematic concepts. This refinement addressed both grammatical and conceptual variations, as elaborated below.

Grammatical variations are variations of a phrase that differ in, for example, tense, number or word order. Phrases that differed in only these ways, such as "spawn and "spawned", "constant" and "constants", or "reporting period" and "period of reporting", were merged into one unique entry.

Conceptual variations are variations of phrase that either subsume or are subsumed by known problematic phrases, or that represent semantically coherent closures of these relations. Thus, as noted earlier, the presence of "commandable rate" in the initial phrase list led to consideration of, e.g., "commandable mode" and "commandable size". "Commandable" by itself was ultimately left in the list to represent all of these variations. Similarly, both "buffer" and "ring buffer" were initially determined to be problematic, but inspection of their locations in the requirements revealed multiple additional qualified versions of "buffer". It was decided to treat "buffer" as the phrase of interest so that the metrics for all qualified versions would be taken into account. As the common link among them, it was the target of interest for measurement, since measuring only the variations individually would mask the wider impact of "buffer".

As a result of such grammatical and conceptual mergers and reassignments, this step produced a refined list of unique problematic phrases. This list was then analyzed in the fourth step of the process.

4.3.4 Determine Occurrence Properties of the Phrases

This step computed the occurrence properties throughout the requirements documentation of the problematic phrases in the refined list. For example, the problematic phrase "errno" occurred six times. Using prototype tool support to partially automate the process, we first measured the total frequency of occurrence in the requirements documents of each phrase. This measurement included all instances of the phrases or of its variations.

Second, we measured the distribution of occurrence across the requirements documents for each phrase. This measurement counted the number of separate modules in which each phrase or its variations occurred, relative to the total number of modules. For example, the occurrences of the phrase "errno" were spread over six different modules.

4.3.5 Analyze Occurrence Properties of the Phrases

In this step, the occurrence properties measured previously were analyzed in order to detect patterns allowing level 1 and level 2 implications to be drawn. We describe these patterns and their implications below.

Level 1 Implications. A level 1 implication is an implication that arises from the occurrence properties of a single phrase. The level 1 implications with which we are concerned are those derivable from the frequencies and distributions of individual phrases. In particular, high relative frequency indicates a high number of opportunities for misinterpretation e.g., of exactly to what the value of "errno" refers. This implies that prioritization in correction and maintenance should be assigned in order of highest to lowest frequency phrases.

Similarly, high relative distribution indicates wide influence on multiple aspects of a system and thus the potential for misunderstandings to result in non-localized or uncontained faults that affect the system in pervasive ways. This implies that prioritization in correction and maintenance should be assigned in order of highest to lowest distribution phrases.

Level 2 Implications. A level 2 implication is an implication that arises from the coincidence of occurrence properties of more than one phrase analyzed together.

The level 2 implications with which we are concerned are those derivable from the coincidences of frequencies or distributions among phrases. In particular, if phrases with relatively coincident frequencies can be judged to be semantically related concepts, they may indicate cross-cutting conceptual dimensions of the system that are

difficult to communicate clearly. For example, two terms of high frequency that both concern aspects of the measurement units in use in the system might suggest preferential direction of resources toward anything having to do with measurement units.

Similarly, phrases with relatively or partially coincident distributions can indicate system components or other functional divisions (as far as the coincidence maps to functional divisions of the requirements) that are cognitively more difficult to manage. That is, if several terms are distributed such that their occurrences coincide in a particular section of the requirements, this might indicate preferential direction of resources to the refinement of the requirements for the component or functional division treated by that section.

5. Results

Using the selection and refinement methods described in steps one through three above, we produced a set of 27 phrases identified to be problematic given evidence from testing materials. The 27 phrases were: absolute count, active value, buffer, commandable rate, compression control parameter, configuration parameter, constant, discard, driver, dynamically switch, errno, floating point support, frame depth, frame limit, housekeeping, idle, M frame, MB, monitor task execution, multiplexer value, relative count, repeated error, reporting period, reset, semaphore, spawn, and timeout. We examined the occurrence properties for these 27 and found the following patterns. We first present and discuss the measured property data, and then present additional findings that fell outside of the original classification.

5.1 Measured Occurrence Properties

In this section we first present the data produced in steps one through five of the method defined in Section 4. We then discuss the implications of these data for correction and maintenance of requirements documentation.

5.1.1 Data

As stated above, a total of 27 phrases were identified to be problematic given evidence from testing materials. Of these 27, the following occurrence properties were recorded:

- 15 of the phrases occur in the requirements text; 12 do not.
- For the 15 phrases that occur in the requirements text, the frequencies are represented in Figure 1. At the extremes, 5 unique phrases of the 15 occurred once each throughout the documents, while one unique phrase occurred 18 times.
- For the 15 phrases in the requirements text, their distributions across the 19 modules of the SRS are represented in Figure 2. At the extremes, 5 unique phrases of the 15 occur in only one of the 19 modules each, while one unique phrase occurred in 6 different modules.

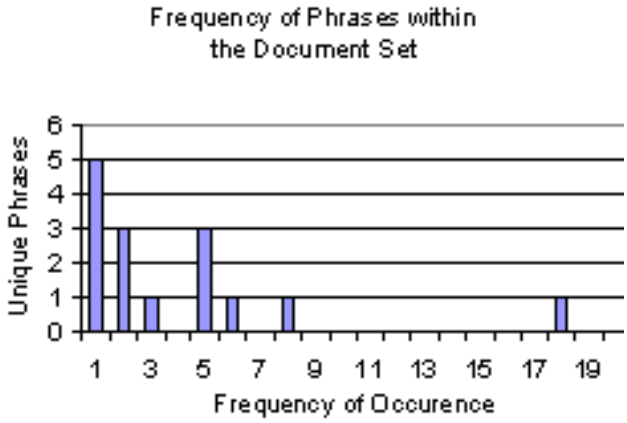

Figure 1. Phrase Frequencies

- Finally, for the 15 phrases that occur in the requirements document, their distributions coincided as represented in Figure 3. At the extremes, 6 of the 19 modules of the SRS contain no problematic phrase, while one module, module 13, contains 7 of the unique phrases determined to be problematic.

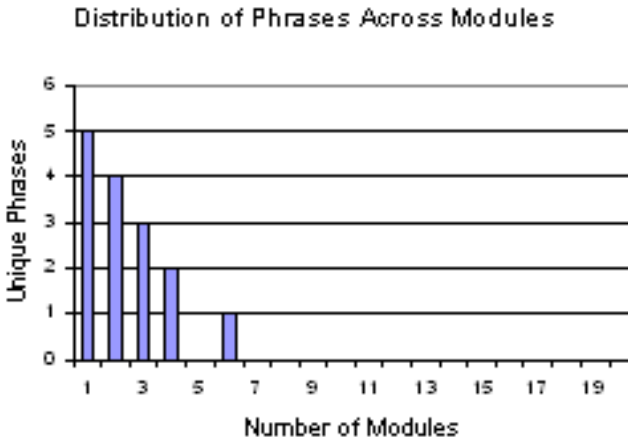

Figure 2. Phrase Distributions

5.1.2 Discussion

Disparate Vocabularies. The first discovery was that a significant minority (12 of 27) of the phrases identified as problematic and requirements-related did not occur in the text of the requirements. One possibility was that some concepts important to the validity of the system were missing from the requirements, e.g., "configuration parameters", "multiplexer value", that is, that the requirements were measurably incomplete. In fact, we found a vocabulary mismatch between that used by the developers of the requirements and that used by testers of the system. Such a mismatch can make it harder for testers and maintainers to move efficiently and with fidelity between the two vocabularies in order to perform correction and maintenance. One recommendation that emerged from our study was for improved procedural support for tighter coupling among the vocabularies of different stakeholders.

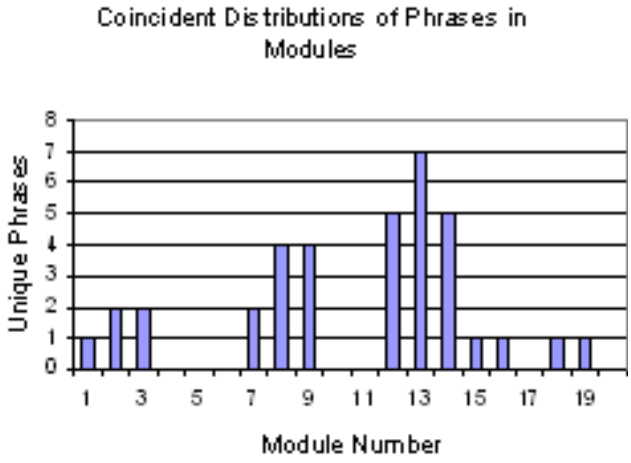

Figure 3. Coincident Distributions

Prioritization by Frequency. Ranking of the phrases by frequency showed that six of the problematic phrases occurred five or more times each. These phrases were (in decreasing frequency) *buffer* (18 occurrences), *timeout* (8), *errno* (6), *discard* (5), *driver* (5), and *reset* (5). Such phrases have relatively more opportunities to cause problems so we recommend that frequently occurring problematic phrases be addressed before the less frequently occurring phrases.

Prioritization by Distribution. Ranking of the phrases by distribution showed that six of the problematic phrases occur across three or more modules each. These phrases were (in decreasing order) *errno* (6 modules), *buffer* (4), *timeout* (4), *discard* (3), *driver* (3), and *reset* (3). These phrases have relatively more potential to affect a wider portion of the system. We recommend preferential attention to problematic phrases that occur in multiple modules.

Prioritization by Coincident Properties. The phrases that had similar frequencies did not appear to have a discernible semantic connection. We can thus make no recommendation regarding semantically complex cross-cutting dimensions of the system.

However, we did see a pattern to the coincidence of distributions: five modules each contain four or more of the problematic phrases. It may be that these five modules represent cognitively more difficult system components or functional divisions that should be addressed preferentially in review and improvement. In addition, there were some finer-grained coincidences represented. For example, one of the phrases occurred in three different modules within the same subsection, with each such subsection dealing with the same topic across the three modules. Similarly, several phrases occurred in another subsection of multiple modules. This indicates further potential issues in well-defined sub-areas of the system, allowing additional focusing of resources.

These results indicate that there is more information available in problem reporting than is generally used. In particular, occurrence properties of problematic phrases showed some patterns that can provide guidance in allocating resources for correction and maintenance. This allows portions of a SRS at higher risk for deficient communication to be preferentially addressed.

5.2 Additional Observed Patterns

In addition to the patterns detected through measurement of occurrence properties, we identified some other patterns that help further characterize communication deficiencies in requirements.

Problems with Domain Project-Specific Acronyms. Acronyms caused many problems. They were a source of obscurity, in that they frequently represented the case introduced in Section 3 in which no meaning is available to which to map. We excluded project-specific acronyms from our measurement to not skew the data, but did include the acronym "MB" (megabyte) because it was not project-specific. The problem in this case was not with determining the proper expansion of the acronym but with its semantic interpretation: one party interpreted it as one million while another interpreted it as 1024 x 1024.

Recurring Linguistic Deficiencies. Several classes of linguistic deficiency described in Section 3 occurred repeatedly, i.e., overloaded, undefined, circular, and non-predictive phrases, terminology and definitions. These are well-defined objects of study for which we are

interested in developing efficient preventive measures. Linguistic devices that represent under-specification (such as “some” and “like”) were also common, as well as typographical errors (e.g., “to” for “too” and “admistration” for “administration”).

Better management of recurring types of communication deficiencies in requirements specifications can reduce testing and operational defects. To this end, we are simultaneously pursuing investigation and recommendations on appropriate definition practices [13]. It was not possible in the application described here to interact with the project, e.g., to provide an improved or marked-up requirements specification and solicit feedback on the changes from the project. Future work is thus needed to evaluate the effectiveness of the suggestions resulting from our work here to reduce misunderstandings of requirements that lead to defects. A first step is to gather anecdotal feedback by asking testers to compare portions of natural-language specifications containing problematic phrases with alternative versions implementing the recommendations.

5.3 A Note on Process Integration

In order to help organize and execute the investigations, we explored partial automation of some of the activities. In particular, we developed prototype tool support for both indexing and locating instances of problematic phrases within requirements text, as well as for computing occurrence property metrics for the phrases selected. Partial automation is essential for this approach to be cost-effective for large software systems. To this end, we plan to integrate this method with existing partially-automated support tools for managing natural language issues in requirements [14].

6. Summary

This work described a feed-forward mechanism by which problem reports generated during testing are analyzed to discover where requirements specifications need to be improved to reduce latent, related faults and to preclude defects due to miscommunication during operations. The paper presented criteria for identifying problematic phrases and demonstrated the analysis process on a real-world system. Results showed that the analysis of the frequency and distribution of problematic phrases in the requirements specification identified further potential sources of miscommunication. Some of these instances were particular to the system being studied, while others were indicative of common difficulties in communicating domain knowledge through text.

The occurrence properties of the problematic phrases also provided guidance for prioritizing resources for corrective action, and were shown to offer reasonable predictability regarding the location of other, associated communication gaps in the requirements specifications. These findings suggest the feasibility of a proactive approach to improving requirements by anticipating and correcting latent textual vulnerabilities. The results also suggest implications for projects that combine this and other partially-automated methods for analysis of requirements to avoid recurring types of communication deficiency and improve the maintenance of requirements knowledge for a system.

7. Acknowledgements

The authors thank Mike Chapman, project manager of the Metrics Data Program, for his assistance with the dataset. The authors also thank Sean Travis for assistance in automating the generation of metrics. This work was funded by NSF under contracts CCR-0205447, 0204139, 0205588 and REU-0311877, and by NASA under contract NAG-1-02103.

8. References

[1] B. Curtis, H. Krasner, and N. Iscoe “A Field Study of the Software Design Process for Large Systems” Communications of the ACM, vol. 31, no. 11, 1988, pp. 1268-1287.

[2] N. E. Fenton and N. Ohlsson, “Quantitative Analysis of Faults and Failures in a Complex Software System,” IEEE Trans on Software Eng, vol. 26, no. 8, Aug, 2000, pp. 797-814.

[3] C. Goddard, Semantic Analysis, Oxford University Press, Oxford, 1998.

[4] K. S. Hanks, and J. C. Knight, “Improving Communication of Critical Domain Knowledge in High-Consequence Software Development: an Empirical Study.” 21st International System Safety Conference (ISSC'03), Ottawa, Canada (August, 2003)

[5] K. S. Hanks, J. C. Knight, and E. A. Strunk, “Erroneous Requirements: A Linguistic Basis for Their Occurrence and an Approach to Their Reduction,” Proc. 26th NASA Goddard Software Eng Workshop, IEEE, Greenbelt, MD, Nov., 2001.

[6] S. Lauesen and O. Vinter, “Preventing Requirements Defects: An Experiment in Process Improvement,” Requirements Engineering Journal, 2001, pp. 37-50.

[7] M. Leszak, D.E. Perry and D. Stoll, “Classification and Evaluation of Defects in a Project Retrospective,” The Journal of Systems and Software, vol. 61, issue 3, 1 April, 2002, pp. 173-187.

[8] R. Lutz and I. C. Mikulski, “Empirical Analysis of Safety-Critical Anomalies during Operations,” with C. Mikulski, IEEE Transactions on Software Engineering,” vol. 30, no., 3, March, 2004, pp. 172-180.

[9] R. Lutz and I. C. Mikulski, "Resolving Requirements Discovery in Testing and Operations," with C. Mikulski, Proc. 11th IEEE Requirements Engineering Conference (RE'03), Sept. 8-12, 2003, Monterey Bay, CA, pp. 33-41.

[10] Metrics Data Program, NASA Independent Verification and Validation Facility, http://mdp.ivv.nasa.gov.

[11] T. J. Ostrand and E. J. Weyuker, "The Distribution of Faults in a Large Industrial Software System," Proc Int'l Symp on Software Testing and Analysis, in Software Engineering Notes, July, 2002, pp. 55-64.

[12] G. Salton, Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer, Addison-Wesley, Reading, PA, 1989.

[13] K. Wasson, "Partial Reductive Paraphrase: Toward More Transparent Requirements," University of Virginia Department of Computer Science Technical Report CS-2004-16, May 2004.

[14] K. Wasson, J. Knight, E. Strunk, and S. Travis, "Tools Supporting the Communication of Critical Domain Knowledge in High-Consequence Systems Development," Proc. 22nd International Conference in Computer Safety, Reliability and Security (SafeComp2003), Sept. 23-26, 2003, Edinburgh, Scotland, UK, pp. 317-330.

[15] K. A. Weiss, N. Leveson, K. Lundqvist, N. Farid, and M. Stringfellow, "An Analysis of Causation in Aerospace Accidents," Space, 2001, Aug., 2001.