Resolving Requirements Discovery in Testing and Operations

Robyn R. Lutz
Jet Propulsion Laboratory
and Iowa State University
rlutz@cs.iastate.edu

Inés Carmen Mikulski
Jet Propulsion Laboratory
Pasadena, CA 91109-8099
ines.c.mikulski@jpl.nasa.gov

Abstract

This paper describes the results of an investigation into requirements discovery during testing and operations. Requirements discovery includes both new requirements and new knowledge regarding existing requirements. Analysis of anomaly reports shows that many of the anomalies that occur during these phases involve requirements discovery. Previous work by the authors identified four common mechanisms for requirements discovery and resolution during testing. The results reported here extend that work in two ways: (1) to show that very similar requirements-discovery mechanisms are at work in both testing and operations, and (2) to evaluate the requirements-discovery mechanisms against experience with seven additional systems. The paper discusses the consequences of these classifications and results in terms of reducing requirements-based defects in critical, embedded systems. Keywords: requirements discovery, requirements evolution, defect analysis, safety-critical systems, testing, operations.

1. Introduction

For many critical, embedded systems, software requirements continue to be discovered throughout the system's lifetime. The work described in this paper investigates requirements discovery during the later phases of projects, in particular during testing and operations. Analysis of anomaly reports during testing and operations shows that many of the anomalies during these phases involve software requirements discovery. Stated in another way, incomplete requirements and requirements misunderstandings are the source of many testing and operational anomalies.

Requirements discovery includes both new requirements and new knowledge regarding existing requirements that emerge during testing and operations. We are interested here in understanding more precisely what types of requirements discovery occur in systems after they are built and what types of resolution occur to handle these requirements-related anomalies. To achieve this we analyze software anomaly reports in order to characterize patterns of requirements discovery and resolution.

In previous work [13] we identified four common mechanisms for requirements discovery and resolution during the integration and system testing of a safety-critical system:

- Incomplete requirements, resolved by changes to the software
- Unexpected requirements interactions, resolved by changes to the operational procedures
- Requirements confusion on the part of the testing personnel, resolved by changes to the documentation, and
- Requirements confusion on the part of the testing personnel, resolved by a decision that no change was needed.

In this paper we evaluate these four mechanisms for requirements discovery and resolution on operational (i.e., deployed) systems. The results reported here extend the previous work in two ways: (1) to show that very similar requirements-discovery mechanisms are at work in both testing and operations, and (2) to evaluate the requirements-discovery mechanisms against experience with seven additional systems.

*The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. It was funded by NASA's Office of Safety and Mission Assurance, Center Initiative UPN 323-08. The first author's research is supported in part by National Science Foundation Grants CCR-0204139 and CCR-0205588.

The paper also discusses the consequences of these classifications and results in terms of reducing requirements-based defects in critical, embedded systems. Specifically, the experience reported here showed:

- that both testing and operational anomalies reveal a dependence on procedures to satisfy some newly identified requirement interactions, such as required sequencings of activities,
- that anomaly reports for one system can identify requirements-related risk for similar, future systems,
- that "false-positive" anomaly reports, where the software behavior is actually correct but is unexpected, provide valuable insights into requirements confusions,
- and that analysis of anomaly reports allows us to pinpoint some recurring patterns of requirements confusion.

The rest of the paper is organized as follows. Section 2 describes the approach used to analyze the anomalies. Section 3 presents and evaluates the results. Section 4 discusses the implications of these results and derives some lessons learned. Section 5 puts the results in the context of recent, related work. Section 6 provides concluding remarks and identifies some open questions.

2. Approach

The approach to the work reported here was to analyze the anomaly reports generated during testing and operations to gain insight into requirements problems. These data are contained in an institutional, multi-project, on-line database. Although the on-line forms differ somewhat for testing and operations, the forms are similar. Both contain the following three parts: a description of the problem, a subsequent analysis of the problem, and a description of the corrective action taken to close out the anomaly report.

The testing dataset consisted of 326 filled-in software anomaly reports written during integration and system testing for the twin Mars Exploration Rover (MER) spacecraft currently under development for launch in summer, 2003. This more than doubles the number of testing anomaly reports analyzed in the initial set [13] and includes data up to mid-December, 2002. MER will explore Mars with two robotic rovers equipped to search for, among other things, evidence of past water.

The operational dataset consisted of the 189 anomaly reports in the highest-criticality level generated post-deployment by the operations teams on seven other spacecraft. Operational data were analyzed for the following recent or current spacecraft: the Galileo mission to Jupiter, launched in 1989; Mars Global Surveyor, launched in 1996; Cassini/Huygens, launched in 1997 to Saturn and Titan; Deep Space 1, an ion-propulsion and remote-agent technology mission, launched in 1998; Mars Climate Orbiter, launched in1998; Mars Polar Lander, launched in 1999; and Stardust, a comet sample-return mission, also launched in 1999.

The anomaly reports document not only defects but any behavior that is unexpected by the testing or operational personnel. The anomaly reports are thus a rich source of discovery of both latent requirements (where the software does not behave correctly in some situation) and of requirements confusion (where the software behaves correctly but unexpectedly).

The method of analysis for the anomaly reports is an adaptation of Orthogonal Defect Classification (ODC)[2]. ODC provides a way to "extract signatures from defects" and to correlate the defects to attributes of the development process.

The ODC-based approach uses four attributes to characterize each anomaly report. The first is the Activity, which describes when the anomaly occurred (e.g., System Test or Flight Operations). The Trigger describes the environment or condition that had to exist for the anomaly to surface (e.g., a Fault Recovery condition or Hardware/Software interaction). The Target characterizes the high-level entity that was fixed in response to the anomaly's occurrence (e.g., Flight Software or Information Development). Finally, the Type describes at a lower-level the actual fix that was made (e.g., Documentation, Procedure, or Function/Algorithm). Table 1 lists the ODC signatures extracted for the requirements-discovery investigation.

Each anomaly was classified twice, once by each of the two authors. If there were discrepancies between these two classifications, they were reconciled in joint discussions. Both authors have experience on flight projects at JPL, but neither is involved with the testing or operations of the systems under study. Spacecraft personnel, especially on MER, generously assisted us by answering domain and process questions. A fuller description of the classification process appears in [12].

While all available test anomalies were analyzed, only the *critical* operational anomaly reports were analyzed. By "critical" we mean that the anomaly was ranked by the project as highly critical. Depending on the fields used by the project in their reporting, such anomalies were thus marked as "red-flag," "potential-red-flag," "high-criticality," or "significant or catastrophic failure effect risk." Criticality ratings currently existed for only

a small subset of the testing reports, so criticality ratings were not useful for analysis of the testing anomalies.

The fact that the same requirements-discovery mechanisms evident in testing caused *critical* software anomalies post-launch motivate continuing investigation in this area. The goal of the multi-year study in which this work is embedded is to reduce the number of safety-critical software anomalies that occur after launch.

Table 1. ODC signatures investigated

Category	ODC Target	ODC Type
Incomplete requirements and software fix	"Flight Software"	"Function Algorithm" (primarily)
Unexepcted requirement interactions and procedural fix	"Information Development"	"Missing Procedures" or "Procedures Not Followed"
Requirements confusion and documentation fix	"Information Development"	"Documen-tation" or "Procedure"
Requirements confusion and no fix	"None/Unknown"	"Nothing Fixed"

3. Results and analysis

The results of the ODC analysis of the testing and operational datasets showed that the mechanisms for requirements discovery are very similar across both the testing and the operations phases. Figure 1 summarizes the results. It shows that 65 of the 326 testing anomalies and 25 of the 189 critical operational anomalies involve the requirements mechanisms identified here. The following subsections describe each of those four mechanisms with a discussion of the ODC analyses for both the testing and the operational datasets. Because some of the testing work has been previously reported in [13], we describe it only in enough detail to understand the comparison with operations. Because the requirements discovery work in operations is new, we describe it more fully.

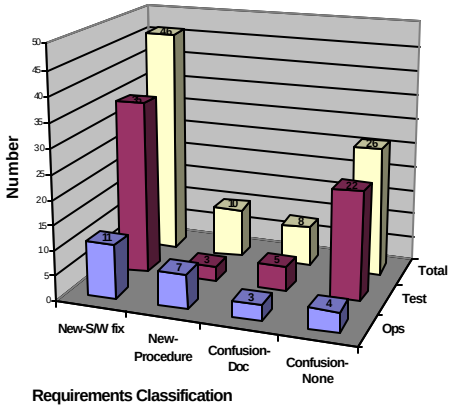

Figure 2. Classification of anomaly reports

3.1 Incomplete requirements, resolved by changes to software

The first mechanism for requirements discovery is incomplete or missing requirements resolved by changes to the software. In these cases an anomaly report is written during testing or operations. This anomaly involves or results in the discovery of new requirements knowledge. The corrective action taken to fix the anomaly is to implement the new requirement in the onboard flight software.

Excluded from this class of requirements discovery are requirements involved in the planned evolution or scheduled maintenance of the system. Although these activities routinely involve new requirements, these new requirements are not anomaly-driven and do not reflect requirements discovery in the same way. Software is regularly uploaded to the spacecraft before a new mission phase to control the activities associated with that new phase. For example, as the mission passes from cruise to planetary encounter, new software requirements will be implemented in the flight software. However, these planned updates do not routinely reflect the discovery of new requirements. This contrasts with the unplanned changes to requirements prompted by critical anomalies during operations that are studied here.

In testing, there were 160 anomaly reports with an ODC Target of "Flight Software." The ODC Target describes what was fixed. Thirty-five of these involved incomplete or missing requirements resolved by changes to the software. These missing requirements were either unidentified or new requirements. Many of these describe timing or initialization issues arising from the interaction among software components or between software and hardware. For exa mple, in one such anomaly a new requirement became evident during testing for the initial state for a component to wait for completion of the initial move of a motor. In another case an off-nominal scenario was identified in

testing in which certain interfaces had to be temporarily disabled to achieve correct behavior. In a third anomaly, analysis of the problem revealed a requirement to re-enable a reset driver during a reboot.

In operations, there were forty-four anomaly reports with ODC Target = "Flight Software." Eleven of these involved missing or incomplete requirements. In four cases a new software requirement compensated for the failure or degradation of a hardware component. For example, when a damaged solar array could not deploy correctly, a new flight software requirement added needed functionality in response. In another anomaly, noisy transducers caused excessive resetting of hardware components, a risk to the system. In response, a new flight software requirement compensated for the noise.

In seven other cases the anomaly was handled by a new software requirement to handle an unusual event or scenario. In these cases the requirements discovery involved unforeseen system behavior that was resolved by requiring additional fault tolerance for similar, future incidents. In one anomaly, for example, an unexpected outflow of debris temporarily blinded the spacecraft, making it difficult to determine its position in space. As a consequence of the anomaly, new software requirements were implemented to make the spacecraft robust against similar events in the future.

In both testing and operations, requirements discovery was often resolved by changes to software in the systems studied. In testing, new requirements emerged most often from subtle dependencies among software components or between the software and the hardware. In operations, rare scenarios or hardware degradations caused critical anomalies resolved by urgent, unplanned software requirements changes.

3.2. Unexpected requirements interactions, resolved by changes to operational procedures

The second mechanism for requirements discovery also involves new requirements, but in these cases the new requirement is implemented via an operational procedure. Such anomalies usually involve unexpected requirements interactions detected during testing or operations. Analysis of the anomaly sometimes results in a new requirement that certain activities be performed in a specific order (e.g., to prevent a race condition) or in a specific timing relationship. The change made to resolve the anomaly in the cases described here is not to the software but to the procedures.

In testing there were eighteen anomaly reports with ODC Target = "Information Development" and ODC Type = "Missing or Incomplete Procedure." Of these, there were three anomaly reports that displayed this mechanism. For example, in one case, a software fault monitor issued redundant off commands from a particular state reached during testing. Although this software behavior was correct, it was undesirable. A decision was made to prevent these redundant commands by procedurally selecting limits that would avoid that state in the future.

In post-launch operations there were seven critical anomalies with ODC Target = "Information Development" and ODC Type = "Procedure." In each case the new requirements knowledge was implemented by a procedure.

For example, one anomaly identified a need to be able to recover the commands remaining to be executed after an abnormal termination occurred. This requirement was resolved by creation of a new operational procedure to respond to similar situations in the future. In another case a problem occurred when two streams of data were sent simultaneously. The anomaly revealed a latent requirement that had not been previously recognized to ensure that only one stream of data at a time be transferred. Again, this was handled procedurally. In a third anomaly the software behavior was incorrect in that an aero-braking maneuver (to lower the periapse) was erroneously performed twice rather than once. This occurred when the software was loaded to memory too soon, i.e., to an area of memory that was currently active. The fix was to add a procedure to enforce a new requirement preventing the recurrence of the configuration problem.

Handling a new requirement via a procedural change avoids the cost and risk of updating complex, safety-critical flight software. However, it introduces the risk that operational personnel may not implement the procedure every time a similar situation occurs. Resolution of critical anomalies such as these via changes to procedures places high dependence on the requirements knowledge and motivation of the operational personnel.

The anticipated length of a system's lifetime may be a factor in the decision as to whether to handle an emergent requirement procedurally or not. For a relatively short-lived system (e.g., a mission to Mars, measured in months) a change to operational procedures may make sense. For a relatively long-lived system (e.g., a seven-year trip to Saturn followed by a multi-year scientific mission), there will inevitably be personnel turnover and ensuing loss of requirements knowledge. For a long-lived system, procedural implementations of requirements may add risk.

3.3 Requirements confusion, resolved by changes to documentation

In both of the first two mechanisms for requirements discovery, the anomaly report reveals the existence of a new requirement or of a newly understood dependency among requirements. In the first mechanism the anomaly results in a change to software. In the second mechanism the anomaly results in a change to operational procedures to enforce the correct behavior of the system. The third and fourth mechanisms differ from the first two in that the third and fourth involve the discovery of a requirements confusion rather than the discovery of a missing or incomplete requirement. In these cases, the software works correctly, but the testing or operational personnel are surprised by this behavior.

In the third mechanism the anomaly is resolved by fixing the documentation (e.g., the list of flight rules or the design document) to explain the required behavior and the requirements rationale. Analysis of problem reports from testing showed sixteen anomalies with an ODC Target = "Information Development" and an ODC Type = "Documentation." Five of these involved incorrect assumptions about the requirements on the part of the testers. These misunderstandings were made manifest during testing when correct software behavior did not match the testers' expectation. Such testing reports were handled by correcting the source of the misunderstanding via improved documentation.

For example, one anomaly was caused by an incorrect assumption by testing personnel that some heaters would remain off as the software transitioned between two specific modes. The anomaly was resolved by correcting the design documentation to describe the software requirement implemented by another component to turn the heaters off when this transition occurred.

Requirements confusion also caused critical operational anomalies in the systems studied. Analysis identified three such anomalies with an ODC signature of Target = "Information Development" and Type = "Documentation." Of these, one anomaly involved requirements confusion. In that case, the anomaly reported a drop in battery power resulting from a requirements misunderstanding of the behavior initiated by the command that was used. The corrective action taken was to document the required behavior and associated command in an operational flight rule.

Unlike in testing, anomalies caused by requirements confusion also occasionally resulted in the improved documentation of a procedure. Two of the forty-six critical operational anomalies with Target = "Information Development" and Type = "Procedure" involved the correction of a requirements misunderstanding through the documentation of procedures. In these cases there was no new requirement (unlike in mechanism 2) but instead improved communication of a known constraint. For example, in one anomaly a required precondition for a calibration (that the instrument be in an inertial mode) was not understood. The problem was avoided in future calibrations by documenting this requirement in the systems checklist.

3.4 Requirements confusion, resolved without change

As in the previous mechanism, the cause of these anomalies is requirements misunderstanding. However, in mechanism 4 no fix is made. There were sixty-seven testing anomalies with ODC Target = "None/Unknown" and ODC Type = "Nothing Fixed." Such anomaly reports are false positives, reporting a problem when, in fact, the software behaved correctly and in accordance with the specified requirements. In these anomalies the projects subsequently determined that no change was needed. For example, in some cases no change was made because the situation could not recur in the rest of the mission.

In most cases, resolution of the anomaly report without a fix was appropriate. However, analysis of testing problem reports shows that in some anomalies with this ODC signature, the same requirements confusion might be able to recur in operations. In such cases, the requirements misunderstanding on the part of the testing personnel might be repeated by operational personnel, yielding a perhaps serious operational anomaly. .For example, in one scenario the software declared a cold boot even though a warm boot had occurred. This behavior is required, but the rationale is somewhat complicated and merits further documentation to preclude similar confusion.

In another testing incident the tester expected that commands issued to a component when it was off would be rejected. Instead, the commands unexpectedly executed when the component was rebooted. This behavior was, in fact, required, but (reasonably) was not the behavior expected by the tester. Since this misunderstanding might be able to recur in operations with serious effect, it is worth the effort to record the software's actual behavior to help reduce gaps between expected and actual performance.

Similarly, there were four cases of requirements confusion in operations that warranted clarification in the documentation to avoid future anomalies due to the same requirements confusion. In one anomaly report a

defect was caused by the counterintuitive use of the words "deploy" and "stow." In this case "deploy" was synonymous with "close an instrument cover" and "stow" was synonymous with "open or remove the instrument cover." The potential for confusion regarding the required software behavior when the instrument cover had to be moved was thus quite high. In another case a rare situation was found in which a parity error could be missed in a certain combination of circumstances. A decision was reasonably made not to fix it on this system due to resource constraints. However, since the problematic configuration is also possible with other systems, further documentation for future, similar systems is recommended.

Interestingly, in all four of these anomalies the lack of corrective action to remedy the requirements misunderstanding was judged by the projects to suffice for the system on which it occurred. However, in all four cases it was noted in the anomaly report that the misunderstanding could also occur on other, future spacecraft. That is, the requirements misunderstanding in each case was perceived as a recurrent risk on other systems. This focus on the next-generation systems by operational personnel suggests a need in defect analysis to broaden the perspective from consideration of a single system to consideration of a set, or family, of similar systems (in this case, interplanetary spacecraft). These results suggest the possibility that better reuse of knowledge regarding past requirements confusions may forestall similar requirements confusions on other systems in the same product family

4. Implications for testing and operations

The experience reported here indicates that similar mechanisms for requirements discovery occur in testing and operations.

4.1 Minor differences between phases

The two ways in which requirements discovery and resolution differ in testing and operations were minor. First, in testing the anomaly reports studied sometimes resulted in procedural changes, but they did not result in fuller documentation of existing procedures. However, in operations anomalies involving requirements confusions sometimes led to improved documentation of procedures (without any procedural changes). Since the testing defects were drawn from one project and the operational defects from seven, further data is needed to draw any conclusion.

The second way in which requirements discovery differed in the two phases was that in testing the focus of the anomaly reports was on the system under test. In operations the focus of the anomaly reports broadened to consider implications for similar, future systems.

4.2 Identifying risks for similar systems

As described above, when a requirements confusion was perceived as possibly recurring on other, similar spacecraft, the operational analyst often logged this concern for the future. A lesson learned that emerged from this study was that valuable pointers to possible risks on similar systems by knowledgeable, operational personnel should be stored so as to be readily retrievable in the future. How to best capture and maintain these indicators of future risk for the product family is an open problem.

4.3 Procedures satisfy constraints on requirement interactions

Both testing and operational anomalies revealed a dependence on procedures to achieve some newly identified requirement interactions. Especially for critical anomalies, allocation of requirements to procedures carries the risk that the procedure will not be carried out correctly on each occasion when the situation requires it. This concern reflects the number of times that the ODC Type for anomalies was "procedures not followed." Procedures also tend to be written largely in natural language, making rigorous analysis less likely.

Procedures often enforce requirements for sequentiality of actions or preconditions for correct software behavior. This suggests that explicit traceability from requirements to procedures be maintained. Similarly, a domain (or product-family)-specific checklist of requirements commonly handled by procedures in this domain (e.g., calibrate before use) might reduce the number of anomalies due to missing or incomplete procedures.

4.4 Patterns of requirements confusion

Certain types of requirements misunderstandings recur in the anomaly reports, suggesting that they may be frequent sources of confusion. How best to identify and target this subset of requirements (e.g., for improved specification or extra validation) merits study.

Some examples of recurring requirements confusion are evident from analysis of the anomalies. For example, in situations where both high-water marks (which record the highest value reached since the last reset) and counters (that may be reset periodically or

aperiodically) are relevant, the exact meaning of their values is sometimes mistaken. Persistence counts for high-water marks (how long has this value been the highest?), for example, caused multiple confusions.

Requirements misunderstandings between relative time measurements (deltas) and absolute time measurements (e.g., Universal Time Code) and resets can also occur. Another example of recurring requirements misunderstanding involves the distinction between software timing delays required by interface delays or transients and by "inherent," performance-related delays.

With respect to states, distinguishing component unavailability from component unresponsiveness also caused requirements-based misunderstandings. This confusion is of concern because it is often involved in health checks to detect and diagnose faulty states. The relationship between instrument states and software modes (e.g., is the heater always on in this mode?), as well as the precise relationship between redundant data (e.g., commands or warning messages) and redundant actions were sometimes quite subtle.

Work is needed in this area both to identify additional patterns of requirements confusion and to develop practical means of clarifying these patterns for the testing and operational personnel. Understanding of the requirements by the developers promotes correct software, but is only half of the story. Understanding of the requirements by the testing and operational teams is also needed for the correct use and functioning of the software.

5. Related work

The work on requirements discovery described here draws from or has implications for work in three areas: requirements evolution, requirements-related defect analysis, and goal-obstacle analysis.

Requirements discovery consists of both new requirements and of new knowledge about existing requirements. With respect to new requirements, there is a significant body of work in the area of requirements evolution. Many large or long-lived systems have requirements that evolve even after deployment to adapt to changes in user needs, hardware or software platforms, environmental factors, or policies (e.g., certification or legislation) [1,8]. In earlier work we showed that requirements also evolve during operations to compensate for hardware degradation or to add robustness when rare scenarios occur [12].

Most studies of requirements evolution consider the problems involved in what Dubois and Pohl have recently called "continuous requirements management" [4]. The focus is on establishing processes to specify and scope proposed changes to requirements. However, most of that work deals with new requirements rather than, as here, with identifying and recovering from incomplete or misunderstood requirements.

Requirements discovery during operations also involves new knowledge about existing requirements, and their dependencies, interactions with the system, or constraints. Defect analysis results in in this area have tended to concentrate on requirements in systems under development rather than on operational systems. For example, defect analysis during testing has been used to evaluate the readiness of the software for release or to estimate the reliability of the software [6]. Fenton and Ohlsson have described the problems in using defect analysis results to measure the quality of deployed software [5]. Dalal, Hamada, Matthews, and Patton have used ODC in operational systems but with the purpose of guiding pre-release process improvement [3]. More recently, Ostrand and Weyuker have compared pre and post-release faults in an investigation of module fault density and fault-proneness [14]. Unlike the systems studied here, they found very few (ten) high-severity post-release faults in thirteen releases of an inventory-tracking system and did not study fault causes.

Defect analysis has shown that misunderstanding of requirements and their underlying rationales frequently cause defects. Lauesen and Vinter, for example, looked at 200 of the 800 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 [9]. Similarly, in an earlier study of testing defects in the spacecraft domain, one of the authors found that the most common causes of critical software defects were misunderstanding the software's interfaces with the system and discrepancies between documented requirements and actual requirements [1].

As mentioned earlier, the work described here differs from this previous work in that, by looking at anomaly reports rather than just defect reports, we can more accurately gauge the role of requirements confusions. As we have seen, false-positive reports of problems (where the software behaves correctly but unexpectedly) are common and reveal latent misunderstandings.

This is important because requirements misunderstandings have been frequently cited as the source of many accidents [16]. Hanks, Knight, and Strunk have specifically implicated breakdowns in the communication of domain knowledge as a major cause of requirements defects in high-assurance systems [7].

In summarizing the results of a large defect-analysis study, Leszak, Perry and Stoll state that "domain and system knowledge continue to be one of the largest underlying problems in software development [10]."

The third area of related work is goal-obstacle analysis. Anomaly reports are written to document perceived obstacles to the achievement of required behavior. The list of ODC targets (what gets fixed) describes common ways to handle the obstacles. The results reported here provide some experience-based confirmation of the importance of some classes and subclasses of obstacles described by van Lamsweerde and Letier [15].

Obstacle analysis supports the notion that if a requirements misunderstanding can be an obstacle to the satisfaction of some future goal, then it merits resolution (e.g., documentation) to prevent that. Van Lamsweerde and Letier identified "Wrong Belief" as a possible obstacle to meeting system goals. "Wrong Belief" occurs when "the necessary information about the object state as recorded in the agent's memory is different from the actual state of the object." The Wrong Belief obstacle class is further refined into subclasses, such as Information Outdated, Information Confusion, Information Forgotten, Wrong Inference, Wrong Information Provided, and Information Corrupted.

We found occurrences in the post-launch critical anomalies of the first four of the six Wrong Belief subclasses listed above. We also identified a possible new subclass by analysis of the anomaly reports. This new subclass would be "Information Not Used," which we define as "information available to agent not used." As an example, in one case a flight rule constraint was available but was not used. This differs from the existing "Information Forgotten" subclass, which is defined as "information no longer available."

The four mechanisms for requirements discovery and resolution during testing and operations appear to be largely consistent with the framework for obstacle analysis during the requirements phase described in [15]. Requirements discovery resolved by implementing a new requirement in software (Mechanism 1) either adds a new goal or changes a goal in order to mitigate an obstacle. Requirements discovery resolved by changing the operational procedures (as when a newly understood required sequencing of interleaved activities is provided operationally) (Mechanism 2) often shifts the responsibility for a goal from the software to another agent (i.e., the procedure). In long-lived systems such shifts of goal implementation are common [12]. In particular, we have found that as hardware degrades the software is often allocated compensatory new requirements.

Requirements misunderstanding describes situations in which the requirements are correct but the required behavior is unexpected. Reducing occurrences of the obstacle in such cases may entail changes to the documentation to prevent the requirements misunderstanding on this or future systems (Mechanism 3). Finally, van Lamsweerde and Letier describe the option to just tolerate the obstacle. In this case a decision (perhaps unwise) has been made that no change is necessary (Mechanism 4). The experience reported here confirms the value of continued requirements-engineering activities as long as requirements discovery continues, i.e., into testing and operations, for complex, critical systems such as these.

6. Conclusion

The results reported here show that requirements discovery continued to cause anomalies during both testing and operations in the systems studied. Furthermore, very similar mechanisms for requirements discovery and resolution were at work in both testing and operations.

By understanding how requirements confusions contribute to anomalies, we hope to reduce their incidence during operations. Resolving and documenting requirements confusions in testing may prevent recurrence of some of those confusions during operations.

The findings suggest some patterns in what confuses people, at least within the spacecraft domain. It is an open question whether the same confusions that occur in testing will, if left uncorrected, recur in operations. Understandably, no project has volunteered to test this hypothesis. However, we have seen that some anomalies do recur. One testing anomaly report even referred to the "rediscovery" of the requirements knowledge it documented.

Given the attention that people writing the anomaly reports accord to future systems in their comments, it also appears that documenting requirements confusions in operations may have value in preventing some recurrences of those confusions on similar, future systems. That is, in the testing phase the goal is to document requirements confusions sufficiently to prevent recurrence of the confusion in the operational phase of that system. In the operational phase, the goal is to document requirements confusions sufficiently to prevent recurrence on this or similar future systems.

Acknowledgments. The authors thank Daniel Erickson and the Mars Exploration Rover engineers and test teams for their assistance and feedback.

References

[1] K. H. Bennett and V. T. Tajlich, "Software Maintenance and Evolution: a Roadmap," in A. Finkelstein, ed. *The Future of Software Engineering*. ACM Press, New York, 2000.

[2] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal Defect Classification—A Concept for In-Process Measurements, *IEEE Trans on SW Eng*, Nov. 1992, pp. 943-956.

[3] S. Dalal, M. Hamada, P. Matthews, and G. Patton, "Using Defect Patterns to Uncover Opportunities for Improvement," *Proc. Int'l Conf Applications of Software Measurement*, 1999.

[4] E. Dubois and K. Pohl, "RE 02: A Major Step toward a Mature Requirements Engineering Community," *IEEE Software*, vol. 20, no. 1, Jan/Feb, 2003, pp. 14-15.

[5] 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, 200, pp. 797-814.

[6] S. Gardiner, ed. *Testing Safety-Critical Software*, Springer-Verlag, London, 1999.

[7] 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.

[8] S. D. P. Harker, K. D. Eason, and J. E. Dobson, "The Change and Evolution of Requirements as a Challenge to the Practice of Software Engineering," *Proc. IEEE Intl Symp on Requirements Eng*, IEEE Computer Society, Los Alamitos, CA, 1992, pp. 266-272.

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

[10] 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.

[11] R. Lutz, "Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems," *Proc IEEE Intl Symp Req Eng*, IEEE CS Press, 1993, pp. 126-133.

[12] R. Lutz and I. C. Mikulski, "Operational Anomalies as a Cause of Safety-Critical Requirements Evolution," *The Journal of Systems and Software*, to appear.

[13] R. Lutz and I. C. Mikulski, "Requirements Discovery during the Testing of Safety-Critical Software," *Proc of Int'l Conf on Software Eng, 2003*, to appear..

[14] 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

[15] A. van Lamsweerde and E. Letier, "Handling Obstacles in Goal-Oriented Requirements Engineering," *IEEE Trans on Software Eng*, vol. 26, no. 10, Oct. 2000, pp. 978-1005.

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