An Empirical Investigation of Fault Repairs and Mitigations in Space Mission System Software

Abstract—Faults in software systems can have different characteristics. In an earlier paper, the anomaly reports for a number of JPL/NASA missions were analyzed and the underlying faults were classified as Bohrbugs, non-aging-related Mandelbugs, and aging-related bugs. In another paper the times to failure for each of these fault types were examined to identify trends within missions as well as across the missions. The results of these papers are now starting to provide guidance to improve the dependability of space mission software.

Just as there are different types of faults, there are different kinds of mitigations of faults and failures. This paper analyzes the mitigations associated with each fault studied in our previous papers. We identify trends of mitigation type proportions within missions as well as from mission to mission. We also look for relationships between fault types and mitigation types. The results will be used to increase the reliability of space mission software.

I. INTRODUCTION

The increasingly better hardware in spacecraft systems makes it possible to plan and design more ambitious missions. This naturally leads to more sophisticated and complex software being developed to achieve the mission goals [1]. However, the growing complexity of flight software makes it more prone to suffer software failures. Hence, the ability of spacecraft systems to deal with software failures and the underlying software faults during operations is a key aspect determining the success or failure of a mission.

Ambitious missions also require spacecraft systems to run autonomously, or with minimum human intervention. Spacecraft systems need to make autonomous decisions during planetary missions, because there often is no time to wait for the round-trip message delay. Moreover, the bandwidth available is limited, which would in turn require the raw data collected to be preprocessed on-board to reduce their volume. As a consequence, spacecraft systems have to make use of autonomous software fault tolerance mechanisms to deal with software failures and their underlying faults during operations.

In order to improve the software fault tolerance mechanisms deployed on spacecraft systems, and to develop more reliable spacecraft systems, it is necessary to study the software failures experienced, and the software faults responsible for them.

Grottke and Trivedi [2], [3], [4] proposed to classify software faults based on their inherent characteristics into Bohrbugs, non-aging related Mandelbugs, and aging-related bugs. This can be considered as a sub-classification of operational, external, human-made, non-malicious, non-deliberate software faults in the context of Avižienis et al. [5]. Chillarege [6] used ODC Triggers to classify Bohrbugs and Mandelbugs. While he obtained similar results as the classification based on the definitions presented in [4], his classification related to specifics of the failure occurrence observed, rather than the inherent properties of the fault itself.

We here give a brief definition of each software fault type, to contextualize the work presented by us in this paper. The term Bohrbug (BOH) was coined by Gray [7] in 1985. It refers to a fault that is easy to isolate and whose manifestation is consistent under a well-defined set of conditions.

In contrast to Bohrbug, the term Mandelbug refers to a fault whose behavior seems to be “non-deterministic.” This means that typically a Mandelbug is difficult to isolate, and failures caused by it are hard to reproduce. Grottke and Trivedi [2], [3], [4] traced these characteristics of a Mandelbug to the complexity of its activation and/or error propagation. Mandelbugs are thus intrinsically related to software complexity: The more complex a piece of software, the higher the risk of its containing a large number of Mandelbugs.

A sub-type of Mandelbugs is responsible for the software aging phenomenon [8], i.e., an increasing failure rate or progressively degrading performance, observed in many kinds of long-running systems [9], [10], [11], [12], [13]. A so-called aging-related bug is able to cause this phenomenon because the rate with which it is activated and/or the rate with which errors caused by it are propagated into (partial) failures increases with the total time the system has been running. Such an increasing error propagation rate is often caused by the accumulation of internal error states.

Mandelbugs can thus be divided into aging-related bugs (ARBs) on the one hand and those Mandelbugs that are not capable of causing software aging, known as non-aging-related Mandelbugs (NAMs), on the other hand. Recent papers studied the software failure reports of 18 different JPL/NASA space missions. Grottke et al. [14] classified the underlying software faults using the above-mentioned fault types. The overall percentages of BOH, NAM, and ARB were 61.4%, 32.1%, and 4.4%, respectively. 2.2% of the faults could not be classified. In [15], we studied the nature of the times to software failure for those eight JPL/NASA missions.
with a substantial number of software faults. Moreover, we analyzed the reliability growth of the software during the missions; for those missions that did not show any trend, we tried to determine the underlying distribution of the times to failure. We observed that reliability growth detected in the time to failure datasets containing both Bohrbugs and non-aging-related Mandelbugs could often be attributed to one of the two fault types, or that it could even be a mere artifact of pooling the data. Likewise, separating between Bohrbugs and non-aging-related Mandelbugs helped to understand the decreasing failure rates featured by the best-fitting models for the times to failure based on the combined Bohrbug/non-aging-related Mandelbug datasets.

The software fault classification is not only theoretical, it also has practical importance. Each type of software fault requires different types of mitigation mechanisms to deal with it and its consequent failures during development, testing, and operations [4], [16].

Typically, Bohrbugs may easily be isolated and removed during testing. Design diversity (i.e., failure to nonidentical) can prevent residual Bohrbugs in operational software from causing failures, as long as the different software implementations do not contain Bohrbugs activated by the same inputs.

For Mandelbugs, which are difficult to remove in the testing phase, design diversity can help as well during operations. However, due to the seemingly non-deterministic behavior of Mandelbugs, it is possible that an operation which previously failed because of a Mandelbug will work perfectly on re-execution. Therefore, techniques like software replication (failover to a standby with an identical software copy), retrying the operation, restarting the application, or rebooting the physical system can be effective to deal with Mandelbugs.

In the special case of aging-related bugs, future failures can be prevented via software rejuvenation approaches [17]. Software rejuvenation is a proactive “maintenance operation” focused on stopping the system, cleaning up the internal state, and restarting the system to an initial state.

The previous examples are mainly software failure mitigation approaches. On the other side, there are approaches based on software fault/error mitigation. These approaches are applicable to any type of software fault. Well-known examples of software fault/error mitigation approaches are fix or patch, workaround, and use as is. The upper portion of Figure 1 shows the Bohr- and Mandelbug fault types described earlier. The lower part of Figure 1 allocates the different fault/error and failure mitigation approaches to these types of faults.

In this paper, we study the failure reports of those eight JPL/NASA missions with a significant number of faults, already analyzed in [14] and [15]. We conduct a classification of the mitigation mechanisms applied by the operators to deal with software faults and failures during operation. We are interested in the proportions of the different mitigation approaches per mission and across the missions. We have studied the frequency of each technique in order to determine if any of the techniques (or a combination of them) is especially effective to deal for a given type of software fault. The ultimate goal is to define guidelines for developers, testers, and finally operators to develop more reliable spacecraft software systems, and to improve the reliability of the spacecraft systems during operation. The current paper allows us to identify the most effective techniques to mitigate faults/errors and failures according to the type of software fault encountered by the operators. This paper will give insights on which techniques need to be applied in the first place by the operators to recover the normal behavior of the system, reducing the recovery time and thus increasing the availability of the spacecraft system. Furthermore, the conclusions obtained will provide us with information about what mechanisms are most suitable for implementation as automatic software fault tolerance mechanisms to deal with faults and failures during operation.

The rest of the paper is organized as follows: Section II reviews the related work in analyzing failure reports and fault
and failure mitigation techniques. Section III describes our approach to classify the different software failure reports based on the mitigation technique employed. In Section IV, we present the results obtained from the classification. Finally, Section V discusses these results and concludes the paper.

II. RELATED WORK

Several papers have addressed the study of complex system failures from different perspectives [7], [14], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30]. In [7], [18], [24] and [26], sources of failures of complex systems are classified into hardware, software and human/operator errors. These studies concluded that hardware represents 10–30% of the sources of failures, human errors represent 30–50%, and software represents 20–50%. If we analyze the evolution over time (from [7] to [26]), it is clear that software is becoming the main cause of failures in complex systems. However, hard disk failures have received special attention: The authors of [21], [29] and [30] have focused on understanding the causes of the hard disk failures, and on correlating them with different hard disk monitoring parameters.

We have found only few papers in which failure reports collected during system operations are analyzed [14], [19], [25], [27], [28]. In [25], [27], [28], failure reports from different JPL/NASA missions were studied and classified. These studies highlighted that the main causes of spacecraft failures were flight and ground software. However, they did not extend the analysis of software failures. To fill in this gap, Grottke et al. [14] classified the software faults underlying the flight software failure reports according to the software fault types presented in Section I; this paper revealed that Bohrbugs represented the largest share of faults and the main cause of software failures experienced, followed by non-aging-related Mandelbugs. Käähnie and Kanou [19] analyzed the failure reports from a commercial telecommunications system for understanding the main causes of system failures. However, to the best of our knowledge there has not been any paper in which the fault repair and mitigation approaches applied during operation have been studied or classified. The current paper tries to shed some light onto this topic.

III. MITIGATION CLASSIFICATION PROCESS

We examined the same failure reports already analyzed in [14] and [15]. These reports, related to eight JPL/NASA spacecraft missions, were completed by the operators during mission operations. Each report contains the description of the incident, the analysis, verification and real-time action conducted by the operators from the ground, and also includes the corrective action applied (if any) to mitigate the failure and/or its underlying fault. The information supplied by the operators was used to classify the mitigating actions taken.

The mitigation types are based on those shown in Figure 1, which postulates relationships between different fault types (i.e., Bohrbugs and Mandelbugs) and specific types of mitigating actions. These mitigation types, used in classifying the failure reports, are described in Table I.

It is important to note that the mitigating actions we identified are those actions taken by members of the ground-based mission operations team in response to the reported failures. At this point we did not classify any autonomous action that may have been taken by the spacecraft, leaving that for future work.

This section describes how the classification was conducted to connect the aforementioned mitigation types with real failure reports. Table II presents five typical failure reports and the mitigating action(s) taken in response by the operators. Although the failure reports have been sanitized to provide anonymity, the information relevant for classifying the associated faults and mitigating actions have been preserved.

The first failure report in Table II describes a BOH that was mitigated by changing the source code. An on-board
<table>
<thead>
<tr>
<th>Description of incident</th>
<th>Analysis, verification, and/or real-time action</th>
<th>Final corrective action</th>
<th>Fault and mitigation classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial telemetry from on-board software Version XX indicated that Emergency Antenna was set to value B. This parameter in System Fault Protection should be value A. Uplink and Downlink communication at this point in the mission is not possible with value B. If the spacecraft executed Safing at this time, it would require the Command Loss Response Algorithm to execute an antenna configuration swap to value A in order to communicate with the spacecraft. (About TT time).</td>
<td>A real-time command was sent to the spacecraft to change the RAM configuration to value A. (Prime String Only) The Online String is currently running Version YY with a RAM value of value A. Both Version YY and Version XX in on-board storage require an update to make the Emergency Antenna value A. On mm-dd-yyyy commands were sent to patch the FSW loads on both on-board storage devices.</td>
<td>All FSW copies now contain the correct Safing antenna value. Update of the Emergency Antenna parameter is being addressed by PFR WWWWW and this anomaly report can be closed.</td>
<td>BOH, Fix/Patch</td>
</tr>
<tr>
<td>A file named “filename” was found on the Flyby file system as we prepared for the planet watch activity. Prior AutoNav activities are supposed to conclude by copying and deleting this file – opnav file should not exist at the beginning of an AutoNav activity. Directory timestamp indicates opnav file was modified on the preceding planet watch activity.</td>
<td>New command written and tested to delete file from the flyby spacecraft.</td>
<td>mm-dd-yyyy AutoNav Uplink Check-list now has an item to check that each AutoNav activity has a command to delete the file afterwards.</td>
<td>BOH, Workaround</td>
</tr>
<tr>
<td>Starting with SEQ SEQ_NAME, there is much unexpected activity (looks like the FP attitude estimate repair sequence), apparently competing with the sequences SEQ_NAME and SEQ_NAME2, because the commands are interleaved. This causes unacceptable conflicts in the ADCS configuration. Initial assessment is that FP responses stop the critical sequence, but since COMPONENT_NAME autonomously kicks off the sequences that specifically perform the Activity_Type_A5, these end up running in parallel. For example, the below set of commands (from the integrated log for test X) shows a COMPONENT_NAME power cycle at time T1, but a delta-v is still commanded at time T2. This is completely inconsistent with the COMPONENT_NAME Activity_Type_A design, and known it takes for COMPONENT_NAME to perform a warm re-start. (test log follows).</td>
<td>Confirmed that there is a sneak path in the software logic for a latent type A request to be fulfilled in the middle of a fault response. FSW SCR XXXX was opened to implement a software fix to prevent this from occurring. Will retest with faulted COMPONENT_NAME encounter runs.</td>
<td>This MCR was implemented in FSW VV, and loaded on the S/C on mm-dd-yyyy. MCR YYYY to be included in FSW VV Test of MCR below - Change/Command Requested: Bracket Encounter System Safing (Return to point) with mode block for delta-v. Reason for Change/Command: On the COMPONENT_NAME, transition to delta-v state is done via an asynchronous sequence call from mode manager. Faulted testing has show it possible for an Activity_Type_A to occur in the middle of point state recovery. Note there is a code change involved with this MCR. However, the change is benign on the flyby as the flyby does not use the mode manager for delta-v execution. Impact if Not Implemented: Unstable performance, improperly executed delta-v maneuver. Possible mission degradation. Implementation Approach: (Implementation details follow).</td>
<td>NAM, Fix/Patch</td>
</tr>
<tr>
<td>Instrument A suddenly produced anomalous housekeeping data at about mm-dd-yyyy hh:mm:ss.</td>
<td>Analysis of instrument operation and instrument data and flight software showed that this is an artifact of the current instrument FSW, which causes this anomaly under certain conditions.</td>
<td>A procedural work-around has been developed which has fixed this problem. Instrument is re-initialized using a “Fast Housekeeping” command, which sets all conditions to a known state. This problem has not been observed since the procedures have been revised.</td>
<td>NAM, Workaround</td>
</tr>
<tr>
<td>On mm-dd-yyyy hh:mm:ss Instrument A stopped generating valid HK and science packets. The last valid HK packet received indicated nominal instrument. After successful FSW check-out, HK SW status counters indicated two FSW exceptions. After the FSW load no valid science packets were generated until the CPU-reset command, after which both nominal HK and science data were generated. But then at mm-dd-yyyy hh:mm:ss the instrument stopped producing valid HK. (Telemetry log excerpt) The CPU-Reset command was uplinked at mm-dd-yyyy hh:mm:ss. Operation as expected until mm-dd-yyyy hh:mm:ss the instrument stopped producing valid HK only. The instrument software onboard has been compared with the ground SW. No difference has been identified. The FSW version X behavior is reproducible on ground: After many hours, no HK packets are written to the RTIU. Assumption: The program stack has been too full.</td>
<td>This was a flight software related issue. We found some inconsistencies within the FSW code and did according corrections. Now the FSW version Y runs stable over many months. We did not have this problem again. Recommend closure of this anomaly report.</td>
<td>ARB, Fix/Patch</td>
<td></td>
</tr>
</tbody>
</table>
sequence encountered an unexpected value for one of the parameters it read from RAM; the problem was repaired by changing the value of the parameter stored in spacecraft memory. The second failure report also describes a BOH; however, this one was mitigated using a workaround rather than a fix. After the completion of an on-board operation, a file that was expected to have been deleted was found to have remained in the spacecraft file system. Rather than modifying the on-board software to ensure intermediate file deletion, a workaround was developed to ensure that the operations engineers delete it after the completion of each such on-board operation by transmitting the appropriate command to the spacecraft. The next two failures had each been caused by a NAM. The first one was mitigated by modifying the source code – the fault is a complicated one involving the potential interleaving of instructions in different on-board sequences in such a way that the ability of the spacecraft to achieve its science goals could be seriously degraded. In this case, the severity of the consequences was high enough to warrant repairing the fault; the operations engineers were able to develop an understanding of the failure mechanism with enough confidence, which enabled them to make the fix. The second NAM was mitigated with a workaround: One of the instruments on the spacecraft produced anomalous data; detailed analysis of the instrument’s software and method of operation identified previously unknown circumstances under which this failure would occur. The failure was deemed to be sufficiently infrequent, and to not threaten the mission’s overall science goals so a workaround was determined to be sufficiently effective and involve less risk than a change to the source code. The final failure had been due to an ARB that was mitigated with a fix – apparently the on-board software for an instrument had a gradual stack overflow that eventually stopped the production of a specific type of downlink telemetry. The operations team was able to identify and reproduce the problem, and then uplink a modified version of the flight software to the instrument.

The classification of the mitigating actions resulting from each failure occurrence was based on such descriptions. It was conducted in parallel by two persons. After that, the classifications were compared to resolve disagreements. In approximately 1/3 of the cases, the initial classifications by the two individuals did not match. In these cases, a detailed analysis was conducted by the two individuals to achieve a consensus. In all the cases, both investigators were able to reach a final classification agreement.

IV. MITIGATION TYPE FREQUENCY ANALYSIS

Based on the above classification, we categorized the failure reports related to the flight software of the JPL/NASA missions as shown in Table III. The first column in this table lists 20 distinct Corrective Action Sets, each of which contains one or more instances of one or more of the types of mitigating actions defined in Table I. A Corrective Action Set simply identifies the types of mitigating actions associated with the disposition of an individual failure report. From left to right, the bits in the Corrective Action set vector denote whether the mitigation types shown in Table I are contained in the corrective actions for a failure report. For example, the first row in Table III indicates corrective actions containing only fixes/patches. The fifth row indicates corrective actions containing fixes/patches and reboots. Finally, the corrective action set in the sixth row applies to those failures for which mitigating actions could not be identified. The Corrective Action Sets are a high-level abstraction of the information contained in the failure reports; they do not contain any information about the order in which the mitigating actions were taken, or the number of actions of each mitigation type that were performed.

For each Corrective Action Set, the relative frequency with which it occurs for BOHs, NAMs, and ARBs is noted in the first three columns of Table III. The last two columns show the relative frequencies for all Mandelbugs (i.e., for NAMs and ARBs taken together), as well as for all three fault types taken together.

The last row of this table indicates the total number of faults of each type present in the data we analyzed.

For each fault type – BOH, NAM, and ARB – the most frequent 75% (or more) of the Corrective Action Sets contain only a single type of mitigating action. The related mitigation types are Fix/Patch, Workaround (WA) and Use as is (UAI). For our subsequent analysis, we collapsed the Corrective Action Sets shown in Table III down to three partially overlapping sets: those Corrective Action Sets containing fixes/patches, those containing workarounds, and those containing a “use as is” disposition. The sets are partially overlapping in that a failure report can lead to mitigation actions of several different

<table>
<thead>
<tr>
<th>Corrective Action Set</th>
<th>BOH</th>
<th>NAM</th>
<th>ARB</th>
<th>BOH+ ARB</th>
<th>NAM+ ARB</th>
<th>BOH+ NAM+ ARB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,0,0,0,0,0,0,0,0,0,0,0</td>
<td>0.687</td>
<td>0.250</td>
<td>0.565</td>
<td>0.288</td>
<td>0.530</td>
<td></td>
</tr>
<tr>
<td>0,0,0,0,0,0,0,0,1,0,0,0</td>
<td>0.114</td>
<td>0.310</td>
<td>0.130</td>
<td>0.288</td>
<td>0.183</td>
<td></td>
</tr>
<tr>
<td>0,0,0,0,0,0,1,0,0,0,0,0</td>
<td>0.079</td>
<td>0.201</td>
<td>0.175</td>
<td>0.197</td>
<td>0.126</td>
<td></td>
</tr>
<tr>
<td>1,0,0,0,0,0,1,0,0,0,0,0</td>
<td>0.086</td>
<td>0.054</td>
<td>0.043</td>
<td>0.053</td>
<td>0.073</td>
<td></td>
</tr>
<tr>
<td>1,0,1,0,0,0,0,0,0,0,0,0</td>
<td>0.007</td>
<td>0.067</td>
<td>0.0</td>
<td>0.058</td>
<td>0.027</td>
<td></td>
</tr>
<tr>
<td>0,0,0,0,0,0,0,0,0,0,0,1</td>
<td>0.007</td>
<td>0.030</td>
<td>0</td>
<td>0.026</td>
<td>0.014</td>
<td></td>
</tr>
<tr>
<td>1,0,1,0,0,0,0,0,0,0,0,0</td>
<td>0.003</td>
<td>0.024</td>
<td>0</td>
<td>0.021</td>
<td>0.010</td>
<td></td>
</tr>
<tr>
<td>0,0,0,0,0,0,1,0,0,0,0,0</td>
<td>0</td>
<td>0.006</td>
<td>0.043</td>
<td>0.010</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>0,0,0,1,0,0,0,1,0,0,0,0</td>
<td>0</td>
<td>0.006</td>
<td>0.043</td>
<td>0.010</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>0,0,1,0,0,0,0,0,1,0,0,0</td>
<td>0.003</td>
<td>0.006</td>
<td>0</td>
<td>0.005</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>1,0,0,0,0,0,0,1,0,0,0,0</td>
<td>0.003</td>
<td>0.006</td>
<td>0</td>
<td>0.005</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>0,0,0,0,0,1,0,0,0,0,0,0</td>
<td>0</td>
<td>0.006</td>
<td>0</td>
<td>0.005</td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td>0,0,0,1,0,0,0,0,1,0,0,0</td>
<td>0</td>
<td>0.006</td>
<td>0</td>
<td>0.005</td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td>0,0,1,0,0,0,0,0,0,0,0,0</td>
<td>0</td>
<td>0.006</td>
<td>0</td>
<td>0.005</td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td>0,1,0,0,0,0,0,0,0,0,0,0</td>
<td>0</td>
<td>0.006</td>
<td>0</td>
<td>0.005</td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td>1,0,0,0,0,0,0,0,1,0,0,0</td>
<td>0</td>
<td>0.006</td>
<td>0</td>
<td>0.005</td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td>1,0,0,0,0,0,1,0,0,0,0,0</td>
<td>0</td>
<td>0.006</td>
<td>0</td>
<td>0.005</td>
<td>0.002</td>
<td></td>
</tr>
</tbody>
</table>

| Number of faults over all missions | 288 | 164 | 23 | 187 | 475 |
types, and therefore more than one of the three sets may apply to it. We also consider those sets containing WA and/or UAI to observe the proportion of mitigating actions which included at least one of these techniques, as opposed to those actions which did not include either of them. Although there was some overlap between the Corrective Action Sets, Table III indicates that this overlap was not substantial.

Table IV lists the relative frequencies with which the corrective actions for each type of fault (BOH, NAM, or ARB) included the three types of mitigating actions. Because the Corrective Action Sets overlap, the sum of the frequencies for three distinct mitigation types Fix/Patch, WA, and UAI will total more than 1, as will the sum of the frequencies for the mitigation types Fix/Patch and WA+UAI.

Table V shows this same information for two combined sets of faults – NAM and ARB taken together, then BOH, NAM, and ARB taken together. Our goal in this analysis is to determine whether the frequencies of mitigation types are similar or distinguishable for the different kinds of faults.

Table IV indicates that for the three fault types the frequencies of each of the three mitigation types are different. Over 75% of the mitigating actions associated with BOHs include a fix, while less than half those associated with NAMs do so. By contrast, little more than 25% of the mitigating actions associated with BOHs include a workaround or a “use as is,” while this is the case for 60% of those associated with NAMs.

Note that the frequency with which the mitigating actions associated with an ARB include development of a workaround is close to that of NAMs. However, while the frequency with which mitigating an ARB involves fixes is much lower than that for BOHs, it is considerably higher than that for NAMs. Similarly, “use as is” dispositions are much rarer for ARBs than for NAMs. We will discuss this in more detail in Section V.

We next examine the proportions of mitigation type frequencies for individual missions to determine whether the frequencies shown in Table IV and Table V change from mission to mission. The details are contained in Tables VI–IX. Columns 2–4 of these tables give the frequencies of mitigating actions including the named type; column 5 gives the frequencies of mitigating actions that include either workaround, or “use as is”, or both.

Table VI lists the frequencies of each mitigation type for Bohrbugs in all eight missions. For every mission, we see that mitigating actions involving fixes occur more frequently than those involving workarounds or “use as is” dispositions. In fact, with the exception of Mission 6, fixes occur more frequently than WA+UAI mitigation types.

Table VII shows the frequencies of each mitigation type for non-ageing-related Mandelbugs in all eight missions. For most missions, workaround and “use as is” mitigation types occur at similar frequencies to fixes. However, for all missions except Mission 8 both workarounds and “use as is” mitigation types taken together occur significantly more often than fixes.

Our findings are similar when we consider NAMs and ARBs taken together, as seen from Table VIII. The main difference between NAMs alone and NAMs and ARBs together appears
to be that ARBs are fixed at a greater frequency than NAMs, contributing to a smaller difference between the frequencies of WA and UAI mitigation types taken together and fixes. ARBs are not considered separately on a mission-by-mission basis because the number detected per mission is too small to perform any meaningful analysis – there is total of 25 ARBs over all missions.

Table IX shows the results of considering all three types of faults together. Taking into account that the Mission IDs used in this paper reflect the launch order (i.e., Mission 1 is the first one launched, while Mission 8 is the most recent one), the table seems to suggest a difference between the earlier four missions and the four later ones. For Missions 1–4, Fix/Patch and WA+UAI occur at roughly the same frequencies. However, there is no such pattern for the later four missions. Faults are fixed substantially more often for Missions 5, 7, and 8, while WA and UAI types taken together occur at significantly greater frequency than fixes for Mission 6.

To further investigate the possible differences between the first four missions and the last four with respect to the mitigating actions taken, we aggregate the data as shown in Table X. This table is similar in layout to Tables VI–IX. Instead of looking at individual missions, however, it groups together Missions 1–4 and Missions 5–8. For each type of fault, we see that the proportion of fixes for the first four missions is smaller than the proportion of fixes for the last four.

While the proportion of WA+UAI taken together decreases for BOH and for all three fault types taken together, it remains nearly the same for NAM and NAM+ARB.

V. DISCUSSION AND CONCLUSION

For the set of failure reports studied in this paper, we have observed different types of mitigating actions taken in response to different types of faults. The differences appear to be consistent across the eight missions for which we have analyzed failure reports, increasing the likelihood that our findings will be applicable to future similar missions.

Our finding that most Bohrbugs are mitigated via fixes, combined with the earlier paper that suggested that Bohrbugs appear to be the type of faults most commonly encountered during mission operations [14] indicate that maintenance development will continue to be an important part of mission operations activities for future missions.

It may come as a surprise that fixes are the most frequent type of mitigating action taken not only for Bohrbugs, but for non-aging-related Mandelbugs and aging-related bugs as well. Given that these latter types of faults are difficult to isolate, it would be reasonable to suppose that developing workarounds would be a more common mitigation mechanism. However, workarounds often involve losing access to some portions of a system’s functionality. Considering that the systems we analyzed are one-of-a-kind systems deployed with the explicit goal of acquiring high-value (and high-visibility) scientific observations, it appears that in this case the development and mission operations organizations make it a high priority to be able to identify and remove these types of faults during mission operations. Moreover, the spacecrafts on which the flight software is running log a huge amount of system parameters. JPL/NASA operations personnel can analyze these traces, play them back on test equipment, etc. Unlike in many other industrial settings, it is thus often possible to determine the root causes even for failures due to non-aging-related Mandelbugs and aging-related bugs, making fixes feasible.

We have also seen that as compared with non-aging-related Mandelbugs, aging-related bugs are fixed/patched more often, while “use as is” dispositions are less frequent. This may be explained by the fact that, although the error propagation of aging-related bugs is complex due to the interaction with the system-internal environment, aging effects (such as the buffer overflow described in the fifth failure report listed in Table II) are rather easy to detect from system logs, which often allows the underlying fault to be identified and fixed. Furthermore, while aging-related bugs could be dealt with proactive reboots (i.e., software rejuvenation) for the missions analyzed in this paper the JPL/NASA operations personnel did not employ such techniques – either because they were unaware of them, or because they rather decided to rely on fixes/patches.

Finally, our analyses have revealed that, for each type of fault, the earlier Missions 1–4 tend to show lower frequencies of fixes/patches than the more recent missions Missions 5–8. We conjecture that improved logging of system parameters may have further increased the ability of the operations
personnel to identify and correct the faults responsible for the failures experienced. Moreover, we observed that during the last period of a mission, proposed patches tend to get discarded, because often the effort is deemed to outweigh the benefit due to the short remaining lifetime of the mission. This might bias the fix/patch frequencies for the missions launched earlier. As the more recent missions continue their operations, we will carry on with our analysis of the mitigation types to determine whether or not the observed higher frequency of fixes for these missions is a characteristic of the missions as a whole or an artifact of the smaller amounts of time they have been in operation.

In future work, we will also examine the ground-based support systems for the missions studied in this paper, to investigate if the mitigating actions taken in response to each fault type differ from those encountered for the flight software.

ACKNOWLEDGMENT

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology and at Duke University. The research was sponsored by the National Aeronautics and Space Administration’s Office of Safety and Mission Assurance Software Assurance Research Program. This task is managed locally by JPL’s Assurance Technology Program Office. Duke research was supported in part by the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) under the JPL subcontract #1144019. FAU research was supported by the Dr. Theo and Friedel Schoeller Research Center for Business and Society.

REFERENCES


