An Empirical Investigation of Fault Types in Space Mission System Software

Michael Grottke
University of Erlangen-Nuremberg
Michael.Grottke@wiso.uni-erlangen.de

Allen P. Nikora
Jet Propulsion Laboratory,
California Institute of Technology
Allen.P.Nikora@jpl.nasa.gov

Kishor S. Trivedi
Duke University
kst@ee.duke.edu

Abstract

As space mission software becomes more complex, the ability to effectively deal with faults is increasingly important. The strategies that can be employed for fighting a software bug depend on its fault type. Bohrbugs are easily isolated and removed during software testing. Mandelbugs appear to behave chaotically. While it is more difficult to detect these faults during testing, it may not be necessary to correct them; a simple retry after a failure occurrence may work. Aging-related bugs, a sub-class of Mandelbugs, can cause an increasing failure rate. For these faults, proactive techniques may prevent future failures.

In this paper, we analyze the faults discovered in the on-board software for 18 JPL/NASA space missions. We present the proportions of the various fault types and study how they have evolved over time. Moreover, we examine whether or not the fault type and attributes such as the failure effect are independent.

1. Introduction

Experience has shown that all but the simplest software systems contain faults (often called bugs, or defects). For describing characteristics of software faults that cause failures during testing and operation, practitioners and researchers sometimes refer to “Bohrbugs,” “Heisenbugs,” “Mandelbugs,” and “aging-related bugs.” However, there are no consistent definitions for most of these terms, and often the terms are used without any explicit definition. In recent research [10], [11], [12], we have therefore tried to define the terms as precisely as possible. Throughout this paper we use the following definitions:

- Bohrbug := An easily isolated fault that manifests consistently under a well-defined set of conditions, because its activation and error propagation lack “complexity” as defined below.
- Mandelbug := A fault whose activation and/or error propagation are complex. "Complexity" can be caused by
  1. a time lag between the fault activation and the occurrence of a failure; or
  2. the influence of indirect factors, i.e.,
     a) interactions of the software application with its system-internal environment (hardware, operating system, other applications); or
     b) influence of the timing of inputs and operations (relative to each other, or in terms of the system runtime or calendar time); or
     c) influence of the sequencing of operations; sequencing is considered influential, if the inputs could have been run in a different order and if at least one of the other orders would not have led to a failure.

  Typically, a Mandelbug is difficult to isolate, and/or the failures it causes are not systematically reproducible. Mandelbug is the complementary antonym of Bohrbug; i.e., each fault is either a Mandelbug, or a Bohrbug.
- Aging-related bug := A fault that is capable of causing an increasing failure rate and/or degraded performance. There are two possible reasons for this phenomenon:
  1. The fault causes the accumulation of internal error states.
  2. The activation and/or error propagation of the fault is influenced by the total time the system has been running.

  Aging-related bugs are a sub-type of Mandelbugs.

According to the relationships between the fault types, each fault is a Bohrbug (BOH), a non-aging-related Mandelbug (NAM), or an aging-related bug (ARB).

As we define them, Bohrbugs basically correspond to solid [6] or hard [2] faults, while Mandelbugs are soft [6] or elusive [2] faults, which Gray [6] also refers to as Heisenbugs. However, there are subtle differences. Our definitions do not classify a fault according to its behavior in one specific case (like the failure occurrence that led to the detection of the fault). Rather, we focus on the *potential* manifestation characteristics. A fault is categorized as a Mandelbug if it is *capable* of causing failures that are not systematically reproducible. Likewise, a fault that is *able* to cause an increasing failure rate is an aging-related bug even if it is detected during code inspection and has never had the “chance” to actually make a software system age. Our fault classification is thus related to *inherent* properties of the software fault. The distinction between potential and actual fault behavior is similar to the distinction between trigger and symptom in Orthogonal Defect Classification [4].

In [10], we showed that Lindsay invented the term Heisenbug in the 1960s to refer to faults that stop causing a failure or that manifest differently when one attempts to probe or isolate them. Programmers also use this term in the same manner [19]. Unlike Gray [6], we therefore do not equate Heisenbugs with soft faults.

Despite these theoretical refinements, classifying software faults into Bohrbugs and Mandelbugs will usually not lead to a grossly different result than categorizing them into hard and elusive faults.

Fault classification is of practical interest, because the likelihood of being able to detect and remove the faults during development and testing, as well as the possible strategies for dealing with residual faults during mission operations depend on the fault type. Bohrbugs are the easiest to find during testing. Fault-tolerance for an operational system in the presence of Bohrbugs can mainly be achieved with design diversity, since they deterministically cause failures. Due to their complex behavior, Mandelbugs are more difficult to find, isolate, and correct during testing. Since the re-execution of an operation that failed because of a Mandelbug will often not result in another failure, Mandelbugs can be handled with software replication or – if failure occurrences can be tolerated – with simple retries or more sophisticated approaches like checkpointing and recovery-oriented computing [3]. More specifically, for aging-related bugs, for which the tendency of causing a failure increases with the system run-time, proactive measures that clean the internal system state and thus reduce the failure rate are useful. This kind of “preventive maintenance” is referred to as “software rejuvenation” [15].

It is generally assumed that due to the complex behavior of Mandelbugs (including aging-related bugs) the majority of software faults remaining after thorough testing belong to this class. Most of the authors making this claim support it by referring to a small collection of empirical studies [1], [6], [16]. However, the evidence contained in these studies is much less conclusive than asserted. For example, Gray and Siewiorek [7] state with respect to Adams' study of maintenance records of North American IBM systems [1]: “Some software faults were reported many times, but such virulent bugs made up significantly less than 1 percent of all reports.” While Adams indeed observed that many faults merely affected few users, this does not necessarily mean that these faults were Mandelbugs. Rather, it is possible that a large percentage of these faults were Bohrbugs located in parts of the software executed only by users with an unusual operational profile. (Conversely, some of the faults reported by many users may have been Mandelbugs.) The evidence concerning the percentage of Bohrbugs and Mandelbugs in software systems presented in other studies is similarly inconclusive.

In an ongoing project, we are therefore classifying software faults from current as well as historical JPL missions to gain a better understanding of the proportions of the different types of faults in flight software and ground software. Establishing a baseline will enable JPL to develop techniques and guidelines for (i) improving the detectability of software faults, (ii) masking the effects of faults, and (iii) identifying components most likely to contain difficult-to-detect critical software faults.

In this paper, we present our analysis of the faults discovered in the flight software for 18 space missions. We specifically examine:

- the relationships between fault type and further characteristics, like failure effect, and failure risk;
- differences in the fault type proportions across missions; and
- the development of the fault type proportions within a mission, as the mission duration increases.

The paper is organized as follows: Section 2 reviews related work in analyzing software faults across multiple projects. In Section 3, we describe our approach to classifying faults, before presenting our analysis of fault data in Section 4. The results are discussed in the light of other ongoing research in Section 5. Section 6 concludes the paper.

2. Related work

JPL collects information about incidents of unexpected behavior of space systems in an institutional anomaly reporting system. While many of the anomalies represent system failures (i.e., incorrect system behavior), some of them are based on misunderstandings of how the system is supposed to work. Potential causes for observed space system failures include procedural errors, faults in the hardware or software of the ground system, as well as faults in the hardware or the software of the flight system.

Studies of space system anomalies [8], [9], [13] have previously been conducted at JPL as part of the Ultra-reliability (UR) Program, sponsored by the NASA Office of Safety and Mission Assurance (OSMA). Knowledge gained from these studies includes a better understanding of the relative proportions of different categories of anomalies (e.g., operator misunderstandings vs. failures due to hardware faults vs. failures due to software faults), the times at which anomalies are more likely to be observed during mission operations, and the types of corrective action most frequently taken in response to observed anomalies. For example, Green et al. [8] and Hoffman et al. [13] analyzed in-flight anomalies reported for the Voyager and Galileo missions. Their findings included the result that at least for Galileo, software was a significant generator of failures; flight and ground software together were responsible for somewhat over 40% of the in-flight anomalies recorded in the JPL Problem Reporting System. In [9], Green et al. analyzed in-flight anomalies reported for seven current and historic Mars exploration missions. Since recent missions to Mars have included both orbiter and lander spacecraft, and since several missions were not successful upon reaching Mars, the time period for comparing the missions was limited to the span covering the launch of each spacecraft to its arrival at Mars. Their results indicated that for the missions analyzed, software was a significant source of anomalous behavior, as for the Galileo mission; the proportion of anomalies due to flight and ground software combined ranged from just under 1/2 to nearly 2/3, with later missions having a higher proportion of anomalies being due to software. However, Green et al. did not go into further detail regarding different types of software faults.

3. Approach

This paper focuses on 18 historic and ongoing JPL/NASA missions. Seven of these missions were related to earth orbiters, while eleven others were planetary missions, for which the destination was one or more astronomical bodies beyond the Earth-Moon system. The destination for seven of these planetary missions was Mars, one was an Outer Planets mission, two were targeted to comets, and one returned samples of the Solar wind.

For these missions, over 13,000 anomaly reports recorded after deployment of the respective space system from the developing facility were collected from JPL's institutional Problem Reporting System. Based on the value of the multiple-choice "Cause" field in the anomaly reports, 653 of them were identified as having been related to the flight software of our 18 missions. We further analyzed each of these anomalies by the simple but laborious process of reading the textual descriptions of the anomaly, its analysis and verification, and the corrective action taken. 76 anomalies turned out not to be failures caused by flight software faults, but had causes such as operator mistakes or incorrect operating procedures; another 57 anomalies were identified as being related to software faults that had already been responsible for previously reported failures. By excluding these 133 anomalies we derived at a set of 520 anomalies, each of which represents a unique fault in the flight software of one of the 18 missions. Note that there are anomalies (especially those which cannot be reproduced) for which the cause is not fully understood. Such an anomaly was categorized as a duplicate and removed from the following analysis if the similarity of its description with an earlier anomaly suggested that both had been caused by the same fault.

Following the scheme presented in Section 1, we categorized these software faults into Bohrbugs, non-aging-related Mandelbugs, and aging-related bugs. We based our decisions on the textual descriptions in the anomaly reports, as well as discussions with appropriate development personnel in cases for which a classification could not initially be made. For those faults for which it was not possible to obtain sufficient information for determining the classification, we introduced the additional fault type "unknown" (UNK).

4. Analysis

4.1. Joint analysis of all software faults

Of the 520 software faults identified for all 18 missions, we classified 319 as Bohrbugs, 167 as non-aging-related Mandelbugs and 23 as aging-related bugs. For 11 faults the type could not be determined; these faults were therefore assigned the fault type "unknown". The corresponding proportions of the fault types BOH, NAM, ARB and UNK are thus 0.614, 0.321, 0.044 and 0.021, respectively.

As pointed out above, our fault type definitions focus on the *potential* manifestation characteristics of faults, not on the fault behavior with respect to one *specific* failure occurrence. The reasoning behind this decision was that the fault type should depend on inherent fault characteristics. If, for example, a fault was considered a Mandelbug if a specific failure it caused cannot be reproduced, then the classification would highly depend on the knowledge of the specific user or tester who encountered this failure. The analysis of faults in JPL/NASA flight software has confirmed our approach: Since many system parameters are constantly logged, JPL/NASA engineers are able to eventually reproduce a large percentage of failures that would be difficult to reproduce in "normal" industrial settings. Despite this fact, based on our definitions the underlying faults are always categorized as Mandelbugs if any of the criteria of "complexity" applies to the fault activation or the error propagation mechanism – whether or not the related failure can be reproduced.

We also mentioned that aging-related bugs are faults that can *potentially* cause software aging, i.e., an increasing failure rate and/or performance degradation. Even if such a fault should be detected (for example, during a code inspection) before it has had the chance to actually lead to a decreasing performance, its fault type remains unchanged. However, in this case the fact that the fault could have caused aging may go unnoticed. Therefore, the percentage of faults categorized as aging-related bugs based on fault descriptions and anomaly reports tends to be a lower bound for the true fraction of faults that are aging-related bugs.

Based on the software faults for all 18 missions, we investigated whether there are any dependencies between the fault type and three other criteria according to which anomalies are classified in JPL's Problem Reporting System: the "criticality," the "failure effect," and the "failure risk."

The variable "criticality" distinguishes between anomalies with "unacceptable risk", "accepted risk", "no significant risk", and "no risk". To test the null hypothesis "fault type and criticality are independent" against the alternative hypothesis that they *do* depend, we employed a chi-square independence test. This test compares the contingency table, containing the absolute number of joint observations of each failure type and each criticality, with the independence table, consisting of the expected number of joint observations if the two variables were indeed independent. To evaluate the disagreement between the two tables, for each combination of fault type and criticality category the squared deviation between the observed and the expected absolute frequency is computed and divided by the expectation. The value of the test statistic is the sum of all of these normalized squared deviations. It holds true in general that under the null hypothesis of independence the test statistic follows a chi-square distribution with $(i–1)(j–1)$ degrees of freedom, where i and j represent the number of categories for the first and the second variable, respectively [17], pp. 447–449. In our example, the value of the test statistic amounts to 6.41. Since the corresponding p-value is 0.379, the hypothesis that fault type and criticality are independent cannot be rejected at any reasonable error level, like 5% or 1%. This means that our data do not contain any evidence for an association between the type of a fault and its criticality.

In JPL's Problem Reporting System, "failure effect" assesses the effect of the problem/failure as if it had occurred in flight without the benefit of corrective action or redundancy. The ratings are "negligible effect", "significant effect", and "major or catastrophic effect". Comparing the contingency table with the independence table (both not shown in this paper) indicates that Bohrbugs with "negligible" failure effects are slightly underrepresented, while those with "major or catastrophic" failure effects are overrepresented; the opposite seems to be true for non-aging-related Mandelbugs. However, the decision of testing the null hypothesis "fault type and failure effect are independent" versus the alternative that fault type and failure effect are indeed associated depends on the error level (i.e., the probability for wrongfully rejecting the null hypothesis if it is true) that we are willing to accept: The value of the chi-square statistic is 9.58, which corresponds to a p-value of 0.048. While the independence hypothesis could be rejected at an error level of 5%, this is not the case at an error level of 1%. There is thus some indication that fault type and failure effect might be linked.

"Failure risk" is an assessment of the certainty that the exact failure cause has been determined and that the corrective action will eliminate any known possibility of recurrence of the problem in flight. The ratings are "known cause/certainty in corrective action", "unknown cause/certainty in corrective action", "known cause/uncertainty in corrective action", and "unknown cause/uncertainty in corrective action". In our data, failures caused by Bohrbugs for which the cause is known and the corrective action is certain are clearly overrepresented; likewise, among failures caused by non-aging-related Mandelbugs, those for which either the cause is unknown or the corrective action is uncertain are highly overrepresented. The statistical significance of this relationship can be shown via a chi-square independence test of the null hypothesis "fault type and failure risk are independent" against the alternative hypothesis that they *are* linked. The value of the chi-square statistic obtained is 64.03, implying a p-value of 6.8e-12. The null hypothesis can thus be rejected at any common error level. While this result might have been expected due to the fault type definitions, it is surprising that the dependence is that significant: As pointed out earlier, JPL/NASA engineers are often able to reproduce failure caused by Mandelbugs due to the wealth of system parameters recorded. Although the logged information should also help them to identify the causes of such failures, this does not seem to be the case in general.

4.2. Fault type proportions vs. launch order

While space missions vary in their duration, system size, and fault density of the software, the fault type proportions might be similar across missions. We investigate this conjecture in this section. 44 of the unique faults extracted from JPL's Problem Reporting System were discovered after deployment of the space system from the developing facility, but before launch. Since the types of faults causing failures during normal operation could differ from the kinds of faults discovered in earlier phases, we drop these 44 faults for the following analyses, focusing on the unique faults detected after launch.

Moreover, for 10 of the 18 missions considered so far the number of unique software faults discovered is very small: The *combined* number of unique faults detected for these 10 missions amounts to 39, averaging to a mere 3.9 faults per mission. Clearly, for these missions the fault type proportions are unlikely to have stabilized. Our examination of fault type proportions within missions is therefore based on the remaining eight missions. These missions are summarized in Table I. The smallest number of unique faults available for any of these missions is 23; the total number of unique faults that our analysis is based on is 437.

For each mission the first column provides a unique numerical identifier corresponding to launch order; i.e., mission ID 1 was the first mission launched. The second column represents the operational mission duration covered in our data. For historic missions, this duration is the period between launch and the end-of-mission; for current missions, it is the period between launch and the date on which the information was gathered from JPL's Problem Reporting System. Showing the durations in days would provide clues helping to identify the missions. In order to maintain mission anonymity, all durations have therefore been normalized in terms of the duration of the longest-running mission (mission ID 1). Of course this normalization does not change the relative length of missions: For example, a mission that lasted half as many days as mission ID 1 would be assigned a duration of 0.5. The remaining four columns represent the proportions of Bohrbugs, non-aging-related Mandelbugs, aging-related bugs, and faults of unknown type detected in the flight software of the respective mission after launch.

The last two rows of Table I list the (unweighted) average fault type proportions, as well as the standard deviations of the individual fault type proportions observed. Parts of the variation might be explained by the time at which the space systems were developed, launched and operated. Figure 1, in which the missions are arranged in launch order, shows how the fault type proportions changed across missions.

Table I. Mission IDs, normalized durations and fault type proportions for the eight missions with the largest number of unique faults

ID	Normalized duration	Fault type proportions			
		BOH	NAM	ARB	UNK
1	1.000	0.595	0.270	0.135	0.000
2	0.911	0.571	0.379	0.043	0.007
3	0.657	0.481	0.481	0.000	0.037
4	0.582	0.554	0.369	0.062	0.015
5	0.292	0.810	0.143	0.048	0.000
6	0.376	0.522	0.435	0.000	0.043
7	0.226	0.815	0.130	0.019	0.037
8	0.171	0.643	0.343	0.014	0.000
Avg. proportions		0.635	0.307	0.038	0.020
Standard deviations		0.114	0.112	0.042	0.017

Obviously, for three of the four most recent missions (IDs 5, 7, and 8) the proportion of Bohrbugs is larger than for any of the first four missions. Besides the corresponding decrease in the proportion of all Mandelbugs, we can also see that especially the proportion of aging-related bugs seems to be lower for the later missions. These findings could suggest the following conclusions, which are not mutually exclusive:

- In more recent development projects, the proportion of residual software faults that are Bohrbugs is larger than for earlier missions. There are two possible explanations:
  - The development process for recent missions has changed so that a higher proportion of the faults created are Bohrbugs.
  - Alternatively, for more recent missions, fault detection and removal techniques have become more effective at reducing the number of Mandelbugs remaining in the system at launch time.
- For the more recent space missions, the operational environment is better controlled. Non-aging-related Mandelbugs and aging-related bugs for which a failure occurrence requires specific and unusual environmental conditions are now underrepresented among the detected faults.

One might also think that the more recent missions (but not the older ones) employ techniques like software replication or software rejuvenation, masking existing non-aging-related Mandelbugs and aging-related bugs. However, within the missions analyzed there appear to be no significant differences in the fault-tolerance strategies used. Such masking effects are thus unlikely to be the cause of our findings.

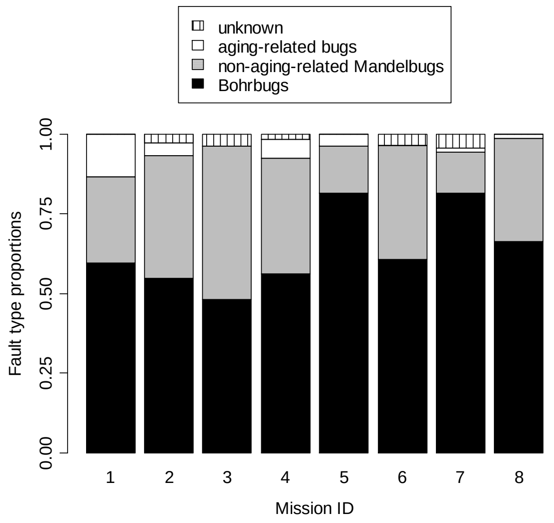

Figure 1. Fault type proportions for the eight missions with the largest number of unique faults

There is yet another possibility: The findings might be an artifact of the analysis. As they were launched later, the duration covered by the data tends to be shorter for the more recent missions as opposed to the earlier missions, some of which are still ongoing. Since one may conjecture that the proportion of failures caused by aging-related bugs increases with mission duration (after all, it is the nature of these faults to manifest themselves at a higher rate after longer periods of continuous operation), the observed differences in the fault type proportions could simply be driven by the different mission runtimes.

4.3. Fault type proportions vs. mission runtimes

To investigate the conjecture formulated at the end of the last section, we now study how the fault type proportions evolve within missions over their runtimes.

Figure 2 compares the development of the proportion of Bohrbugs for the four earlier missions. All runtimes are again normalized by the total duration of mission ID 1; for this mission, information on the fault type proportions is therefore available up to a normalized runtime of 1.0 (or 100%). For each mission, the proportion of Bohrbugs is updated whenever a new fault is discovered in the flight software of that mission. For example, after a runtime in days that corresponds to around 19 percent of the total duration of mission ID 1, the proportion of Bohrbugs for that mission drops first to 83% and a very short time later to 71%. After the same number of days since launch, the proportion of Bohrbugs among the faults detected in the software of mission ID 3 is 39%. The number of days for which this mission is covered in our data is about 66% of the duration of mission ID 1; therefore, the line related to mission ID 3 breaks off at an x-coordinate of 0.66.

Figure 2 does not indicate that within one mission the proportion of Bohrbugs usually decreases monotonically over the mission runtime. However, it does show that as a mission progresses and more and more faults are discovered there tends to be less and less fluctuation in the proportion of Bohrbugs within a mission. What is more, across missions the proportions seem to stabilize around almost the same value after a similar number of days since launch. After a mission runtime that corresponds to 40 percent of the duration of mission ID 1, for all missions the proportion of Bohrbugs lies in the interval (35%, 65%); after a normalized runtime of 0.55, the lower bound of this interval has further risen to 45%. At a normalized runtime of 0.75, the proportion of Bohrbugs has stabilized at a value around 58% for the two missions for which data are still available.

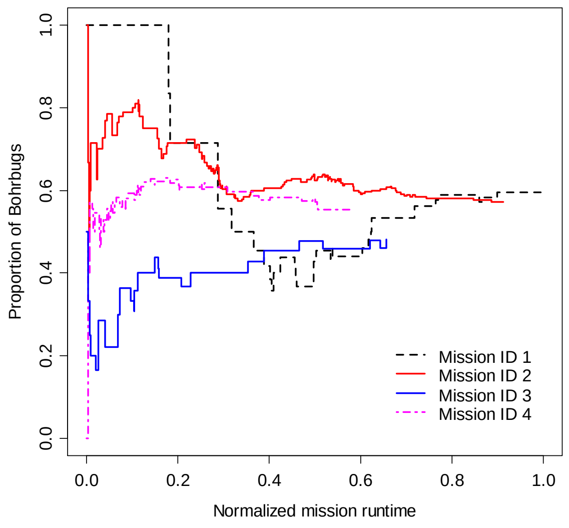

Figure 2. Proportion of Bohrbugs for missions ID 1 to 4

These findings suggest that for the earlier four of our eight missions, the fraction of Bohrbugs among all faults contained in the software at launch time may indeed have been very similar.

It is not surprising that this might be the case: The types of systems developed were similar to each other in terms of complexity and functionality. Although a significant effort was undertaken to improve software process maturity over the years, the development processes themselves as defined by the project software management plans were similar to each other, and the institutional standards governing software development did not change significantly. Moreover, many of the developers worked on more than one of these projects.

The variation in the proportion of Bohrbugs among the detected faults across these missions (see Table I) could thus be explained by the differences in the mission runtimes covered by the data. To check this surmise, we extract information on how the proportion of Bohrbugs develops over time for the earlier four projects in the form of 95% confidence intervals; these intervals can then be compared with the available data for the four later missions, to study whether or not they are substantially different from the earlier ones.

For each of 100 evenly-spaced normalized mission runtimes t_i (i = 1, …, 100) between zero and one, we collect B_i, the set of current Bohrbug proportions for those missions (among missions ID 1 to 4) whose normalized duration is greater than or equal to t_i. We denote the cardinality of the set B_i by k_i. (For example, for the normalized runtime $t_i = 0.4$, observations are still available for all $k_i = 4$ missions, with the current set of Bohrbug proportions B_i = {0.417, 0.455, 0.576, 0.605}; after a normalized runtime of $t_i = 0.6$, only $k_i = 3$ missions provide information about the current proportion of Bohrbugs, and B_i = {0.440, 0.458, 0.595}.) The bounds of a 95% confidence interval for the proportion of Bohrbugs at t_i can only be determined if k_i is larger than one. To this end, a bootstrapping-type algorithm is then employed.

Bootstrapping is a data-based simulation approach accounting for the fact that in inferential statistics the k observations available are a random sample from a (possibly infinite) sample space [5]. To evaluate variability, bootstrapping produces many samples of size k, each time drawing from the k observations without replacement, and calculates the statistics of interest (e.g., the sample mean) for all of these samples.

Since the number of observations in B_i is always small, our bootstrapping-type algorithm does not have to rely on sampling. Instead, it directly creates all possible combinations of k_i values chosen from B_i with replacement. Then a distribution is fitted to each of these $n_i = k_i^{k_i}$ combinations C_{ij} ($j = 1, \ldots, n_i$):

- If all values in C_{ij} are identical, then the fitted distribution is a singular distribution at this value.
- Otherwise, a beta distribution is fitted to the values in C_{ij}. Due to its flexible shape and the fact that its domain is restricted to the interval from zero and one, the beta distribution is often used to model fractions. However, since the end-points of the interval are not included in the domain, values of zero and one contained in C_{ij} (which may especially be observed for short mission runtimes) are replaced by values slightly larger than zero and less than one, respectively.

To derive the 95% confidence interval for the Bohrbug proportion at normalized mission runtime t_i, we then sample 2000 values from the mixed distribution in which each of the n_i fitted distributions is given weight $1/n_i$. The lower (upper) bound of the confidence interval is the lower (upper) empirical 2.5% quantile of the 2000 values sampled.

We implemented this bootstrapping-type approach for analyzing the data in the programming language provided by the statistics software R [18]. Figure 3 and Figure 4 show the 95% confidence interval for the proportion of Bohrbugs as dotted lines. The growing stabilization with mission runtime – already observed in Figure 2 – is reflected in the decreasing width of the confidence interval (i.e., the difference between its upper and lower bounds tends to get smaller). For normalized runtimes larger than 0.582, parts of this decline can be attributed to the fact that the calculations are based on a smaller number of missions. However, even for shorter mission runtimes the reduction in variability is clearly noticeable.

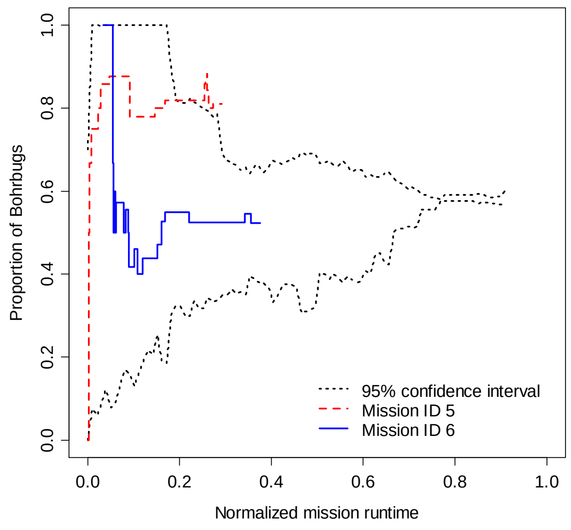

Figure 3. Proportion of Bohrbugs for missions ID 5 and 6, and 95% confidence interval based on missions ID 1 to 4

Figure 3 and Figure 4 also compare the confidence interval established for the four earlier missions with the development of the Bohrbug proportions for the four later ones. Table I and Figure 1 showed that for three of the later missions the Bohrbug proportion is higher than for the earlier missions. We now see from Figure 3 and Figure 4 that for mission ID 8 the difference in the Bohrbug proportion could be explained by the fact that at short mission runtimes the fault type proportions have not yet stabilized as much as after longer runtimes. However, for missions ID 5 and 7 several observations lie outside the 95% confidence interval, suggesting that there may indeed be a difference between the first four and these later missions.

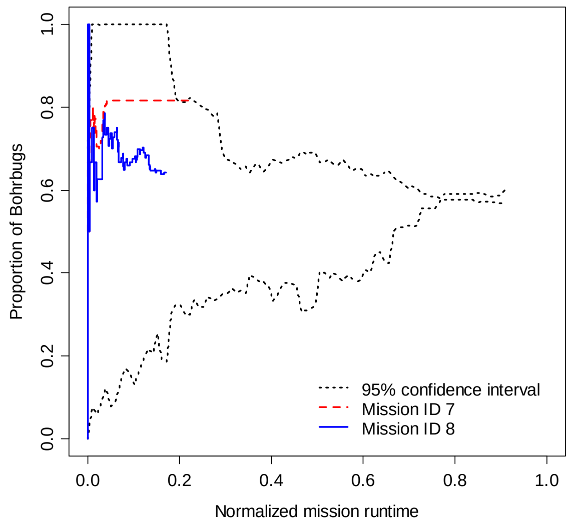

Figure 4. Proportion of Bohrbugs for missions ID 7 and 8, and 95% confidence interval based on missions ID 1 to 4

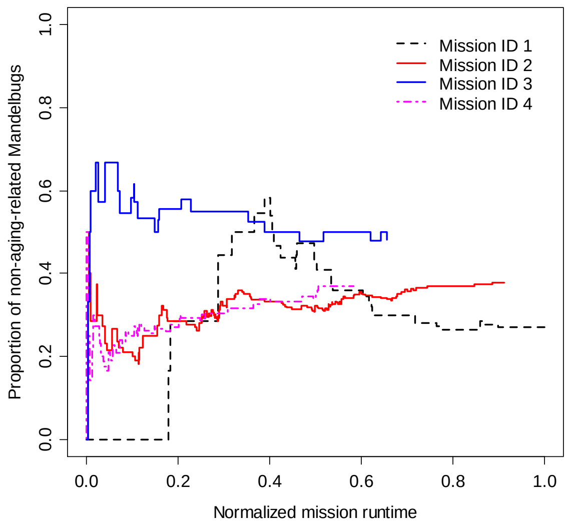

Figure 5. Proportion of non-aging-related Mandelbugs for missions ID 1 to 4

Since each software fault is either a Bohrbug or a Mandelbug, the results presented for Bohrbugs analogously apply to the category of all Mandelbugs. Specifically, the evidence suggests that the fraction of initial faults that are Mandelbugs may be smaller for missions ID 5 and 7 than for the first four missions analyzed.

However, this does not necessarily mean that these findings also apply to the two sub-types of Mandelbugs. The same analyses were therefore also carried out for the proportions of non-aging-related Mandelbugs. The results are shown in Figure 5 to Figure 7.

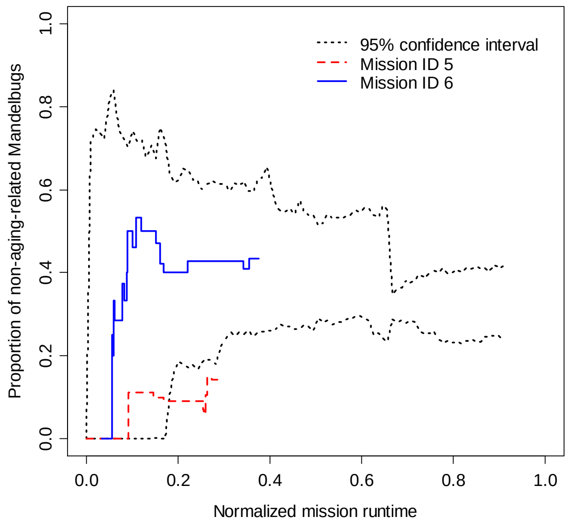

Figure 6. Proportion of non-aging-related Mandelbugs for missions ID 5 and 6, and 95% confidence interval based on missions ID 1 to 4

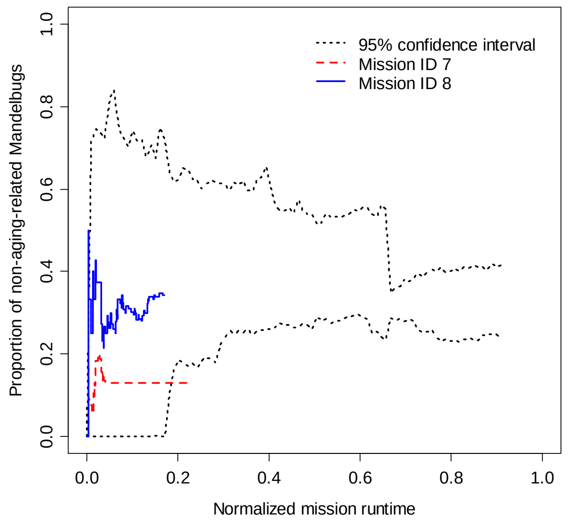

Figure 7. Proportion of non-aging-related Mandelbugs for missions ID 7 and 8, and 95% confidence interval based on missions ID 1 to 4

The data depicted in Figure 5 indicate that for the four earlier missions the variability in the proportion of non-aging-related Mandelbugs tends to get less with increasing mission duration, although it is not clear whether the proportions will in the end settle to very similar values. Corresponding with this behavior is again a decreasing trend in the widths of the 95% confidence interval derived with the bootstrapping-type algorithm, as can be seen in Figure 6 and Figure 7.

Again, it is missions ID 5 and 7 for which some of the proportions of non-aging-related Mandelbugs lie outside the 95% confidence interval derived for the earlier missions. This could mean that for these missions the fraction of non-aging-related Mandelbugs among all initial software faults is lower than for the first four missions.

Of course, different initial fault type proportions are not the only possible explanation for the higher (lower) proportions of Bohrbugs (non-aging-related Mandelbugs) observed for missions ID 5 and 7. Instead, these initial proportions could be similar, but the stabilization might occur later than for the earlier missions. This might for example happen if the intended mission duration is longer for the later missions, because non-aging-related Mandelbugs can be expected to show themselves at a higher rate once operational constraints are relaxed after a mission's primary goals have been achieved.

We finally analyzed the proportions of aging-related bugs; however, additional figures are omitted due to space constraints. The proportions of aging-related bugs initially observed during the four earlier missions showed less overall variability than the proportions of the other fault types. Moreover, the proportions stabilize to a lesser extent as mission runtime increases; no clear trend is discernible for the widths of the 95% confidence intervals determined with the bootstrapping-type algorithm. Comparing the proportions of aging-related bugs for the four later missions with the 95% confidence intervals based on the four earlier missions, we could not find any clear differences.

5. Discussion

We have analyzed software anomaly reports for 18 JPL robotic exploration space missions to study the dependencies between the fault type and further characteristics like failure effect, and failure risk; the proportions of Bohrbugs, non-aging-related Mandelbugs, and aging-related bugs among all software faults detected; changes in these proportions in the course of a mission; and variations in the fault type proportions across missions. Our findings are as follows:

1. Although the wealth of system parameters logged enables engineers at JPL/NASA to reproduce many of the failures caused by Mandelbugs, there is a highly significant relationship between the fault type and the failure risk: For failures due to Mandelbugs the cause is often questionable or not understood. However, at an error level of 1%, the hypotheses that fault type and criticality as well as fault type and failure effect are independent could not be rejected.
2. Among the 520 software faults detected in all 18 missions after deployment of the space system from the developing facility, 61.4% were Bohrbugs, and 36.5% were Mandelbugs; this latter number includes those 4.4% of all software faults that were aging-related bugs. While these proportions highlight the importance of Mandelbugs, they also seem to indicate that additional testing and technical review prior to deployment of the space system from the developing facility could further reduce the number of Bohrbugs found later. Detailed analyses of testing effort and defect repair times will be required to develop more effective strategies.
3. The widths of the confidence intervals for the proportions of Bohrbugs and non-aging-related Mandelbugs calculated based on four early missions showed decreasing trends. This suggests that after long mission durations the proportions of Bohrbugs/non-aging-related Mandelbugs among the detected faults are similar across missions. A possible explanation is that at launch time, after completion of the testing phase, the proportion of Bohrbugs among the residual flight software faults is similar for these four missions; the same applies to the initial proportion of non-aging-related Mandelbugs.
4. The decreasing widths of the confidence intervals also imply that much of the variation in the fault type proportions of the four early missions seen in Table I can be explained by the fact that for short-running missions with a low absolute number of faults detected the fault type proportions have not yet stabilized. This may also be the case for the four more recent missions analyzed, although there is some evidence that the flight software of earlier missions contained a smaller proportion of Bohrbugs and a higher proportion of Mandelbugs.

These findings will be able to provide guidance in the fault detection, identification, and recovery (FDIR) techniques implemented in space mission systems, as well as guidance in the verification strategies to be used during development. For example, since Mandelbugs are difficult to detect and remove during software testing, the rather large proportion of Mandelbugs among the residual faults at launch time indicates the potential benefit of employing verification techniques such as model checking and theorem proving in addition to dynamic testing. A significant proportion of the Mandelbugs we found are related to the effects of instruction ordering in multi-threaded systems (e.g., race conditions, deadlocks). Techniques such as model checking were developed to find these types of defects; for instance, the SPIN model checker [14] was developed specifically to find timing-related faults in telephony protocols. These faults can be very difficult to find by testing because (i) testers will usually not be able to control the order in which instructions are executed for the system under test, and (ii) the computational state space is almost always too large to test all of the possible execution orderings, even if the tester did have sufficiently detailed control.

Our analyses are based on those anomalies classified as related to flight software according to the “Cause” field. Recent work by Green et al. [9] as part of the UR Program indicates that in the JPL Problem Reporting System the number of software-related anomalies may be significantly undercounted. Green et al. analyzed in detail approximately 1300 anomalies of all types between launch and orbit insertion for seven Mars missions. These anomalies were categorized according to type (e.g., flight software, flight hardware) not by examination of the “Cause” field of the anomaly report, but by a detailed reading of the text describing the anomalous behavior, the analysis conducted to determine the cause, and the final corrective action. Because many of the anomaly reports that were analyzed for Green’s UR task form a subset of the anomaly reports that we analyzed, any of Green’s results that may affect the validity of our work should be considered.

Of particular interest is the finding by Green et al. that the number of flight software anomalies identified by reading the descriptive text of the anomaly reports was approximately six times the number of flight software anomalies identified by the anomaly reports' "Cause" field. This indicates that the number of software anomalies may be significantly underreported in the JPL Problem Reporting System.

We studied additional anomalies that were not part of the original analysis to determine the effect of this finding on our results. From the Mars missions analyzed by Green et al. [9], we identified 62 flight software anomalies that had been labelled as other types of anomaly (e.g., flight hardware, procedural) in the "Cause" field and classified them according to our criteria. The proportions of the types BOH, NAM, ARB and UNK for these additional anomalies were 0.694, 0.258, 0.016, and 0.032. By way of comparison, the global proportions of the fault types BOH, NAM, ARB and UNK reported at the beginning of Section 4.1 were 0.614, 0.321, 0.044 and 0.021. Although for the additional anomalies the proportion of Bohrbugs was higher and the proportion of Mandelbugs was somewhat lower, the proportions are similar. The somewhat higher proportion of Bohrbugs and lower proportion of Mandelbugs may be explained by the fact that all of the additional anomalies discussed by Green et al. [9] were observed during the cruise phase between launch and planetary orbit insertion, a period during which less activity takes place and which is better controlled and understood. This could allow fewer opportunities for conditions enabling Mandelbugs to manifest themselves to arise. Of course, the relatively small sample size prevents this additional analysis from being conclusive. It does, however, affirm the viability of the hypothesis that a future analysis of additional "true" flight software anomalies will not substantially change the results reported in this paper.

6. Future work

In our ongoing project, we are continuing the analysis of the anomaly reports collected from the JPL Problem Reporting System. We are currently analyzing the anomaly reports for ground-based mission support software systems.

Our goal is to use the project results as the basis for recommending more effective fault identification, removal, and mitigation techniques for future robotic space mission software. The findings reported in this paper are a first step, since they provide us with evidence for the relative importance of the different fault types for space mission system software.

Acknowledgments

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and 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.

References

[1] E. N. Adams, Optimizing preventive service of software products, *IBM Journal of Res. & Development* 28(1):2–14, 1984.

[2] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing, *IEEE Transactions on Dependable and Secure Computing* 1(1):11–33, 2004.

[3] G. Candea, J. Cutler, and A. Fox, Improving availability with recursive microreboots: a soft-state system case study, *Performance Evaluation* 56(1-4):213–248, 2004.

[4] R. Chillarege. Orthogonal Defect Classification, in M. R. Lyu, editor, *Handbook of Software Reliability Engineering*. McGraw-Hill, New York, 1995.

[5] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*. Chapman & Hall, New York, 1993.

[6] J. Gray, Why do computers stop and what can be done about it? in *Proc. Fifth Symposium on Reliability in Distributed Systems*, 1986, pp. 3–12.

[7] J. Gray and D. P. Siewiorek, High-availability computer systems, *IEEE Computer* 24(9):39–48, 1991.

[8] N. W. Green, A. R. Hoffman, and H. B. Garrett, Anomaly trends for long-life robotic spacecraft, *Journal of Spacecraft and Rockets* 43(1):218–224, 2006.

[9] N. W. Green, A. R. Hoffman, T. K. M. Schow, and H. B. Garrett, Anomaly trends for robotic missions to Mars: Implications for mission reliability, AIAA 2006-269, in *Proc. 44th AIAA Aerospace Sciences Meeting and Exhibit*, 2006, pp. 1–9.

[10] M. Grottke and K. S. Trivedi, Software faults, software aging and software rejuvenation, *Journal of the Reliability Engineering Association of Japan* 27(7):425–438, 2005.

[11] M. Grottke and K. S. Trivedi, A classification of software faults, in *Supplemental Proc. Sixteenth International Symposium on Software Reliability Engineering*, 2005, pp. 4.19-4.20.

[12] M. Grottke and K. S. Trivedi, Fighting bugs: Remove, retry, replicate, and rejuvenate, *IEEE Computer* 40(2): 107–109, 2007.

[13] A. R. Hoffman, N. W. Green, and H. B. Garrett, Assessment of in-flight anomalies of long life outer planet missions, in *Proc. European Space Agency 5th International Symposium on Environmental Testing for Space Programmes*, 2004, pp. 43–50.

[14] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*, Addison-Wesley, Boston, 2003.

[15] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton, Software rejuvenation: Analysis, module and applications, in *Proc. Twenty-fifth International Symposium on Fault-Tolerant Computing*, 1995, pp. 381–390.

[16] I. Lee and R. K. Iyer, Software dependability in the Tandem GUARDIAN system, *IEEE Trans. Software Engineering* 21(5):455–467, 1995.

[17] P. Newbold, *Statistics for Business and Economics*. Third edition. Prentice-Hall, Englewood Cliffs, 1991.

[18] R Development Core Team. *R: A Language and Environment for Statistical Computing*. Reference index, v. 2.10.1, R Foundation for Statistical Computing, Vienna, Austria, 2009.

[19] E. S. Raymond, *The New Hacker's Dictionary*. MIT Press, Cambridge, 1991.