The nature of the times to flight software failure during space missions

Javier Alonso
Duke University
javier.alonso@duke.edu

Michael Grottke
Friedrich-Alexander-Universität
Erlangen-Nürnberg
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
ktrivedi@duke.edu

***Abstract*—The growing complexity of mission-critical space mission software makes it prone to suffer failures during operations. The success of space missions depends on the ability of the systems to deal with software failures, or to avoid them in the first place. In order to develop more effective mitigation techniques, it is necessary to understand the nature of the failures and the underlying software faults.**

Based on their characteristics, software faults can be classified into Bohrbugs, non-aging-related Mandelbugs, and aging-related bugs. Each type of fault requires different kinds of mitigation techniques. While Bohrbugs are usually easy to fix during development or testing, this is not the case for non-aging-related Mandelbugs and aging-related bugs due to their inherent complexity. Systems need mechanisms like software restart, software replication or software rejuvenation to deal with failures caused by these faults during the operational phase.

In a previous study, we classified space mission flight software faults into the three above-mentioned categories based on problems reported during operations. That study concentrated on the percentages of the faults of each type and the variation of these percentages within and across different missions. This paper extends that work by exploring the nature of the times to software failure due to Bohrbugs and non-aging-related Mandelbugs for eight JPL/NASA missions. We start by applying trend tests to the times to failure to check if there is any reliability growth (or decay) for each type of failure. For those times to failure sequences with no trend, we fit distributions to the data sets and carry out goodness-of-fit tests. The results will be used to guide the development of improved operational failure mitigation techniques, thereby increasing the reliability of space mission software.

I. Introduction

Spacecraft systems make increasing use of better and more powerful hardware, with reduced mass and energy consumption. This hardware improvement allows deploying more sophisticated flight software, which in turn has enabled the design of more ambitious missions. This naturally leads to the the development of more complex software with new features to monitor and control an increasing number of hardware devices that are needed for the success of such space missions [1]. However, the growing complexity of flight software makes it more prone to suffer failures. The ability of the spacecraft systems to deal with software failures during operation can determine the success or failure of a mission.

The growing requirements of space missions also necessitate flight software with the capability to run autonomously, or at least with minimum human intervention. Planetary spacecraft need to make decisions autonomously because there is no time to wait for the round-trip message delay. Moreover, spacecraft can be isolated from the earth for hours, or even days, and the volume of data collected has to be preprocessed on-board to reduce the bandwidth to transfer the data to the earth. For this reason, autonomous fault protection mechanisms have been introduced into flight software.

To develop better mitigation techniques for dealing with software failures during operation, and hence to develop more reliable space mission software, it is necessary to have a deeper understanding of the nature of the software failures, and the underlying software faults.

Software faults, or "bugs", are the underlying causes of software failures. Based on their characteristics, they can be classified into Bohrbugs, non-aging-related Mandelbugs, and aging-related bugs. In our previous papers we have attempted to define these terms as precisely as possible [2], [3], [4], [5].

The term **Bohrbug** (BOH) was coined by Gray [6] in 1985. It refers to a fault that is easy to isolate and whose manifestation is consistent under a well-defined set of conditions, because its activation and error propagation lack "complexity" as defined below.

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. In our definition of a Mandelbug we trace these characteristics to the complexity of its activation and/or error propagation. This 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.

To appear in Proc. 23rd IEEE International Symposium on Software Reliability Engineering, 2012.

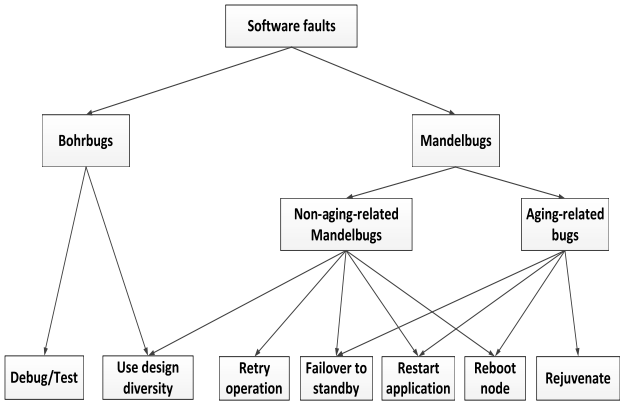

Fig. 1. Software fault/failure mitigation classification tree

Mandelbugs are intrinsically related to software complexity: The more complex a piece of software, the higher the risk of its containing a large number of Mandelbugs. In fact, there are similarities between our Mandelbug definition and the definition, due to Dörner, of system complexity as "the label we give to the existence of many interdependent variables in a given system. The more variables and the greater their interdependence, the greater that system's complexity. (...) The links between the variables oblige us to attend to a great many features simultaneously, and that, concomitantly, makes it impossible for us to undertake only one action in a complex system. (...) A system of variables is 'interrelated' if an action that affects or is meant to affect one part of the system will also affect other parts of it. Interrelatedness guarantees that an action aimed at one variable will have side effects and long-term repercussions." [7, p. 38]

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.

The upper portion of Figure 1, adapted from [14], summarizes the relationship between these types of faults.

The importance of classifying faults is not only theoretical. It has also a practical importance, because every fault type requires different approaches to deal with it in the development and testing phase, as well as during operation [4], [14].

Typically, Bohrbugs may easily be isolated and removed during testing. Design diversity 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 [15]. 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 lower part of Figure 1 depicts the different mitigation approaches for each fault type.

Due to the practical importance of the fault type classification, in our previous paper [5], we analyzed the problem reports from 18 JPL/NASA space missions, and classified flight software faults into the three above-mentioned categories. That study concentrated on the percentages of the unique faults of each type. The overall percentages of Bohrbugs, non-aging-related Mandelbugs, and aging-related bugs were 61.4%, 32.1%, and 4.4%, respectively; 2.2% of the faults could not be classified. We also examined the relationship between the fault types and the failure severity or effects. Moreover, for the eight missions with a substantial number of flight software faults detected, we analyzed the variation of the fault type percentages within and across different missions.

In this paper, we analyze the problem reports from the same eight JPL/NASA space missions, but focusing on the time intervals between failure occurrences, also known as times to failure (TTFs). Based on the previous failure report classification, we conduct two different and complementary analyses in order to better understand the nature of the times to failure. First, we examine the existence of a trend in the times to failure per mission and type of fault. This analysis indicates if the reliability of the flight software tends to increase or decrease as more and more failures have occurred. Second, for those times to failure sequences with no trend detected we try to determine appropriate models for the underlying distributions, assuming that the values observed are independently and identically distributed. This study, and the previous one, will help to define guidelines for developing more effective fault tolerance and failure mitigation techniques during operations.

The rest of the paper is organized as follows: Section II reviews related work analyzing the nature of the times to (software and hardware) failure. Section III describes the classification process conducted to classify the problem reports according to the type of fault which caused the failure. Section IV analyzes the times to failure sequences in order to determine if there is any trend. For those data sets with no trend, we fit distributions, carry out goodness-of-fit tests, and determine the best distributional model in Section V. Section VI discusses the results obtained, and concludes the paper.

II. Related work

Understanding system failures becomes critical to develop more reliable systems. Although only relatively few empirical failure data sets are available, there are several papers addressing the important task of understanding the nature of failures from different perspectives.

Several papers have focused on analyzing the causes of system failures [6], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27]. In [6], [16], [20], [21], [23], the main sources of failures considered were human mistakes, software faults, and hardware faults. These studies concluded that hardware represents 10–30% of the sources of failures, human mistakes represent 30–50%, and software represents the main cause in 20–50% of the failures, depending on the system and study. Other sources, such as environmental or natural disasters, were also considered, but their impact on failures was very small in comparison with the above-mentioned ones.

Hard disk failures and their causes have received special attention [18], [26], [27]. These studies were focused on correlating different parameters monitored from the hard disks and other storage subcomponents with the failures. The primary goal was to understand the main failure causes.

A few studies related to space mission failures have addressed the classification of the source of the failures [22], [24], [25]. Theses studies highlighted that the main sources of failures in the space missions analyzed were the flight and ground software. However, they went no further in analyzing the fault types in the case of software failures. In our previous work, we tried to fill in this gap, classifying the software failures from 18 JPL/NASA missions according to the type of the underlying software fault [5]. The results showed that Bohrbugs were the main cause of software failures, followed by non-aging-related Mandelbugs. A small subset of failures were due to aging-related bugs.

Other papers have been focused on understanding the nature of the distribution of TTFs [28], [29], [30], [31], [32], [27]. All these studies were focused on fitting distributional models to the TTFs, and determining the most suitable one. They concluded that the TTFs could be modeled with gamma or Weibull distributions. However, all studies examined the complete failure data of the respective systems (or subcomponents), without separately analyzing the failures from different sources of failures. Furthermore, the previous analyses assumed that all the TTFs were independent and identically distributed. To the best of our knowledge, there is no existent work that carries out both a reliability growth analysis and a distributional analysis for separate fault types in multiple systems. This indicates the relevance of the current paper in order to obtain a better understanding of the nature of the TTFs in space missions.

III. Failure classification process

The problem reports from the eight JPL/NASA missions under analysis were filled in 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 correct or mitigate the failure and its consequences. Based on this problem information typed by the operators, the failure reports were classified according to the underlying fault which caused the failure.

This section describes how this classification was conducted in order to show how to connect the aforementioned fault definitions with real failure reports. Table I presents five failure reports to exemplify the failure report classification conducted. The failure reports have been sanitized to guarantee the anonymity of the data. However, the data relevant for classifying the underlying fault have been preserved.

The first failure example is caused by a Bohrbug. The cause of the failure was easily found, isolated, and defined by the operators. Due to this, the corrective action was based on an updated version of the flight software.

The second example clearly describes a failure caused by a non-aging-related Mandelbug. The operators detected the failure, which was related to interactions of concurrently-running tasks. The repair developed was not employed, however, the justification being that it was too late in the mission for the repair to be useful and that the failure was an uncommon occurrence. The final action was to use the software as is.

The third example, another example of a non-aging-related Mandelbug, also presents side effects due to the interaction of different operations. In this case, the failure depends on the timing of the operations. The developers were able to devise a repair, and applied it to the on-board software.

The last two failures were caused by aging-related bugs. In both cases the failure rate was increasing with time. However, the corrective action applied was different. In the first case, the engineers were able to find, isolate, and fix the fault; via testing they confirmed the effectiveness of the patch. In the second case, they applied a kind of software rejuvenation, namely, a periodic proactive clean-up of the stack to avoid the consequences of the fault. The aging-related bug was not fixed, but its consequences were avoided.

The classification of the fault type causing each failure was based on these types of descriptions. The classification was conducted in parallel by two persons. After that, the classifications were compared. In the few cases of a mismatch a detailed analysis was conducted by the two persons to achieve a consensus.

Based on the above classification, we classified those 481 failure reports of the eight JPL/NASA missions related to flight software as shown in Table II. Note that in our earlier study [5] we had removed repeated failure occurrences caused by the same fault ("duplicates"), because we wished to study unique software faults. In this work, focusing on the TTFs, such failures are included. A total of six failures could not be classified; they were therefore assigned to the "unknown" (UNK) category.

In order to conduct the reliability growth (or decay) analysis or to fit any distribution, we need a minimum sample size for every data set. While the ARB data sets for missions ID1 and ID2 could be enough to conduct the analysis presented in the

TABLE I
SANITIZED PROBLEM REPORT CLASSIFICATION EXAMPLES

Description of incident	**Analysis, verification, and/or real-time action**	**Final corrective action**	**Underlying fault**
Fault protection powered off the accelerometer and made the TIMER prime. While the accelerometer was on, it was reporting a pulse count change equivalent to an acceleration of 1.3 times the fault protection limit. The test is performed after a calibration completes.	Review showed that the nominal accelerometer bias was set to "x" in the code. The difference between the nominal bias in the flight software and the actual bias exceeded the fault protection limit. Ground calibration data showed a bias slightly greater than "x", which is within the tolerance of fault protection from the observed bias. If the ground-calibrated bias value had been in the code, this problem would not have occurred. The nominal bias was patched and an accelerometer calibration was repeated. No fault protection activity was observed.	The flight software will be permanently changed via FSC (Flight Software Change) number "Y". This change will be made in version "Z" of the flight software.	BOH
During the attitude control FSW Checkout (YYYY-DOY/HH:MM:SS), telemetry channel reporting the Number of Stars (NoS) incorrectly reported "X" star counts while the star identification (SID) activity was suspended. The NoS channel should always report 0 stars during periods of SID suspend. This incident only occurred during the Yth SID Suspend test. This happened once in flight, and once in a Type Z run with 200 suspends.	See Corrective Action.	Change request W has been created to document this finding. This happened once in flight, and once in a Type Z run with 200 suspends. This problem occurs when SID_SUSPEND or other SID commands cause SID to abort. There is a possibility that the background task will overwrite the NoS channel after the foreground process updates it. The change request was rejected by the Change Review Board held on mm-dd-yyyy, "because it is too late to make FSW changes." Low probability of occurrence. NO FIX.	NAM
Analysis revealed {mission_1} FSW has the potential to miss a sequence abort event. During the next "X" ms frame after an engine aborts SQPM will see the abort state and act accordingly. This issue was discovered during a {mission_2} program software review. However, if another sequence was spawned on the sequence engine that just failed prior to the SQPM check, the state would no longer show the aborted state. Reference anomaly report "Y".	SQPM "polls" VM for sequence aborts by sampling the engine states at "W" Hz. When an engine aborts, during the next "X" ms frame, SQPM sees the abort state and acts accordingly. However, if another sequence was spawned on the engine that just failed prior to the SQPM check, the aborted state no longer shows up. A test has demonstrated the fault, which could be corrected with an SQPM design change to use a VM mechanism to allow a callback routine to be invoked any time a sequence aborts. This will indicate a sequence error, ensuring that all aborts are acted upon.	mm-dd-yy: SQPM Abort Patch successfully installed on the SC on mm/yy. Recommend closure of this report.	NAM
The instrument team discovered that after running for an extended period a stack pointer in instrument flight software becomes confused and as a result writes a telemetry packet into flight software memory over code that is used on boot-up.	The instrument team developed a patch to correct the problem with instrument flight software. This patch was uplinked in real time within instrument commanding to flight software during the "A" and "B" sequences as to eliminate the risk.	The new version of instrument flight software fixes the algorithm so that the stack pointer no longer becomes confused and now correctly writes to the data buffer. The instrument team confirmed via testing and a memory dump that the memory overwrite malfunction is no longer occurring.	ARB
Recent testing indicates that data stack space is not properly deallocated during virtual machine "force unload", "block stop", and "halt" commands, each of which causes threads of executing sequences to be abnormally terminated. Rather than recover the data space memory for further use, the memory is lost.		Stack space is not properly scavenged when a virtual machine engine is stopped abnormally with a "block stop" or "halt" command. This problem also affects an engine running code in another engine if the engine storing the code is a target of a "force unload" command. Routine "X" now pops off all calling information from the engine stack under all termination conditions so that it does not accumulate in the abnormal termination case.	ARB

TABLE II
NUMBER OF FAILURES PER FAULT TYPE PER EACH SPACE MISSION

Mission	BOH	NAM	ARB	UNK	Total failures
ID1	23	12	7	0	42
ID2	85	61	9	1	156
ID3	13	14	0	1	28
ID4	41	25	4	1	71
ID5	19	3	1	0	23
ID6	12	15	0	1	28
ID7	48	8	1	2	59
ID8	47	26	1	0	74

TABLE III
LAPLACE TEST STATISTICS

Mission	BOH	NAM	BOH+NAM
ID1	0.382	−1.417	−0.335
ID2	−1.286	−0.136	**−3.143**
ID3	−2.051	**−3.838**	**−3.817**
ID4	**−5.280**	−1.529	**−4.976**
ID5	0.081	–	0.646
ID6	−2.267	**−2.747**	**−3.255**
ID7	1.609	−0.400	1.309
ID8	−1.444	0.584	−0.893

next sections, there are no other missions with a sufficient number of failures caused by ARBs for comparison. For this reason, we have decided to remove from our analysis all ARB data sets. Furthermore, we have removed the NAM data set related to mission ID5 due to its extremely small sample size. In contrast, we have retained the NAM data set for mission ID7; although the number of failures caused by NAMs is also rather small for this mission, we have several other NAM data sets to compare with.

Each failure report contains the day on which the incident occurred, but not the exact hour and minute. When calculating the TTFs in terms of days, all TTFs computed would thus necessarily be (non-negative) integer values. Moreover, two failures filed on the same day would lead to a TTF of zero. This could cause problems in the subsequent analyses. We therefore randomly assigned each failure to a specific moment on its day of occurrence. This randomization should not greatly influence our results, because it affects the calculated TTFs on the order of tenths of days, while most TTFs amount to much more than a day. To make sure that this is indeed the case, we conducted all analyses based on two different sets of data generated by randomization. Indeed, there were no substantial differences, and in the following we thus present the results obtained for the first randomization carried out.

IV. RELIABILITY TREND ANALYSIS

For missions whose failure history indicates reliability growth, the application of software reliability models to that data can help answer questions about the nature of the failure process, like the following ones: Do the failure processes for the different types of faults differ? For each type of fault, does the failure process remain the same from mission to mission? Answering these questions can help guide decisions concerning maintenance and constraints that should be placed on a system's operational profile during fielded use.

A plot of the running arithmetic averages of the TTFs indicates that software reliability growth may be occurring for several of the eight missions we studied. Figure 2 shows the scaled running arithmetic averages of the TTFs related to BOHs and NAMs for all eight missions; a number of them appear to exhibit reliability growth. To accurately identify those data sets featuring a trend in the TTF sequences, we apply the two-sided Laplace test [33], testing the null hypothesis of "no trend". Per [33], lower and upper asymptotic critical values for the Laplace test statistic indicating a trend at a type I error level of α are given by the $\frac{\alpha}{2}$- and $(1-\frac{\alpha}{2})$-quantiles of the standard normal distribution.

Table III shows the Laplace test statistics calculated for all 23 data sets; values significant at $\alpha = 1\%$ are indicated with bold typeface. The seven data sets exhibiting a trend are:

- ID2_BOH+NAM, ID3_BOH+NAM, ID4_BOH+NAM, and ID6_BOH+NAM. These data sets combine the Bohrbugs and non-aging-related Mandelbugs for missions ID2, ID3, ID4, and ID6, and cover the entire period from launch to the date on which we collected the failure data from the JPL/NASA problem reporting system.
- ID4_BOH. This data set consists of the Bohrbugs for mission ID4 from the entire period from launch to the time at which we collected the data.
- ID3_NAM and ID6_NAM. These data sets consist of the NAMs for missions ID3 and ID6 over the same interval as for the other data sets.

From the fact that all the significant values are negative, we can further conclude that these TTF sequences are subject to reliability growth (rather than reliability decline).

The missions have been numbered chronologically; in Table III, they are arranged from top to bottom in ascending order of launch date. We see that most of the data sets exhibiting reliability growth are from the earlier missions: Five of the seven data sets come from the first four missions, while only two data sets are related to missions ID5–ID8.

We obtain a deeper understanding of the trend in a BOH+NAM data set by distinguishing between the underlying fault types: For missions ID3 and ID6, the trends in the pooled failure data can be traced back to the failures caused by non-aging-related Mandelbugs, while for mission ID4 only the TTFs related to BOHs (but not those related to NAMs) feature a trend. With respect to mission ID2, the "no trend" hypothesis can be rejected neither for the BOH nor for the NAM data; the trend detected in the BOH+NAM data set seems to be an artifact of not differentiating between the two kinds of faults.

It has been conjectured that with increasing operational time the percentage of failures caused by Bohrbugs tends to decrease, whereas the fraction of failures caused by Mandelbugs grows [34, p. 28]. While this assumption seems to be corroborated by the pattern detected for mission ID4, for all the other missions this is not the case.

We applied eleven software reliability models to the seven

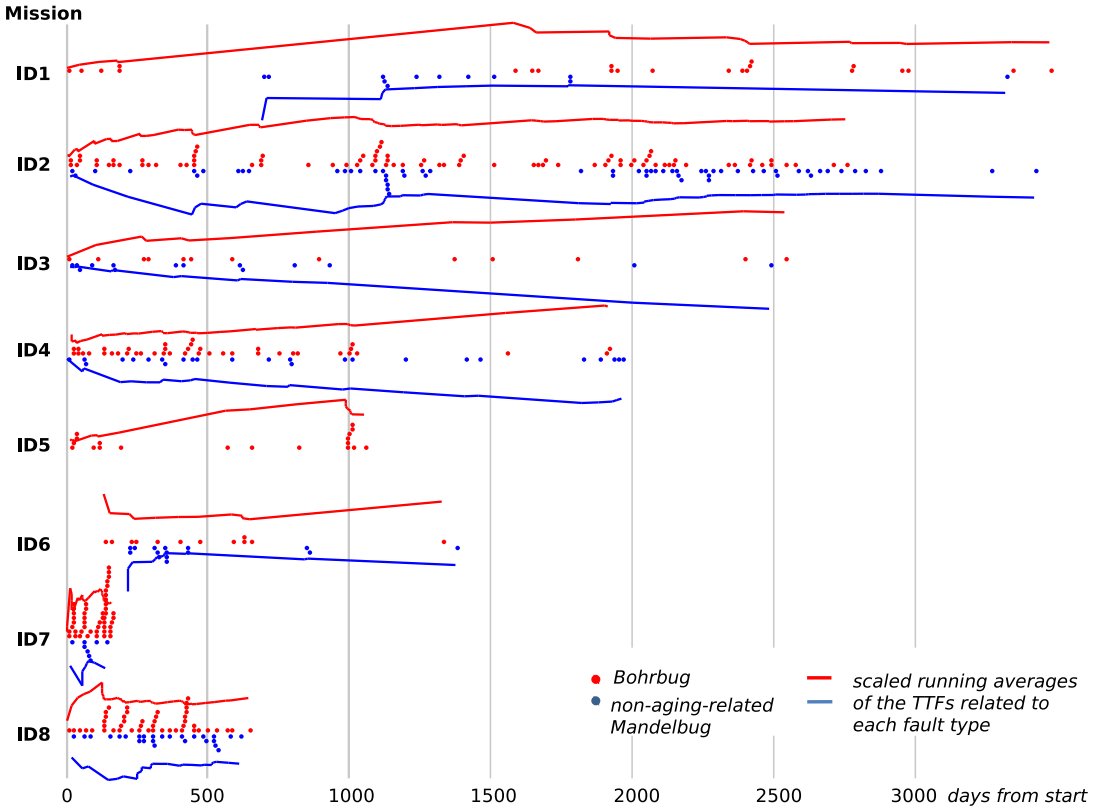

Fig. 2. Scaled running arithmetic averages of the TTFs related to BOHs and NAMs

TABLE IV
SOFTWARE RELIABILITY MODELS APPLIED TO DATA SETS

SMERFS^3		SREPT
Geometric (GEO)	Musa basic (MB)	Goel-Okumoto (GO)
Jelinski-Moranda (JM)	Musa logarithmic (ML)	Weibull (WB)
Littlewood-Verrall linear (LVL)	NHPP	S-shaped (S)
Littlewood-Verrall quadratic (LVQ)		Log-logistic (LOG)

data sets exhibiting reliability growth. The models, implemented in SMERFS^3 [35] and SREPT [36], are listed in Table IV. In using SMERFS^3, we attempt to apply the model applicability criteria of prequential likelihood ("accuracy" in SMERFS^3), bias, bias trend, and model noise [37], in addition to estimating the model parameters and the expected TTFs. The default settings of SMERFS^3 are used in computing the model applicability criteria. For the models implemented in SREPT, we obtain parameter estimates for the models, from which the expected TTFs and the models' mean value functions are computed besides values for the model bias applicability criterion. However, we were not able to compute the applicability criteria for many of the data sets. Because of this, and since the model performance measures for SREPT and SMERFS^3 are different, we had to identify a set of performance measures that can be computed externally to the tools. In addition to the mean square error (MSE), we used the mean absolute scaled error (MASE) and mean absolute percentage error (MAPE) to compare the performance of the models with one another [38].

For each data set, we rank the models that have run successfully according to their MSE, MASE, and MAPE values. We then compute a figure of merit for the model by finding the median of those three ranks. The model having the best performance for a given data set is the one having the lowest median rank. Table V provides an example. The first three columns show the raw values of MSE, MASE, and MAPE for the data set ID4_BOH+NAM across all models. The next three columns show the rank of each model within that data set, and the last column shows the median of the individual ranks. That median is then used as our figure of merit.

Table VI lists these median ranks; Table VII shows the models attaining median rank 1, 2, and 3 (if any) for each data set. If parameter estimation was not successful when fitting a model to a data set this is indicated by a dashed entry in Table VI; performance of this model was then not evaluated for the data set. In Table VII, we differentiate between models assuming an upper bound on the number of failures (indicated by italics) and those that assume no such bound.

We use the results shown in Tables VI and VII to identify patterns and trends in model performance, which would translate to patterns and trends in the failure discovery process.

TABLE V
RANKING MODELS USING MULTIPLE CRITERIA

Model	Criteria values			Criteria ranks			Median rank
	MSE	MASE	MAPE	MSE	MASE	MAPE	
GEO	6.385	2.095	0.113	2	3	1	2
JM	5.391	1.844	0.160	1	1	3	1
LVL	596.44	54.83	1.515	10	11	10	10
LVQ	46823	23.284	0.821	11	10	9	10
MB	6.627	2.044	0.172	3	2	4	3
ML	7.375	2.251	0.126	6	6	2	6
NHPP	6.955	2.101	0.176	5	4	5	5
GO	6.766	2.101	0.176	4	5	6	5
W	16.080	3.165	0.423	7	7	7	7
S	44.235	4.739	8.962	9	9	11	9
LOG	18.602	3.981	0.564	8	8	8	8

TABLE VI
MODEL PERFORMANCE ACROSS FAILURE HISTORIES - MEDIAN RANKS

Model	BOH+NAM				BOH	NAM	
	ID2	ID3	ID4	ID6	ID4	ID3	ID6
GEO	3	2	2	7	6	4	7
JM	7	3	1	6	2	–	6
LVL	9	–	10	11	11	4	–
LVQ	10	10	10	10	6	1	–
MB	4	7	3	5	4	6	6
ML	6	1	6	8	9	2	8
NHPP	–	6	5	3	1	7	4
GO	2	5	5	4	3	9	4
WB	1	8	7	9	5	10	9
S	5	9	9	2	10	8	2
LOG	8	4	8	1	8	3	1

TABLE VII
BEST PERFORMING MODELS FOR EACH FAILURE DATA SET

Median rank	BOH+NAM				BOH	NAM	
	ID2	ID3	ID4	ID6	ID4	ID3	ID6
1	*WB*	ML	*JM*	*LOG*	*NHPP*	LVQ	*LOG*
2	*GO*	GEO	GEO	*S*	*JM*	ML	*S*
3	GEO	*JM*	*MB*	*NHPP*	*GO*	*LOG*	–

Starting with the set of BOH-related failures for mission ID4, Table VII indicates that the three best-performing models all belong to the class of models that assume an upper bound to the number of failures that will eventually be experienced. Furthermore, each of those three models has the best possible rank for the individual performance evaluation criteria (e.g., the highest-ranked model has a rank of 1 for each criterion, and the second ranked model has a rank of 2 for each criterion). Continuing with the NAM-only data sets, we see that the two best-performing models for mission ID3 are of a different type than those for mission ID6: while the former ones belong to the class of models assuming no upper bound on the number of failures that will eventually be observed, the latter ones have such an upper bound.

When we examine the four data sets containing the failures related to both BOHs and NAMs, we see that the preferred type of model for the majority of these data sets is one that assumes an upper bound on the number of failures that will eventually be experienced. This observation would be consistent with a situation in which the number of NAMs were small compared to the number of BOHs. However, Table II indicates that this is not the case – this fact as well as the result that NAM data sets can exhibit reliability growth indicate that we will need to more closely examine the development and operations processes for these missions.

V. DISTRIBUTIONAL ANALYSIS

We now restrict the analysis to those data sets for which the Laplace test could not reject the no-trend hypothesis. For each of these data sets, all TTFs observed may have been sampled from distributions sharing the same expected value. Assuming more specifically that within a data set all TTFs are independently and identically distributed (iid), it is of high interest to identify (an appropriate model for) each underlying TTF distribution.

To this end, we employ a number of parametric distributions that have often been used to model TTF data, namely the Weibull distribution with cumulative distribution function (cdf)

$$F(t) = 1 - \exp\left[-\left(\frac{t}{\theta}\right)^k\right]$$

and the gamma distribution with cdf

$$F(t) = \frac{1}{\Gamma(k)\theta^k}\int_0^t x^{k-1}\exp\left(-\frac{x}{\theta}\right)dx.$$

For both distributions, $\theta > 0$ is the scale parameter and $k > 0$ is the shape parameter. Also, each of these distributions features a decreasing (increasing) failure rate if the shape parameter is smaller (larger) than one. For $k = 1$, either of the distributions is reduced to the exponential distribution (with mean θ and constant failure rate $1/\theta$), the third parametric model included in our analysis.

We use maximum likelihood estimation to fit the three distributions to each of the data sets. While this method determines the parameter value(s) for which it is most plausible that the observed data were generated by the respective model, it is possible that even the fitted model does not represent the data well.

A popular approach to evaluating the goodness of fit is the Kolmogorov-Smirnov (KS) test. In its original form, it tests the null hypothesis that a data set constitutes iid samples from a certain continuous distribution with known parameters; the cdf $F_0(t)$ of this hypothesized distribution is thus fully specified. Under these circumstances, the distribution of the KS test statistic, derived by multiplying the square root of the sample size n with the maximum absolute deviation between $F_0(t)$ and the empirical cdf, depends on n, but it does neither depend on the type of the hypothesized distribution nor on its parameter values. Regardless of the hypothesis tested, it is thus possible to make a test decision based on the critical values contained in one table, and implemented in many statistical tools.

However, this is different if some or all of the parameter values have to be estimated. While the results by David

TABLE VIII
KS TEST STATISTICS, BEST CANDIDATE MODELS AND THEIR RESPECTIVE FAILURE RATE BEHAVIOR

Data set	KS (exponential)	KS (Weibull)	KS (gamma)	Best candidate model	$\hat{\theta}$	$\hat{k}$	Failure rate
ID1_BOH+NAM	**1.337**	**1.186**	0.565	gamma	173.4	0.572	decreasing
ID5_BOH+NAM	**1.777**	**1.101**	0.816	gamma	114.0	0.419	decreasing
ID7_BOH+NAM	**1.515**	**1.514**	1.041	gamma	4.514	0.636	decreasing
ID8_BOH+NAM	0.980	**1.665**	0.544	gamma	11.37	0.776	decreasing
ID1_BOH	**1.423**	0.956	0.576	Weibull	90.84	0.568	decreasing
ID2_BOH	1.123	**1.753**	0.659	gamma	44.97	0.720	decreasing
ID3_BOH	0.655	0.863	0.767	exponential	194.9	—	constant
ID5_BOH	**1.713**	**1.023**	0.773	gamma	136.2	0.406	decreasing
ID6_BOH	0.863	**1.021**	0.689	gamma	185.8	0.593	decreasing
ID7_BOH	**1.410**	**1.424**	0.809	gamma	5.693	0.588	decreasing
ID8_BOH	0.744	**1.184**	0.814	exponential	13.70	—	constant
ID1_NAM	1.201	0.858	0.646	Weibull	179.8	0.591	decreasing
ID2_NAM	1.258	**1.707**	0.926	gamma	82.58	0.679	decreasing
ID4_NAM	0.513	**1.101**	0.500	exponential	78.31	—	constant
ID7_NAM	0.440	0.707	0.471	exponential	17.33	—	constant
ID8_NAM	0.987	0.928	0.988	exponential	23.56	—	constant

and Johnson [39] imply that the distribution of the KS test statistic does not depend on the unknown parameters as long as they are location or scale parameters estimated via the maximum likelihood method, it does depend on the type of the distribution in the null hypothesis. For each type of distribution tested a different set of critical values is required.

Using Monte Carlo simulations with 5000 repetitions, Lilliefors [40] calculated critical values of the KS test statistic for testing an exponential distribution with unknown mean θ. Durbin [41] derived exact expressions for these critical values, and tabulated them for sample sizes n between two and 100. He also showed that a compact approximation due to Stephens and published by Pearson and Hartley [42, p. 359] is quite accurate, especially for large sample sizes. For $n > 100$, we will therefore employ this approximation.

As noted above, the Weibull distribution includes a shape parameter k. However, Chandra et al. [43] have shown that testing whether a sample $x_1, ..., x_n$ is from a Weibull distribution with unknown θ and k is equivalent to testing whether $-\ln x_1, ..., -\ln x_n$ have been sampled from an extreme-value distribution with unknown location parameter $\xi = -\ln\theta$ and scale $\delta = 1/k$. The critical values of the related KS test statistics, tabulated by Chandra et al. for various sample sizes based on Monte Carlo simulations with 10000 repetitions, do not depend on these parameters.

If the shape parameter of the gamma distribution needs to be estimated, then the distribution of the KS statistic depends on the true value of k [44, p. 151]. Critical values derived via Monte Carlo simulations employing one specific k, like the ones carried out by Tadikamalla [45] for $k = 1$, are thus applicable only when the true k (or its maximum likelihood estimate $\hat{k}$) is close to this value. While Kulinskaya [46] presented an algorithm for calculate the *asymptotic* distribution of the KS test statistic based on k or $\hat{k}$, we do not know of any work discussing how to analytically obtain critical values for a given *finite* sample size n and a specific (estimated) shape parameter value. For each of the data sets in our study, we therefore use a Monte Carlo simulation (with 100000 repetitions) to compute critical values of the KS test statistic for testing the null hypothesis of a gamma distribution with shape $\hat{k}$ (the maximum likelihood estimate derived from this data set), with n equaling the respective sample size.

Columns 2–4 of Table VIII contain the KS test statistics obtained for all data sets when testing the hypotheses that the TTFs are iid samples from the exponential, the Weibull, and the gamma distribution. Indicated in bold type are those values significant at a 99% confidence level; i.e., the null hypothesis that the respective distribution generated the TTFs can be rejected at a type I error level of 1%. There is no data set for which all three parametric models are rejected; in several cases, two or even all three distributions are retained as "candidate models". However, if the exponential distribution is among the candidate models, the fact that it is a special case of the Weibull and the gamma distribution begs the question whether these more complicated distributions are indeed needed. To determine the best model among the candidate models, we employ the Akaike information criterion (AIC), calculated as [47]

$$\text{AIC} = -2 \cdot \ln(\text{maximum likelihood}) + 2 \cdot p;$$

it balances the maximum log-likelihood value attained, representing the model fit, with the number of model parameters p, thus penalizing for a higher model complexity. In Table VIII, column 5 indicates the best candidate model for each data set, i.e., the one with the lowest AIC value. Moreover, in columns 6 and 7 we list the maximum likelihood estimates of θ and, where applicable, of k calculated for this model. Finally, column 8 explicitly shows whether the fitted best model has a constant, an increasing, or a decreasing failure rate.

From the table, we can see that for all BOH+NAM data

sets without an overall trend, the TTFs are best modeled with gamma distributions featuring a decreasing failure rate. However, for mission ID7 this decreasing failure rate can be attributed to the Bohrbugs only; the best fit to the ID7_NAM data set is achieved by the exponential distribution. Since the exponential distribution is the best model for both the BOH and the NAM data set of mission ID8, the decreasing failure rate obtained for the combined TTFs data set seems to be an artifact of not separating between the fault types responsible for the failures. Again, as highlighted in the trend analysis, it is thus advisable to take into account the underlying fault type instead of studying the TTFs of the pooled failures.

According to the best candidate models fitted, most of the BOH-only data sets show a decreasing failure rate; although there is no *overall* trend in the TTFs sequences, after a failure occurrence the tendency that another failure will occur (given that it has not yet done so) is a declining function of the waiting time. Only for two of the BOH data sets the failure rate tends to be constant. This is different for the TTFs related to non-aging-related Mandelbugs, where three of the five data sets without a trend are best modeled with an exponential distribution.

All in all, there are thus five data sets that do not seem to feature a trend and where the TTFs might be independent realizations of identical, exponentially-distributed random variables. This suggests that for these data sets the failure times can be modeled with a homogeneous Poisson process.

VI. Discussion and conclusion

Table III indicates that earlier missions (ID1–ID4) are more likely to exhibit reliability growth during operations than more recent missions (ID5–ID8). The following possibilities are being explored:

- Space mission on-board software is becoming larger and implements more functionality. For the more recent missions, this may increase the likelihood that more faults are left in the software after launch. Although faults may still be repaired as they are detected, the missions' operational profile may be such that previously unseen faults continue to be exposed over the missions' lifetimes at a rate that effectively counteracts the repair rate.
- On-board software for recent missions may be modified more during mission operations. Robotic planetary exploration spacecraft do not need to be launched with the complete set of functionality required to perform their planned tasks at their destination – the software may be updated en-route or even after reaching the destination to perform its tasks more effectively. For example, the GALILEO on-board software was extensively modified after reaching Jupiter to compensate for the malfunctioning high-gain downlink antenna, and the Voyager 2 on-board software was modified during cruise interval to compensate for the low-light environment of Neptune. More frequent updates can introduce more faults into the on-board software during operations, increasing the effective fault discovery rate.

Table VIII shows that the exponential and gamma models are candidate models for all but two of the data sets for which the Laplace test did not indicate reliability growth at the 1% significance level. For BOHs only, the gamma model indicating a decreasing failure rate is the best candidate in four cases, the exponential model indicating a constant failure rate is the best candidate in two cases, and the Weibull model indicating a decreasing failure rate is the best candidate in one case. For NAMs only, the exponential model indicating a constant failure rate is the best candidate in three cases, and the gamma and Weibull models indicating a decreasing failure rate are the best candidates in one case each. These observations indicate that our initial conjecture about the way in which the proportions of BOHs and NAMs change during a mission applies in some situations, but not consistently. Two possibilities come to mind. For the first, note that our conjecture holds for more of the later missions than for earlier missions – for the later missions, less of the missions' lifetimes have been observed, and we may not yet have observed a large enough portion of the later missions' lifetimes to see the eventual decrease in the NAMs failure rate as we do for the earlier missions. A second possibility is the one already given above: on-board software for recent missions may be more extensively modified than that for earlier missions.

For the BOHs and NAMs taken together, the gamma model indicating a decreasing failure rate is the best candidate in all four cases. This is consistent with results of our earlier work [5] indicating that the proportion of BOHs observed during operations is significantly larger than the proportion of NAMs. In this case, even if the failure rate of the NAMs remains constant, the total failure rate will decrease if the proportion of BOHs is larger than that of the NAMs. Since BOHs should be easier to find and repair than NAMs, this suggests that there may be opportunities to improve the effectiveness of fault identification activities during development.

We are conducting more detailed analyses of the missions' characteristics to explore these possibilities in more detail, and to draw conclusions about the maintenance, fault tolerance, and failure mitigation methods employed.

Acknowledgment

The work 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 #1440119. FAU research was supported by the Dr. Theo and Friedl Schoeller Research Center for Business and Society. We thank Dr. Martin Feather in the Software Product and Process Assurance Group at the Jet Propulsion Laboratory for the visualization of the TTF data sets shown in Figure 2.

REFERENCES

[1] D. Dvorak, Ed., "NASA study on flight software complexity," NASA, Tech. Rep. NASA Office of Chief Engineer, 2009. [Online]. Available: http://www.nasa.gov/pdf/418878main_FSWC_Final_Report.pdf

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

[3] M. Grottke and K. S. Trivedi, "Software faults, software aging and software rejuvenation," *Journal of the Reliability Engineering Association of Japan*, vol. 27, no. 7, pp. 425–438, 2005.

[4] M. Grottke and K. S. Trivedi, "Fighting bugs: Remove, retry, replicate, and rejuvenate," *IEEE Computer*, vol. 40, no. 2, pp. 107–109, 2007.

[5] M. Grottke, A. P. Nikora, and K. S. Trivedi, "An empirical investigation of fault types in space mission system software," in *Proc. 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2010, pp. 447–456.

[6] J. Gray, "Why do computers stop and what can be done about it?" Tandem Computers, Tech. Rep. 85.7, PN87614, 1985.

[7] D. Dörner, *The Logic of Failure: Recognizing and Avoiding Error in Complex Situations*. Cambridge: Perseus Books, 1997.

[8] M. Grottke, R. Matias, and K. S. Trivedi, "The fundamentals of software aging," in *Proc. 1st International Workshop on Software Aging and Rejuvantion*, 2008.

[9] E. Marshall, "Fatal error: How Patriot overlooked a Scud," *Science*, vol. 255, no. 5050, p. 1347, 1992.

[10] A. Avritzer and E. J. Weyuker, "Monitoring smoothly degrading systems for increased dependability," *Empirical Software Engineering*, vol. 2, no. 1, pp. 59–77, 1997.

[11] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert, "Proactive management of software aging," *IBM Journal of Research & Development*, vol. 45, no. 2, pp. 311–332, 2001.

[12] K. Chaudhuri, A. Kothari, R. Swaminathan, R. Tarjan, A. Zhang, and Y. Zhou, "Server allocation problem for multi-tiered applications," HP Labs, Tech. Rep. HPL-2004-151, 2004.

[13] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, "Analysis of software aging in a Web server," *IEEE Transactions on Reliability*, vol. 55, pp. 411–420, 2006.

[14] K. S. Trivedi, M. Grottke, and E. C. Andrade, "Software fault mitigation and availability assurance techniques," *International Journal of Systems Assurance Engineering and Management*, vol. 1, no. 4, pp. 340–350, 2010.

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

[16] J. Gray, "Why do computers stop and what can be done about it?" in *Proc. 5th Symposium on Reliability in Distributed Software and Database Systems*, 1986, pp. 3–12.

[17] J. Xu, Z. Kalbarczyk, and R. K. Iyer, "Networked Windows NT system field failure data analysis," in *Proc. 1999 Pacific Rim International Symposium on Dependable Computing*, 1999, pp. 178–185.

[18] N. Talagala and D. Patterson, "An analysis of error behavior in a large storage system," EECS Department, University of California, Berkeley, Tech. Rep. UCB/CSD-99-1042, 1999.

[19] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler, "An empirical study of operating systems errors," in *Proc. Eighteenth ACM Symposium on Operating Systems Principles*, 2001, pp. 73–88.

[20] P. Enriquez, A. Brown, and D. Patterson, "Lessons from the PSTN for dependable computing - a study of FCC disruption reports," in *Proc. Workshop on Self-Healing, Adaptive, and Self-Managed Systems*, 2002.

[21] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *Proc. 4th Conference on USENIX Symposium on Internet Technologies and Systems*, vol. 4, 2003, pp. 1–16.

[22] A. R. Hoffman, N. H. Green, and H. B. Garrett, "Assessment of In-flight Anomalies of Long Life Outer Planet Missions," in *Environmental Testing for Space Programmes*, ser. ESA Special Publication, K. Fletcher, Ed., vol. 558, Aug. 2004, pp. 43–50.

[23] S. Peret and P. Narasimham, "Causes of failure in web applications," Carnegie Mellon University, Tech. Rep. CMU-PDL-05-109, 2005.

[24] N. W. Green, A. R. Hoffman, and H. B. Garrett, "Anomaly trends for long-life robotic spacecraft," *Journal of Spacecraft and Rockets*, vol. 43, no. 1, pp. 218–224, 2006.

[25] 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," in *Proc. 44th AIAA Aerospace Sciences Meeting and Exhibit*, 2006, pp. 1–9.

[26] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *Proc. 5th USENIX Conference on File and Storage Technologies*, 2007, pp. 17–28.

[27] W. Jiang, C. Hu, Y. Zhou, and A. Kanevsky, "Don't blame disks for every storage subsystem failure," *The USENIX Magazine*, vol. 33, no. 3, pp. 22–31.

[28] D. Long, A. Muir, and R. Golding, "A longitudinal survey of internet host reliability," in *Proc. 14th Symposium on Reliable Distributed Systems*, 1995, pp. 2–9.

[29] T. Heath, R. P. Martin, and T. D. Nguyen, "Improving cluster availability using workstation validation," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 1, pp. 217–227, 2002.

[30] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, and Y. Zhang, "Failure data analysis of a large-scale heterogeneous server environment," in *Proc. 2004 International Conference on Dependable Systems and Networks*, 2004, pp. 772–781.

[31] D. Nurmi, J. Brevik, and R. Wolski, "Modeling machine availability in enterprise and wide-area distributed computing environments," in *Proc. 11th International Euro-Par Conference on Parallel Processing*, 2005, pp. 432–441.

[32] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *Proc. International Conference on Dependable Systems and Networks*, 2006, pp. 249–258.

[33] K. Kanoun and J.-C. Laprie, "Trend analysis," in *Handbook of Software Reliability Engineering*, M. Lyu, Ed. McGraw-Hill, 1996, ch. 10.

[34] S. Orlando, "Software aging analysis of off the shelf software items," Ph.D. dissertation, Università degli Studi di Napoli Federico II, 2007.

[35] W. H. Farr. (2012, May) SMERFS downloads. [Online]. Available: http://www.slingcode.com/smerfs/downloads/#SMERFS3

[36] S. Ramani, S. S. Gokhale, and K. S. Trivedi, "SREPT: Software Reliability Estimation and Prediction Tool," *Performance Evaluation*, vol. 39, no. 1-4, pp. 37–60, 2000.

[37] A. A. Abdel-Ghaly, P. Y. Chan, and B. Littlewood, "Evaluation of competing software reliability predictions," *IEEE Transactions on Software Engineering*, vol. 12, no. 9, pp. 950–967, 1986.

[38] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[39] F. N. David and N. L. Johnson, "The probability integral transformation when parameters are estimated from the sample," *Biometrika*, vol. 35, no. 1-2, pp. 182–190, 1948.

[40] H. W. Lilliefors, "On the Kolmogorov-Smirnov test for the exponential distribution with mean unknown," *Journal of the American Statistical Association*, vol. 64, no. 325, pp. 387–389, 1969.

[41] J. Durbin, "Kolmogorov-Smirnov tests when parameters are estimated with applications to tests of exponentiality and tests on spacings," *Biometrika*, vol. 62, no. 1, pp. 5–22, 1975.

[42] E. S. Pearson and H. O. Hartley, *Biometrika Tables for Statisticians, Vol. II*. Cambridge University Press, 1972.

[43] M. Chandra, N. D. Singpurwalla, and M. A. Stephens, "Kolmogorov statistics for tests of fit for the extreme-value and Weibull distributions," *Journal of the American Statistical Association*, vol. 76, no. 375, pp. 729–731, 1981.

[44] R. B. D'Agostino and M. A. Stephens, *Goodness-of-Fit Techniques*. New York: M. Dekker, 1986.

[45] P. R. Tadikamalla, "Kolmogorov-Smirnov type test-statistics for the gamma, Erlang-2 and the inverse Gaussian distributions when the parameters are unknown," *Communications in Statistics – Simulation and Computation*, vol. 19, no. 1, pp. 305–314, 1990.

[46] E. Kulinskaya, "Coefficients of the asymptotic distribution of the Kolmogorov-Smirnov statistic when parameters are estimated," *Journal of Nonparametric Statistics*, vol. 5, no. 1, pp. 43–60, 1995.

[47] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716 – 723, 1974.