How does testing affect the availability of aging software systems?

Michael Grottke[a,*], Benjamin Schleich[a]

[a] *University of Erlangen-Nuremberg, Germany*

Abstract

This paper proposes an approach to examining how testing affects the operational behavior of aging software systems. Such an approach requires models for the testing phase and the operational phase that explicitly account for crash failures due to both aging-related and non-aging-related bugs. We develop appropriate semi-Markov models and derive expressions for computing the respective transient and steady-state probabilities needed. Our numerical examples suggest that disregarding the effects of non-aging-related bugs can result in wrong conclusions about the testing phase and the operational phase.

Moreover, we show how to combine the two models for a joint analysis in which metrics of interest concerning the operational phase, like the optimal rejuvenation rate, are random variables whose distributions are influenced by the potential outcomes of testing.

Keywords: rejuvenation rate, semi-Markov process, software aging, steady-state availability, testing phase, transient state probability

*Corresponding author. Tel. +49 911 5302 276, Fax +49 911 5302 277, E-mail: `Michael.Grottke@wiso.uni-erlangen.de`.

1. Introduction

"Software aging", first studied formally by Huang et al. [20] under the term "process aging", relates to the phenomenon that during operation a software system may show performance degradation and an increasing failure rate which cannot be attributed to changes in the user behavior or the software code. In fact, software aging is caused by specific software faults (or "bugs"), known as aging-related bugs [16, 17, 18]. Typically, the activation of such faults leads to error states that accumulate inside the running system. The gradual shifting from an error-free to a failure-probable internal state is referred to as the "aging effect" [14]. Classes of aging effects include resource leakage (e.g., memory leakage due to unused but unterminated processes and disk space depletion caused by increasing error log files), the accumulation of round-off errors, and the accrual of data corruption [14]. When trying to determine the percentage of aging-related bugs among the faults contained in a specific software based on failure reports, it is essential to classify unique software faults. Repeated failure occurrences related to a fault already included in the analysis should thus be omitted. The examination of 520 unique software faults in the flight software of 18 JPL/NASA missions [15] revealed that 4.4% of them were aging-related bugs, while 93.5% were non-aging-related bugs; the other faults could not be classified based on the available information.

The technique of proactively removing aging effects via system reboots, application restarts, etc. has been known as "software rejuvenation" [20]. Much research has focused on aging-related bugs causing the system to hang or crash and the question of how frequently to trigger software rejuvenation during system operation from an availability or costs perspective. The model used by Huang et al. [20] for this purpose consists of four states: a highly-robust state in which the system is safe from immediate failures, a failure-probable state in which the system has aged sufficiently for a failure to occur, a failure (crash) state, and a rejuvenation state. Since all transition distributions between these states are assumed to be exponential, this model forms a continuous-time Markov chain. It is generalized by Dohi et al. [7, 8, 9] into a semi-Markov model with general transition distributions. Like in the original model, Dohi et al. assume that software rejuvenation is only triggered in the failure-probable state. This implies that the system or the user is able to perfectly detect at any given time whether or not the presence of aging effects requires the system to be rejuvenated. However, this is often not the case. Therefore, Garg et al. [10] present a Markov regenerative stochastic Petri net in which the running system is always rejuvenated at a fixed time after it entered the highly-robust state. The model allows the distribution of the time to carry out software rejuvenation to depend on the system state in which it was triggered. Removing this possibility, which Garg et al. do not make use of in their numerical example anyway, Suzuki et al. [35] show that the model can be transformed into a semi-Markov process consisting of three states only: an up state, a crash state, and a rejuvenation state. This three-state structure has been employed in many papers investigating software rejuvenation. For example, Garg et al. [11] and Vaidyanathan and Trivedi [37] represent the software behavior by such a three-state semi-Markov model. Similarly, the "operational state model I" used by Salfner and Wolter [33] is a stochastic Petri net consisting of three places corresponding to the above-mentioned states.

In this paper, we present an approach for studying how the testing of an aging software system affects its behavior in the operational phase. How does the length of the testing phase influence the optimal frequency of triggering software rejuvenation, and the availability attainable? How should we allocate our time budget to various testing techniques? When trying to answer these questions, we have to note that system crashes during usage are caused not only by aging-related bugs, but by non-aging-related bugs as well. However, software rejuvenation cannot reduce the risk of future failures due to non-aging-related bugs. Moreover, because of inherent differences between the two fault types it can be expected that their proportions among the faults detected change as testing proceeds: Aging-related bugs tend to show themselves when the software has been running uninterruptedly for some time; this is more likely to happen later in the testing phase, after many of those non-aging-related bugs that make the software crash have been found and removed. For our analyses it is therefore crucial to distinguish between the two fault types in the testing phase as well as in the operational phase.

None of the described models explicitly account for crashes caused by non-aging-related bugs. This is especially obvious for the models by Huang et al. [20], Dohi et al. [7, 8, 9] and Garg et al. [10], where crash failures can only occur after the system has entered the failure-probable state.

Only few research works have dealt with the testing of aging software systems. Matias and Freitas Filho [28] and Matias et al. [27] show how design of experiments can be employed to define test setups suitable for evaluating the influence of various workload factors on aging effects. Building on this work, Carrozza et al. [2] propose an approach to reducing the number and the duration of tests required to detect the presence of software aging at different levels of the workload factors, and to estimate aging trends in performance parameters monitored. In contrast, Matias et al. [27, 29] focus on the distribution of the time to aging-related failure during normal operation to characterize the aging behavior. In [29], they estimate this distribution using quantitative accelerated life tests, which decrease the test duration needed to obtain a sufficient sample size by testing the system at higher-than-normal stress levels. The accelerated degradation test technique employed in [27] does not even require failure time data for estimating the failure time distribution but instead relies on observations of a suitable degradation measure made under various stress levels.

While these approaches are directed at designing test campaigns that help detect and quantify the aging behavior during operations, to the best of our knowledge none of the previous research on software aging deals with the detection and correction of software faults in aging systems during the testing phase.

In the field of software reliability engineering [25, 30, 31], many software reliability models tracking the number of faults removed or faults remaining have been developed. Most of them are one-dimensional counting processes belonging to the class of self-exciting point processes, in which the future of the process may (but does not have to) depend on parts of its own history [4, 12, 13, 22]. For example, in the binomial models [30, pp. 259-267], like the well-known Jelinski-Moranda model [21], the program hazard rate is always the product of the current fault content and the per-fault hazard rate, whereas the program hazard rate does not at all depend on the process history but only on global time (i.e., the time since testing started) in the non-homogeneous Poisson process models [30, pp. 255-259]; both binomial models and non-homogeneous Poisson process models are Markovian. In contrast, according to the Littlewood-Verrall model [24] the program hazard rate is not only influenced by global time and the current fault content, but also by the time since the last fault removal; this model can be represented by a semi-Markov process. To evaluate the effects of software testing on the behavior of an aging system in the operational phase we need a model for the testing phase that separately keeps track of the removal of aging-related bugs and non-aging-related bugs. There does not seem to be an existing software reliability growth model that we can use for our purpose.

This paper is structured as follows: In Section 2, we develop a model for the detection and removal of both aging-related bugs and non-aging-related bugs during testing. Similarly, we present a model for the system behavior in the operational phase that explicitly accounts for crash failures due to both types of faults in Section 3. In Section 4, we then show how the two models can be combined to study the potential effects that testing will have on the system behavior in the operational phase. In particular, we point out that the optimal rejuvenation rate and the optimal availability attainable are random variables; the potential outcomes of testing can be discussed based on the distributions and the expected values of these random variables. Finally, Section 5 concludes the paper.

2. Modeling the testing phase

2.1. Model formulation

During dynamic testing, the software under test is executed, and faults in the code are corrected once they have shown themselves by causing the software to fail. For studying the removal of aging-related and non-aging-related bugs during the testing phase we propose a model based on the following assumptions:

A1 At the beginning of testing, the software contains m aging-related bugs and n non-aging-related bugs that can lead to crash failures.

A2 As soon as a fault causes a crash it is immediately corrected without introducing any new faults, and the software is instantaneously restarted. This restart removes all symptoms of aging like the internal error conditions accrued so far.

A3 Upon a restart, the time to failure due to aging-related bugs follows a gamma distribution with shape $\alpha > 1$ and rate $\beta_j \equiv \beta \cdot (m - j)$, where j is the number of aging-related bugs removed so far; i.e., its probability density function (pdf) and cumulative distribution function (cdf) are given by

$$f_\Gamma(x; \alpha, \beta_j) = \frac{\beta_j^\alpha}{\Gamma(\alpha)} x^{\alpha-1} \exp(-\beta_j x) \quad \text{and} \quad F_\Gamma(x; \alpha, \beta_j) = 1 - \frac{\Gamma(\alpha, \beta_j x)}{\Gamma(\alpha)},$$

respectively, with gamma function $\Gamma(\alpha) = \int_0^\infty \xi^{\alpha-1} \exp(-\xi) d\xi$ and upper incomplete gamma function $\Gamma(\alpha, \beta_j x) = \int_{\beta_j x}^\infty \xi^{\alpha-1} \exp(-\xi) d\xi$.

A4 After i non-aging-related bugs have been removed, the time to failure due to non-aging-related bugs follows an exponential distribution with rate $\phi_i \equiv \phi \cdot (n-i)$; i.e., its pdf and cdf are given by

$$f_{\exp}(x;\phi_i) = \phi_i \exp(-\phi_i x) \quad \text{and} \quad F_{\exp}(x;\phi_i) = 1 - \exp(-\phi_i x),$$

respectively.

Our model only takes into account the time actually spent on executing the software during testing, as is done in most software reliability growth models [30]. According to assumption A2, fault removal and restart of the system therefore happen instantaneously upon a crash. Moreover, fault removal is considered to be perfect, an assumption made by many software reliability growth models, e.g., by all models of binomial type [30, pp. 260–261], including the Littlewood model [23], the Schick-Wolverton model [34], and the Jelinski-Moranda model [21]. Like in the latter model we also assume that the per-fault hazard rate is the same constant ϕ for each non-aging-related bug.

While a time-constant hazard rate seems appropriate for those faults that do not cause software aging, each of the gamma distributions with shape parameter α larger than one used for modeling the times to failure due to aging-related bugs has an increasing hazard rate [36, p. 132]. For $\alpha = 2$, the gamma distribution degenerates to the two-stage Erlang distribution chosen by Vaidyanathan and Trivedi [37] for approximating the time to aging-related failure in their comprehensive model of software behavior in the operational phase. More generally, for $\alpha \in \mathbb{N}^+ \diagdown \{1\}$ it becomes the α-stage Erlang distribution, which intuitively represents software aging due to the accumulation of internal error states: Each one of the $(m-j)$ aging-related bugs remaining in the software causes an error when activated. Assuming that all faults have the same constant activation rate β, such errors are created at rate $\beta \cdot (m - j)$. After α errors have accrued, the software fails. The gamma distribution suggests itself as a candidate for generalization.

The values of the parameters α, β and ϕ depend on the testing technique employed as well as on other factors, like the capability of the testers or the complexity of the software under test.

Of course, our assumptions do not perfectly describe reality. For example, the assumption that faults are always removed without introducing any new faults is too good to be true. Moreover, the identical activation rates among all aging-related bugs (β) and among all non-aging-related bugs (ϕ) imply that the faults of each type are located in the software uniformly relative to the testing profile used, which need not be the case. Also, the activation rates might be affected by time or by other aspects; e.g., the activation rate of non-aging-related bugs could in fact increase with the extent of error accumulation due to aging-related bugs. Furthermore, based on our discussion of the Erlang distribution, using the gamma distribution for modeling the time to failure due to aging-related bugs basically means that all errors caused by these bugs jointly deteriorate the "health" of the running system, until the activation of one aging-related bug finally leads to a failure. In reality, different kinds of aging effects (e.g., memory leakage and unreleased file handlers) may accumulate independently of each other.

Each of these assumptions could be replaced with more realistic counterparts. For example, we could consider one gamma (or Erlang) distribution for the time to failure due to each type of aging effect, modeling the overall time to aging-related failure by the first order statistic of these individual times. However, we would then have to make further assumptions regarding the number of independent aging effects as well as the parameters of the individual distributions. This would introduce not only additional complexity, but also a higher degree of arbitrariness, as long as we do not have enough knowledge about the mechanics of various aging effects. (In the long run, studies like the one presented by Macêdo et al. [26] may help the research community attain the understanding required.) Moreover, as the first of its kind we wish to keep this model explicitly accounting for both aging-related and non-aging-related bugs as simple as possible, while capturing an important aspect neglected by previous research: The aging behavior of a software system is influenced by crash failures due to non-aging-related bugs. We are confident that the main results obtained from our model will not be affected by future refinements of individual assumptions.

We use the tuple (i, j) to define the current state of the software, with i denoting the number of non-aging-related bugs removed so far, and j representing the number of aging-related bugs removed so far. Let the stochastic process $\{Z(t), t \geq 0\}$ model the development of the state of the software over time. According to assumption A3, the future behavior of the process depends not only on its current state, but also on the time x already spent in this state (i.e., the time since the last fault removal). Therefore, $\{Z(t), t \geq 0\}$ follows a semi-Markov model with state space $\mathcal{Z} = \{(0, 0), (0, 1), \dots, (n, m)\}$. This semi-Markov model is shown in Figure 1.

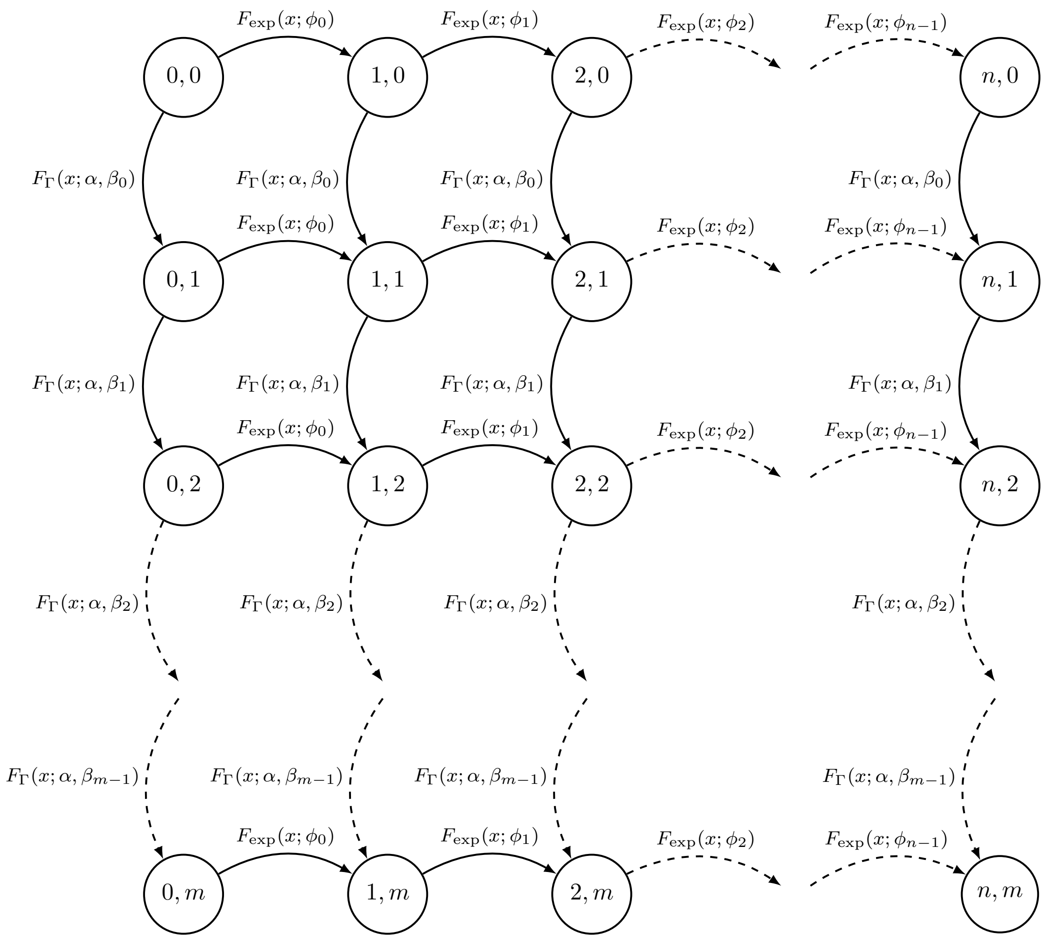


Figure 1: Semi-Markov model for fault removal in the testing phase

2.2. Deriving transient state probabilities

The probability that exactly i non-aging-related bugs and j aging-related bugs have been removed after testing the software for t_e time units corresponds to the transient state probability of state (i, j) at time t_e, given that the system started in state $(0, 0)$ at time 0. In general, the transient state probabilities

$$\pi_{z_s z_e}(t) \equiv P(Z(t) = z_e | Z(0) = z_s), \qquad z_s, z_e \in \mathcal{Z},$$

satisfy the following equation [6]:

$$\pi_{z_s z_e}(t) = \delta_{z_s z_e} \cdot \overline{F}_{z_s}(t) + \sum_{z \in \mathcal{S}(z_s)} \int_0^t R(z_s, z, t-\theta)\pi_{z z_e}(\theta) d\theta. \tag{1}$$

The indicator function $\delta_{z_s z_e}$ is equal to one if $z_s = z_e$, and zero otherwise. $\overline{F}_{z_s}(t)$ denotes the survival function of the sojourn time in state z_s. $\mathcal{S}(z_s)$ is the set of all states directly reachable from state z_s. $R(z_s, z, t-\theta)$ represents the kernel of the semi-Markov process,

$$R(z_1, z_2, t) = P(V_{n+1} = z_2, T_{n+1} - T_n \leq t | V_n = z_1), \quad n = 0, 1, 2.$$

T_n is the time of the n^{th} transition, while $V_n \equiv Z(T_n+)$ is the current state right after this n^{th} transition. The event that upon entering state z_1 a direct transition to state z_2 will occur no later than after t time units implies that no other state will be entered prior to this transition. Based on $f_{z_1 z}(x)$ and $F_{z_1 z}(x)$, the pdf and the cdf assigned to the transition from state z_1 to state $z \in \mathcal{S}(z_1)$, the conditional probability embodied by the kernel can therefore be expressed as

$$R(z_1, z_2, t) = \int_0^t f_{z_1 z_2}(x) \cdot \prod_{\substack{z \in \mathcal{S}(z_1) \\ z \neq z_2}} (1 - F_{z_1 z}(x)) dx, \quad z_2 \in \mathcal{S}(z_1).$$

For any value x of the sojourn time, the integrand is the product of the pdf $f_{z_1 z_2}(x)$, representing the tendency that z_2 is entered a this very moment, and the expression $\prod_{\substack{z \in \mathcal{S}(z_1) \\ z \neq z_2}} (1 - F_{z_1 z}(x))$, the probability that no transition to any other state directly reachable from z_1 has occurred so far.

Evaluating the transient probabilities based on Equation (1) requires the computation of convolutions. It is thus advisable to solve the problem in the

Laplace domain. For the Laplace transform of $\pi_{z_s z_e}(t)$ follows in general [6]:

$$\pi^*_{z_s z_e}(s) = \delta_{z_s z_e} \cdot \overline{F}^*_{z_s}(s) + \sum_{z \in \mathcal{S}(z_s)} r^*_{z_s z}(s) \pi^*_{z z_e}(s). \tag{2}$$

Here $\overline{F}^*_{z_s}(s)$, the Laplace transform of the survival function of the sojourn time in state z_s, can be calculated as [6]

$$\overline{F}^*_{z_s}(s) = \frac{1 - \sum_{z \in \mathcal{S}(z_s)} r^*_{z_s z}(s)}{s},$$

while $r^*_{z_1 z_2}(s)$ represents the Laplace transform of the kernel,

$$\begin{aligned} r^*_{z_1 z_2}(s) &= \int_0^\infty \exp(-st) dR(z_1, z_2, t) \\ &= \int_0^\infty \exp(-st) \cdot f_{z_1 z_2}(t) \cdot \prod_{\substack{z \in \mathcal{S}(z_1) \\ z \neq z_2}} (1 - F_{z_1 z}(t)) dt, \quad z_2 \in \mathcal{S}(z_1). \end{aligned}$$

Note that our specific semi-Markov model has the structure of a bivariate pure birth process. Therefore, the system of equations based on the general Equation (2) can be solved recursively for this model. Specifically, for any two states (i, j) and (i', j') we obtain from Equation (2):

$$\pi^*_{(i,j)(i',j')}(s) = \begin{cases} 0 & \text{if } i > i' \text{ or } j > j', \\ \overline{F}^*_{(i,j)}(s) & \text{if } i = i' \text{ and } j = j', \\ r^*_{(i,j)(i+1,j)}(s) \pi^*_{(i+1,j)(i',j)}(s) & \text{if } i < i' \text{ and } j = j', \\ r^*_{(i,j)(i,j+1)}(s) \pi^*_{(i,j+1)(i,j')}(s) & \text{if } i = i' \text{ and } j < j', \\ r^*_{(i,j)(i+1,j)}(s) \pi^*_{(i+1,j)(i',j')}(s) & \\ \quad + r^*_{(i,j)(i,j+1)}(s) \pi^*_{(i,j+1)(i',j')}(s) & \text{if } i < i' \text{ and } j < j'. \end{cases} \tag{3}$$

Evaluating this for any starting state $z_s = (i, j)$ and end state $z_e = (i', j')$, recursively plugging in the Laplace transforms of the transient state probabilities, yields the form

$$\pi^*_{z_s z_e}(s) = \overline{F}^*_{z_e}(s) \cdot \sum_{b \in \mathcal{P}(z_s,z_e)} \left(\prod_{u=1}^{|b|-1} r^*_{b(u)b(u+1)}(s) \right), \tag{4}$$

where $\mathcal{P}(z_s, z_e)$ denotes the set of all paths from state z_s to state z_e; each path is represented as a tuple containing the states along the path, including the start and end states. For a specific path b, $|b|$ is the length of the tuple, and $b(u)$, $u = 1, \ldots, |b|$, denotes the u^{th} state on the path.

As an example, consider the set of all paths from state $(0, 0)$ to state $(1, 1)$ in our model:

$$\mathcal{P}((0,0),(1,1)) = \{((0,0),(1,0),(1,1)),((0,0),(0,1),(1,1))\}.$$

The length of path $b = ((0, 0), (1, 0), (1, 1))$ is $|b| = 3$, and $b(2) = (1, 0)$ is the second state entered on this path. According to Equation (4), the Laplace transform of $\pi_{(0,0)(1,1)}(t)$ is thus given by

$$\begin{aligned}\pi^*_{(0,0)(1,1)}(s) = \overline{F}^*_{(1,1)}(s) \\ \times \left(r^*_{(0,0)(1,0)}(s) \cdot r^*_{(1,0)(1,1)}(s) + r^*_{(0,0)(0,1)}(s) \cdot r^*_{(0,1)(1,1)}(s)\right).\end{aligned}$$

From Equation (3), we can see that there are two types of Laplace transforms of the kernel, $r^*_{(i,j)(i,j+1)}(s)$ and $r^*_{(i,j)(i+1,j)}(s)$, respectively. The former one is related to the transition from a state (i, j) with $j < m$ to state $(i, j+1)$, i.e., the occurrence of an aging-related failure:

$$\begin{aligned}r^*_{(i,j)(i,j+1)}(s) &= \int_0^\infty \exp(-st) \cdot f_\Gamma(t; \alpha, \beta_j) \cdot (1 - F_{\exp}(t; \phi_i))dt \\ &= \left(\frac{\beta_j}{\beta_j + \phi_i + s}\right)^\alpha.\end{aligned}$$

The latter one is related to the transition from a state (i, j) with $i < n$ to state $(i + 1, j)$, i.e., a failure due to a non-aging-related bug:

$$\begin{aligned} r^*_{(i,j)(i+1,j)}(s) &= \int_0^\infty \exp(-st) \cdot f_{\exp}(t; \phi_i) \cdot (1 - F_\Gamma(t; \alpha, \beta_j))dt \\ &= \frac{\phi_i}{\phi_i + s} \cdot \left[1 - \left(\frac{\beta_j}{\beta_j + \phi_i + s}\right)^\alpha\right]. \end{aligned}$$

If $\alpha \in \mathbb{N}^+ \setminus \{1\}$ we could use partial fraction expansion for inverting the Laplace transform $\pi^*_{z_s z_e}(s)$ to obtain the transient probability $\pi_{z_s z_e}(t)$. However, in general numerical methods are needed. Using the statistical software `R` [32] we have implemented the fixed Talbot algorithm [1], which employs a deformation of the standard contour of the Bromwich integral to improve convergence.

Based on our model we are thus able to calculate the probability that testing a software initially containing m aging-related and n non-aging-related bugs for t_e time units will lead to the detection of j aging-related and i non-aging-related bugs as the transient state probability $\pi_{(0,0)(i,j)}(t_e)$. Deriving this probability for each combination (i, j) with $i = 0, \ldots, n$ and $j = 0, \ldots, m$ results in the joint probability mass function (pmf) of the number of non-aging-related bugs detected and the number of aging-related bugs detected by time t_e.

In Section 2.4, we will use this joint pmf to carry out a model-based study of the testing phase that will for example indicate the existence of an interesting masking effect. The joint pmf will also form an important basis for our analysis of how testing affects the aging behavior in the operational phase (see Section 4). General insights can thus be gained by using realistic values of the model parameters, e.g., taken from the literature. However, a practitioner or a researcher may wish to estimate the model parameters for a specific software product from data collected during the (initial part of the) testing phase of this product, or during the entire testing phase of a similar kind of software.

Assume that the data has been observed while testing the software for t time units. If for each one of the k failures experienced before time t both the occurrence time t_l and the type of the underlying fault g_l (where $g_l = 1$ if the fault is an aging-related bug, and $g_l = 0$ otherwise) are known ($l = 1, 2, \ldots, k$), then the data set containing all relevant information can be represented as $d_t = \{(t_1, g_1), (t_2, g_2), \ldots, (t_k, g_k), t\}$. From this, we easily calculate the number of aging-related bugs and non-aging-related bugs removed after the l^{th} failure occurrence, $j_l \equiv \sum_{u=1}^{l} g_u$ and $i_l \equiv l - \sum_{u=1}^{l} g_u$, respectively. Moreover, the l^{th} time to failure is obtained as $x_l = t_l - t_{l-1}$.

Interpreting the joint probability density function of the random variables of which realizations have been observed as a function of the model parameters $\alpha, \beta, \phi, n, m$ *given* the data set d_t leads us to the likelihood function

$$\begin{aligned}
&\mathcal{L}(\alpha, \beta, \phi, n, m; d_t) \\
&= \prod_{l=1}^{k} \Bigg[g_l \cdot \frac{(\beta\,(m - j_{l-1}))^{\alpha}}{\Gamma(\alpha)} x_l^{\alpha-1} \exp\left(-\beta\,(m - j_{l-1})\,x_l\right) \cdot \exp\left(-\,(n - i_{l-1})\,\phi x_l\right) \\
&\qquad + (1 - g_l) \cdot \phi\,(n - i_{l-1}) \exp\left(-\phi\,(n - i_{l-1})\,x_l\right) \cdot \frac{\Gamma\left(\alpha, \beta\,(m - j_{l-1})\,x_l\right)}{\Gamma(\alpha)} \Bigg] \\
&\quad \times \exp\left(-\phi\,(n - i_k)\,(t - t_k)\right) \cdot \frac{\Gamma\left(\alpha, \beta\,(m - j_k)\,(t - t_k)\right)}{\Gamma(\alpha)}. \qquad (5)
\end{aligned}$$

This likelihood function consists of two distinct parts:

The first factor, itself a product, is related to the k failure occurrence times observed. For the l^{th} failure, the addend including g_l comes into effect if the failure was caused by an aging-related bug; it is the gamma pdf of the time to the $(j_{l-1}+1)^{st}$ failure occurrence due to an aging-related bug, evaluated at the actual time to failure x_l, multiplied with the probability that none of the $n-i_{l-1}$ non-aging-related faults remaining in the software caused a failure in these x_l time units. Similarly, the addend containing $(1-g_l)$, connected with a failure due to a non-aging-related bug, is the exponential pdf of the time to the $(i_{l-1}+1)^{st}$ failure of this type evaluated at x_j and multiplied with the probability of no aging-related failure occurrence in the interval $(t_{l-1}, t_{l-1}+x_l] = (t_{l-1}, t_l]$. It is thus crucial to note that each failure occurrence provides us with two pieces of information: the fact that a bug of a specific type has led to a failure after a certain amount of time, and the fact that bugs of the other type have not caused any failure since the previous failure occurrence. All this information needs to be reflected in the likelihood function to make full use of the data.

The second factor in Equation (5) is connected with the time between the last failure occurrence and the end of the observation period. It represents the probability that neither aging-related nor non-aging-related bugs cause a failure in these $t-t_k$ time units.

The higher the value of the likelihood function for a given combination of values for $\alpha, \beta, \phi, n, m$, the higher the possibility that the model with these parameter values has generated the data actually collected. Maximizing the likelihood function (or its logarithm) with respect to the parameters results in the maximum likelihood estimates $\hat{\alpha}, \hat{\beta}, \hat{\phi}, \hat{n}$, and $\hat{m}$.

2.3. Testing phases with more than one testing approach

So far, we have assumed a homogeneous testing phase with constant parameters α, β and ϕ. However, it is possible that in an overall testing phase of duration t_e various testing approaches (like white-box, black-box, and operational testing) are consecutively used. Such approaches differ in their ability to detect aging-related and non-aging-related bugs.

In fact, such a scenario is easily examined based on our previous results. For example, if during a testing phase testing approach #I (with corresponding parameter values ϕ^I, β^I and α^I) is employed for the first t_W time units, while the software is then tested according to testing approach #II (with corresponding parameter values ϕ^{II}, β^{II} and α^{II}) for the remaining duration $t_e - t_W$, then the probability that i non-aging-related bugs and j aging-related bugs will have been detected at the end of testing can be calculated as

$$\pi_{(0,0)(i,j)}(t_W, t_e) = \sum_{i' \leq i} \sum_{j' \leq j} \pi^I_{(0,0)(i',j')}(t_W) \cdot \pi^{II}_{(i',j')(i,j)}(t_e - t_W), \tag{6}$$

where $\pi^I_{(0,0)(i',j')}(t_W)$ and $\pi^{II}_{(i',j')(i,j)}(t_e - t_W)$ denote transient state probabilities computed based on the first and second set of parameters, respectively, using the method presented in the last section.

It is straightforward to extend Equation (6) for a case in which more than two testing approaches are consecutively used. Moreover, the likelihood function presented at the end of the last section can easily be adapted for data collected in testing phases with multiple approaches employed.

2.4. Numerical examples

Assume that at the beginning of the testing phase a software contains $n = 200$ non-aging-related bugs and $m = 22$ aging-related bugs which can lead to crash failures. The software is tested using testing approach #I that implies the rates $\phi^I = \beta^I = 0.004$ $[h^{-1}]$ for both fault types, and the parameter $\alpha^I = 5.2$. What are the likely outcomes of a testing phase lasting $t_e = 750$ hours?

The bivariate pmf of the number of non-aging-related bugs removed and the number of aging-related bugs removed calculated like described in Section 2.2 is depicted as a bubble chart in Figure 2; i.e., the area of each bubble represents the value of the respective probability mass. From this pmf, we can derive metrics like the expectations and the variances of the marginal distributions. Here the number of non-aging-related bugs removed has an expected value of 190.043 and a variance of 9.462; the expectation and variance of the number of aging-related bugs removed after $t_e = 750$ hours are 1.036 and 0.796, respectively.

How the expected value of the (cumulative) number of aging-related bugs removed develops with test duration t_e is shown by the solid line in Figure 3. Obviously, none of the aging-related bugs tend to get detected during the first 400 hours of testing. The reason for this phenomenon lies in the non-aging-related bugs: As long as many of them are still left in the software, crashes happen rather frequently, which prevents aging effects to accumulate sufficiently for an aging-related failure to occur. Only later, after a large number of non-aging-related bugs has been found and removed, are the aging-related bugs starting to show. The extent to which software aging is masked by the non-aging-related bugs can also been seen from Figure 3: The dashed line indicates how the expected number of aging-related bugs removed develops in the absence of any non-aging-related bugs (i.e., if $n = 0$, all other param-

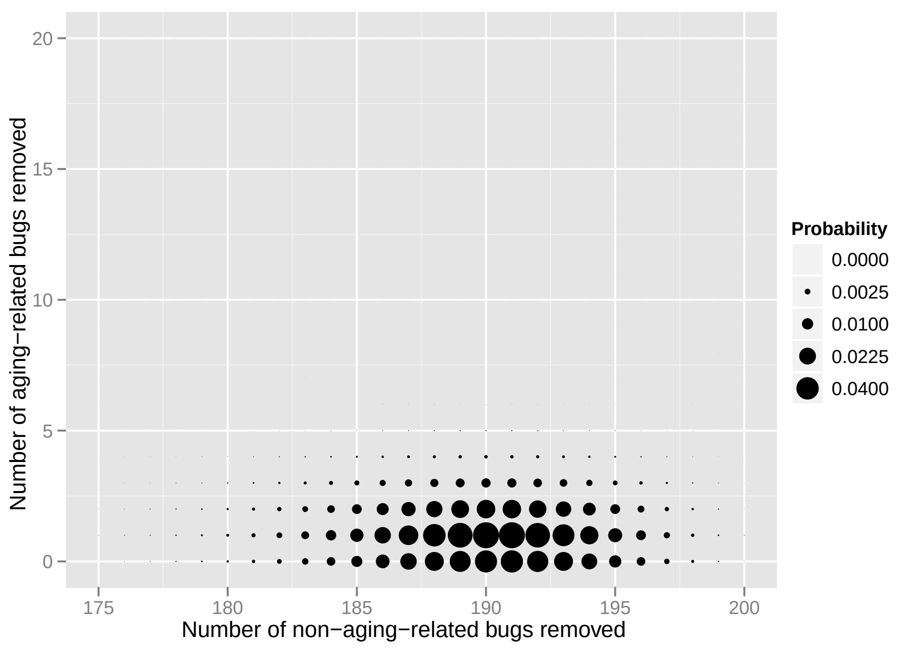


Figure 2: Bivariate pmf after a test duration of $t_e = 750$ $[h]$

eter values remaining unchanged). This example illustrates that we cannot study the two fault types separately when examining the effects of testing.

Let us now assume that the software is first tested using approach #I for $t_W = 650$ hours, while the testing approach #II employed for the remaining $t_e - t_W = 100$ hours of the testing phase focuses on uncovering aging-related bugs. For example, if it is known that certain workload parameters have been especially influential for software aging in similar software or in previous versions of the software under test [27], then test cases could frequently use the relevant levels of these workload parameters, aiming at increasing both the activation rate of the aging-related bugs as well as the aging effect caused per activation. In terms of our model parameters this is likely to lead to a larger value of β and a lower value of α. However, putting much

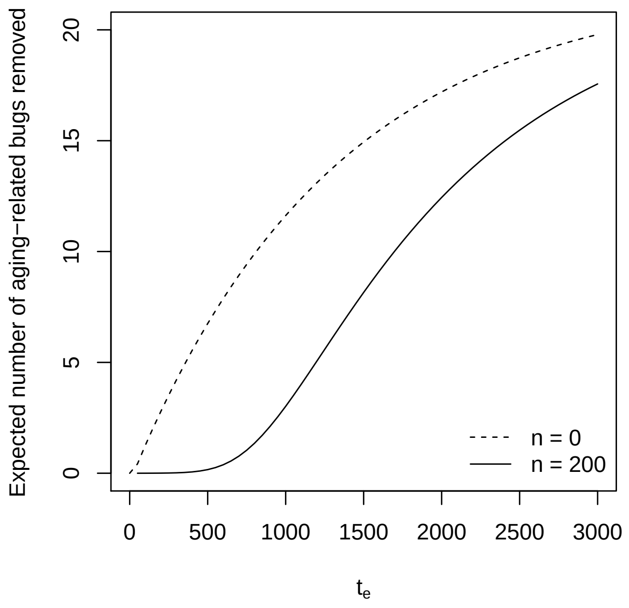


Figure 3: Expected number of aging-related bugs removed as a function of the test duration

emphasis on constellations potentially contributing to software aging may impede the rate at which non-aging-related bugs are activated. We therefore use the parameter values $\alpha^{II} = 5.2/3 = 1.733$, $\beta^{II} = 0.004 \cdot 3 = 0.012$ and $\phi^{II} = 0.004/2 = 0.002$ for modeling testing under approach #II.

Figure 4 shows the bivariate pmf of the number of non-aging-related and aging-related bugs removed at the end of the overall testing phase, using the same scale as in Figure 2. Obviously and not surprisingly, when 100 hours are allocated from testing approach #I to approach #II the bivariate pmf is shifted; it is now more likely that a larger number of aging-related bugs will be detected during testing (expected value: 10.801), while less non-aging-related bugs tend to get found (expected value: 187.838). From Figure 4 we can also see that the probability mass is spread over a larger region;

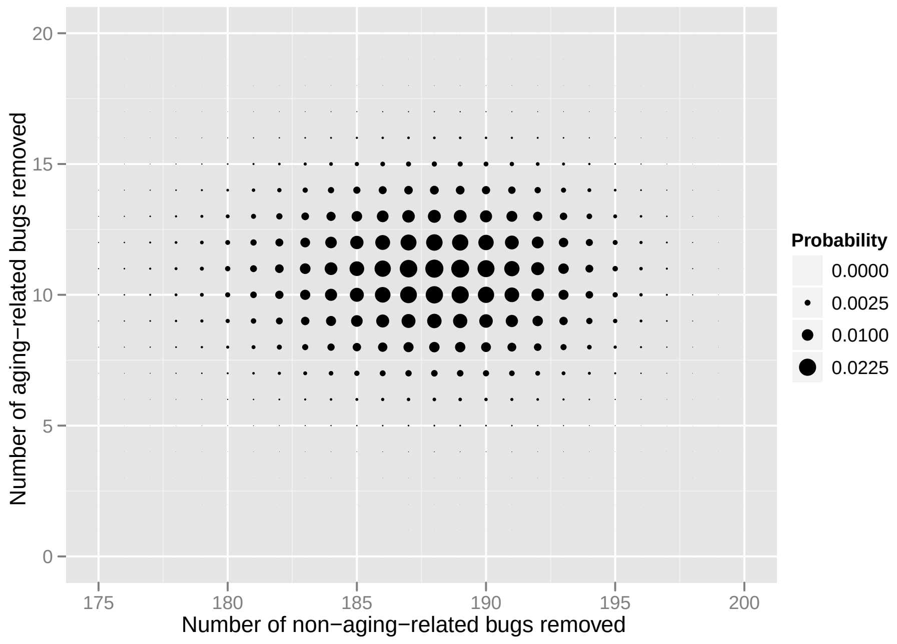


Figure 4: Bivariate pmf after a test duration of $t_e = 750$ $[h]$, switching time $t_W = 650$ $[h]$

the uncertainty about the outcome, especially with respect to the number of aging-related bugs removed, has increased.

3. Modeling the system behavior in the operational phase

3.1. Model formulation

We extend the assumptions made by classical models for the operational behavior of aging software systems [11, 20, 37] to explicitly account for crashes caused by aging-related and non-aging-related bugs:

A5 After the testing phase, the software still contains $\tilde{m}$ aging-related bugs and $\tilde{n}$ non-aging-related bugs that can lead to crash failures.

A6 From the up state S_U, the software system switches into the crash state

S_C due to crash failures caused by aging-related or non-aging-related bugs.

A7 The time to failure due to aging-related bugs follows a gamma distribution with shape $\tilde{\alpha} > 1$ and rate $\tilde{m}\tilde{\beta}$.

A8 The time to failure due to non-aging-related bugs follows an exponential distribution with rate $\tilde{n}\tilde{\phi}$, where $\tilde{\phi}$ is the per-fault hazard rate of a single non-aging-related bug.

A9 The time to recover the system upon a crash follows a general distribution with finite mean h_{S_C}.

A10 While the software system is in state S_U, the user or a software agent triggers software rejuvenation at constant rejuvenation rate τ. The system then switches into the rejuvenation state S_R.

A11 The time to carry out software rejuvenation follows a general distribution with finite mean h_{S_R}. Since software rejuvenation is triggered deliberately, usually $h_{S_R} < h_{S_C}$.

We are using the tilde in the parameters $\tilde{\alpha}, \tilde{\beta}$ and $\tilde{\phi}$ to point out that their values are typically different from the ones of the corresponding parameters in the model for the testing phase.

According to assumptions A5–A11, the stochastic process $\{\tilde{Z}(t), t \geq 0\}$ modeling the development of the system state during the operational phase follows a semi-Markov model with state space $\tilde{\mathcal{Z}} = \{S_U, S_C, S_R\}$. It is depicted in Figure 5. In this diagram, $F_{\text{1OEG}}(x; \tilde{n}\tilde{\phi}, \tilde{\alpha}, \tilde{m}\tilde{\beta})$ denotes the cdf of

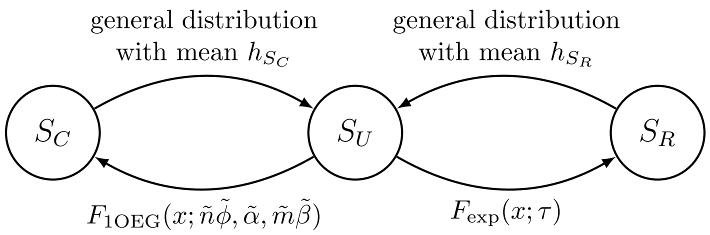

general distribution with mean h_{S_C}
general distribution with mean h_{S_R}
S_C
S_U
S_R
$F_{1\mathrm{OEG}}(x; \tilde{n}\tilde{\phi}, \tilde{\alpha}, \tilde{m}\tilde{\beta})$
$F_{\exp}(x;\tau)$

Figure 5: Semi-Markov model for the system behavior in the operational phase

the first order statistic of an exponential distribution with parameter $\tilde{n}\tilde{\phi}$ and a gamma distribution with parameters $\tilde{\alpha}$ and $\tilde{m}\tilde{\beta}$; i.e.,

$$\begin{aligned} F_{1\mathrm{OEG}}(x; \tilde{n}\tilde{\phi}, \tilde{\alpha}, \tilde{m}\tilde{\beta}) &= 1 - (1 - F_{\exp}(x; \tilde{n}\tilde{\phi}))(1 - F_{\Gamma}(x; \tilde{\alpha}, \tilde{m}\tilde{\beta})) \\ &= 1 - \exp(-\tilde{n}\tilde{\phi}x) \cdot \frac{\Gamma(\tilde{\alpha}, \tilde{m}\tilde{\beta}x)}{\Gamma(\tilde{\alpha})}. \end{aligned}$$

The corresponding pdf is given by

$$\begin{aligned} f_{1\mathrm{OEG}}(x; \tilde{n}\tilde{\phi}, \tilde{\alpha}, \tilde{m}\tilde{\beta}) = \tilde{n}\tilde{\phi} \cdot \exp(-\tilde{n}\tilde{\phi}x) \cdot \frac{\Gamma(\tilde{\alpha}, \tilde{m}\tilde{\beta}x)}{\Gamma(\tilde{\alpha})} \\ + \exp(-\tilde{n}\tilde{\phi}x) \cdot \frac{(\tilde{m}\tilde{\beta})^{\tilde{\alpha}} \cdot \exp(-\tilde{m}\tilde{\beta}x) \cdot x^{\tilde{\alpha}-1}}{\Gamma(\tilde{\alpha})}. \end{aligned}$$

3.2. Deriving steady-state availability

An important metric for evaluating system performance is the steady-state availability A, which is identical to the steady-state probability that the system is in the up state S_U.

Let T_i denote the time at which the i^{th} transition takes place. Since the times $T_1, T_2, \ldots$, constitute renewal points, the stochastic process $\{\tilde{Z}(T_i+), i = 1, 2, \ldots\}$ is an embedded discrete-time Markov chain (DTMC) of the semi-Markov process studied. For three-state operational models with the same structure (but different transition distributions), Garg et al. [11] and Vaidyanathan and Trivedi [37] have shown that the steady-state probabilities in their embedded DTMCs are given by $\tilde{\pi}^d_{S_U} = 1/2$, $\tilde{\pi}^d_{S_C} = p_{S_U S_C}/2$, and $\tilde{\pi}^d_{S_R} = (1 - p_{S_U S_C})/2$, where $p_{S_U S_C}$ represents the transition probability from state S_U to state S_C in the embedded DTMC. Based on our model assumptions, we have

$$\begin{aligned} p_{S_U S_C} &= \int_0^\infty f_{1\text{OEG}}(x; \tilde{n}\tilde{\phi}, \tilde{\alpha}, \tilde{m}\tilde{\beta}) \cdot (1 - F_{\exp}(x;\tau))dx \\ &= \frac{\tilde{n}\tilde{\phi}}{\tilde{n}\tilde{\phi} + \tau} + \frac{\tau}{\tilde{n}\tilde{\phi} + \tau} \cdot \left(\frac{\tilde{m}\tilde{\beta}}{\tilde{m}\tilde{\beta} + \tilde{n}\tilde{\phi} + \tau} \right)^{\tilde{\alpha}}. \end{aligned}$$

To calculate steady-state probabilities related to the semi-Markov model itself, we additionally require the expected sojourn time h_z in each of the system states z. For the states S_R and S_C, the mean sojourn times are the means h_{S_R} and h_{S_C} mentioned in assumptions A9 and A11. For state S_U the cdf of the sojourn time is given by

$$\begin{aligned} H_{S_U}(x) &= 1 - (1 - F_{\exp}(x;\tau)) \cdot \left(1 - F_{1\text{OEG}}(x; \tilde{n}\tilde{\phi}, \tilde{\alpha}, \tilde{m}\tilde{\beta})\right) \\ &= 1 - \exp(-(\tau + \tilde{n}\tilde{\phi})x) \cdot \frac{\Gamma(\tilde{\alpha}, \tilde{m}\tilde{\beta}x)}{\Gamma(\tilde{\alpha})}, \end{aligned}$$

and its expected value amounts to

$$h_{S_U} = \int_0^\infty (1 - H_{S_U}(x))dx = \frac{1}{\tilde{n}\tilde{\phi} + \tau} \cdot \left[1 - \left(\frac{\tilde{m}\tilde{\beta}}{\tilde{m}\tilde{\beta} + \tilde{n}\tilde{\phi} + \tau} \right)^{\tilde{\alpha}} \right].$$

The steady-state availability is thus obtained as [36, p. 472]

$$
\begin{aligned}
A &= \frac{\tilde{\pi}^d_{S_U} \cdot h_{S_U}}{\tilde{\pi}^d_{S_U} \cdot h_{S_U} + \tilde{\pi}^d_{S_R} \cdot h_{S_R} + \tilde{\pi}^d_{S_C} \cdot h_{S_C}} \\
&= \frac{1 - \left(\frac{\tilde{m}\tilde{\beta}}{\tilde{m}\tilde{\beta}+\tilde{n}\tilde{\phi}+\tau}\right)^{\tilde{\alpha}}}{1 - \left(\frac{\tilde{m}\tilde{\beta}}{\tilde{m}\tilde{\beta}+\tilde{n}\tilde{\phi}+\tau}\right)^{\tilde{\alpha}} + h_{S_C}\tilde{n}\tilde{\phi} + h_{S_R}\tau + (h_{S_C} - h_{S_R})\tau\left(\frac{\tilde{m}\tilde{\beta}}{\tilde{m}\tilde{\beta}+\tilde{n}\tilde{\phi}+\tau}\right)^{\tilde{\alpha}}}. \qquad (7)
\end{aligned}
$$

Based on this equation, we can examine how changing the rate with which software rejuvenation is triggered affects the (steady-state) availability. (Since we do not consider transient availabilities in this paper, we will in the following use the terms “steady-state availability” and “availability” interchangeably.) Determining the optimal rejuvenation rate (i.e., the rejuvenation rate for which the availability is maximized), denoted by τ', and the maximum availability attained under this rejuvenation rate, denoted by A', will generally require numerical methods.

However, if the aging behavior is characterized with shape parameter $\tilde{\alpha} = 2$, which entails a two-stage Erlang distribution like in the work by Vaidyanathan and Trivedi [37], then it is easily seen from Equation (7), by differentiating with respect to τ and setting this to zero, that

$$\tau' = \max\left\{0, \tilde{m}\tilde{\beta} \cdot \left(\sqrt{\frac{h_{S_C}}{h_{S_R}}} - 2\right) - \tilde{n}\tilde{\phi}\right\}. \qquad (8)$$

If the second value in the above set is smaller than or equal to zero, software rejuvenation should never be triggered.

3.3. Numerical examples

We first take a look at the real-world example of Apache 2.0, initially released in a non-alpha version in November 2001. It can be assumed that many of the original software faults have meanwhile been reported. For our availability analysis we are interested in those faults that can make the software crash. Similar to Chandra and Chen [3] we query the Apache Software Foundation bug system at `https://issues.apache.org/` for problem reports containing the keywords "crash", "segmentation", "race" and "died"; unlike Chandra and Chen, we also include inflections of these words ("crashed", "crashes", "races", "dies") as well as additional words and their inflections ("hang", "hangs", "hanged", "segfault", "segfaults"). From the resulting list of 39 problem reports we drop one which has been caused by an operator error and one turning out to be a question on how to use Apache rather than the description of a failure occurrence. Moreover, 15 reports concern faults located not in Apache but in the hardware or in some other software like the operating system Gentoo Linux, the NSS library, OpenSSL or, most frequently, PHP. In contrast to other studies, e.g., the one by Cotroneo et al. [5], we wish to classify software faults, not failures. We therefore omit two further problem reports marked as duplicates and related to the same faults as other problem reports included in our list. For each one of the 20 unique faults in Apache we then try to determine whether or not it is an aging-related bug by reading the detailed description of the problem and its analysis. Whereas Cotroneo et al. [5] deduce the existence of aging-related bugs from the frequency with which failures occur, we are looking for indications that the failure rate of a fault increases while the software is continuously running. We find such indications for $\tilde{m} = 4$ faults and classify them as aging-related bugs. While the descriptions of three faults contain evidence concerning the presence of aging effects like progressive memory exhaustion or the accumulation of error messages finally causing a failure, the analysis of the fourth fault suggests that it is activated after a certain number of hits has been received by the web server since its last restart. The $\tilde{n} = 16$ other faults are classified as non-aging-related bugs.

Assume that the faults causing crash failures contained in the initial release of Apache 2.0 were the 20 faults analyzed, that the mean time to rejuvenate the Apache web server is $h_{S_R} = 0.5$ $[h]$ and that recovering the system after a crash takes $h_{S_C} = 2.75$ $[h]$ on average. (These are the parameter values used in the simulation study by Salfner and Wolter [33].) If the time until software aging causes Apache to crash follows a two-stage Erlang distribution, then Equation (8) implies that with respect to availability implementing software rejuvenation in Apache 2.0 would only have been beneficial if

$$\tilde{\beta} > \frac{\tilde{n}}{\tilde{m} \cdot \left(\sqrt{\frac{h_{S_C}}{h_{S_R}}} - 2\right)} \cdot \tilde{\phi} = 11.587 \cdot \tilde{\phi}.$$

It is thus possible that software rejuvenation may not have helped to increase the availability of Apache 2.0, but a final verdict can only be made after a more detailed investigation of the failure behavior and the rejuvenation and recovery times.

However, exclusively focusing on software aging while omitting the effects of non-aging-related bugs we come to the conclusion that software rejuvenation *is* beneficial *regardless* of the rates $\tilde{\beta} > 0$ and $\tilde{\phi}$. Setting $\tilde{n}$ equal to zero, according to Equation (8) the optimal rejuvenation rate is larger than zero if

$$\sqrt{\frac{h_{S_C}}{h_{S_R}}} > 2,$$

which indeed holds for our parameter values; software rejuvenation should thus be employed. This analysis neglects the fact that recovery carried out after failures caused by non-aging-related bugs removes the internal error states accumulated, which reduces the need for triggering software rejuvenation. Therefore, scheduling software rejuvenation based on the aging behavior alone can result in suboptimal decisions.

We now turn again to our more hypothetical example. Assume that at the beginning of the operational phase the software system discussed in Section 2.4 still contains $\tilde{n} = 2$ non-aging-related and $\tilde{m} = 20$ aging-related bugs. During operations, software is normally used in a less stressful way than in the testing phase, and faults therefore tend to cause failures at a lower rate [30]. We account for this via the parameter values $\tilde{\phi} = \tilde{\beta} = 0.0023$ $[h^{-1}]$ and $\tilde{\alpha} = 10.4$. The mean times to carry out rejuvenation and recovery from a crash are kept at $h_{S_R} = 0.5$ $[h]$ and $h_{S_C} = 2.75$ $[h]$, respectively.

Based on Equation (7), we calculate the availability A attained as a function of the rejuvenation rate τ. Figure 6 shows that there is a unique optimum at the rate $\tau' = 0.0047$ $[h^{-1}]$, corresponding to a mean time until rejuvenation is triggered of 213 hours (or 8.9 days). The maximum availability achieved at this rate is $A' = 0.9807$.

4. Combining the models for the testing and the operational phase

4.1. Theoretical considerations

Both our model for the testing phase and our model for the operational phase explicitly account for the number of aging-related and non-aging-related bugs. This allows us to combine them in order to study the effects of testing on the system behavior during usage.

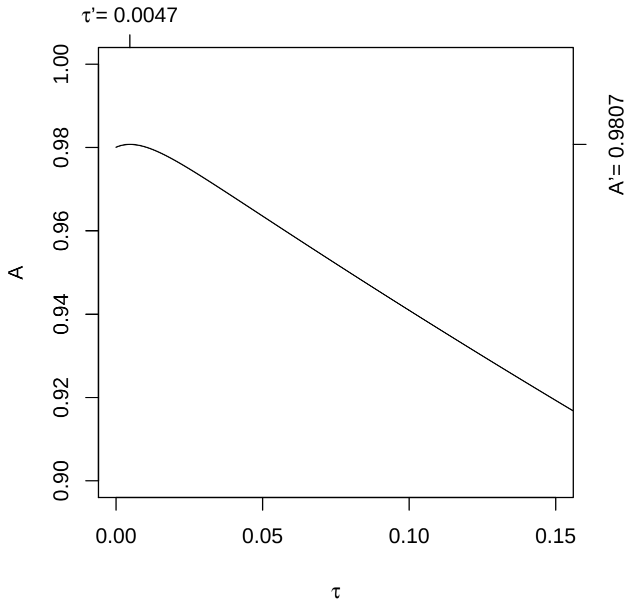


Figure 6: Availability attained as a function of the rejuvenation rate

For a specific number of non-aging-related bugs $\tilde{n}$ and a specific number of aging-related bugs $\tilde{m}$ remaining in the software we can derive the best rejuvenation rate τ' and the availability A' attained at this rate with the model developed in Section 3. To stress that the optimization depends on $\tilde{n}$ and $\tilde{m}$, we will now denote these results by $\tau'(\tilde{n}, \tilde{m})$ and $A'(\tilde{n}, \tilde{m})$. When we are trying to evaluate the potential outcomes of a testing strategy at the beginning of the testing phase, then the number of non-aging-related and aging-related bugs that will not be detected during testing are random variables, $\tilde{N}$ and $\tilde{M}$. The joint pmf of these two random variables can be computed using the results of Section 2. For example, for a homogeneous testing phase lasting t_e time units, the probability that at the end of testing $\tilde{N}$ and $\tilde{M}$ will take the values $\tilde{n}$ and $\tilde{m}$ is given by the transient state probability $\pi_{(0,0)(n-\tilde{n},m-\tilde{m})}(t_e)$. As functions of the random variables $\tilde{N}$ and $\tilde{M}$, the optimal rejuvenation rate $\tau'(\tilde{N},\tilde{M})$ and the maximum availability attainable $A'(\tilde{N},\tilde{M})$ are random variables as well. We thus need to discuss the effects of testing in terms of the distributions of these random variables, and metrics like their expectations.

The distribution of the optimal rejuvenation rate is obtained as follows: For each pair of possible outcomes $\tilde{n}$ and $\tilde{m}$, calculate $\tau'(\tilde{n},\tilde{m})$ and assign to this result the probability mass $\pi_{(0,0)(n-\tilde{n},m-\tilde{m})}(t_e)$; sorting the results in ascending order (aggregating for example multiple optimal rejuvenation rates of zero into a single probability mass) leads to the pmf. By successively cumulating the probabilities, the cdf is obtained. We have automated this procedure, again using the statistical software `R` [32]. Based on the same kind of approach we also derive the pmf and cdf of the maximum availability attainable.

Similarly, the expectations of these random variables can be calculated by weighting each result $\tau'(\tilde{n},\tilde{m})$ or $A'(\tilde{n},\tilde{m})$ with the respective probability mass $\pi_{(0,0)(n-\tilde{n},m-\tilde{m})}(t_e)$:

$$E[\tau'(t_e)] = \sum_{\tilde{n}=0}^{n}\sum_{\tilde{m}=0}^{m} \tau'(\tilde{n},\tilde{m}) \cdot \pi_{(0,0)(n-\tilde{n},m-\tilde{m})}(t_e), \quad \text{and}$$

$$E[A'(t_e)] = \sum_{\tilde{n}=0}^{n}\sum_{\tilde{m}=0}^{m} A'(\tilde{n},\tilde{m}) \cdot \pi_{(0,0)(n-\tilde{n},m-\tilde{m})}(t_e).$$

Note that both expected values depend on the testing duration t_e. This allows us for example to examine how varying the length of the testing period affects the expected maximum availability achievable during operations.

Of course, due to fixed release dates or funding constraints software developing companies cannot arbitrarily extend the testing period. Rather, they have to solve the problem of allocating their limited time budget to various testing approaches such that the quality of the software will be as good as possible under the given circumstances.

In Section 2.3 we have seen how to compute the transient state probabilities at the end of a testing phase in which several testing approaches are consecutively used. While we will again focus on a scenario with two testing approaches, an extension to a larger number of approaches can easily be made. Each transient state probability $\pi_{(0,0)(i,j)}(t_W, t_e)$ calculated via Equation (6) is a function not only of the overall test duration t_e, but also of the time t_W when testing switches from the first approach to the second one. Therefore, the distributions of the optimal rejuvenation rate τ' and the maximum availability attainable A' depend on t_W as well. For a given test duration t_e we can thus for example analyze how the expected value of the latter distribution, $E[A'(t_W, t_e)]$, develops as t_W varies between zero and t_e; while $t_W = 0$ means that the second testing approach is used throughout the entire testing phase, in the limiting case $t_W = t_e$ only the first testing approach is employed. The optimal switching time is the one resulting in the highest expected maximum availability attainable.

4.2. Numerical examples

Consider again the example analyzed in Section 2.4 of a software initially containing $n = 200$ non-aging-related and $m = 22$ aging-related bugs, tested with approach #I implying the parameters $\alpha^I = 5.2$ and $\beta^I = \phi^I = 0.004$ $[h^{-1}]$. We now examine the effect of different testing durations t_e on the aging behavior in the operational phase, for which we assume the parameter values $\tilde{\alpha} = 10.4$ and $\tilde{\phi} = \tilde{\beta} = 0.0023$ $[h^{-1}]$ already used in Section 3.3. Note

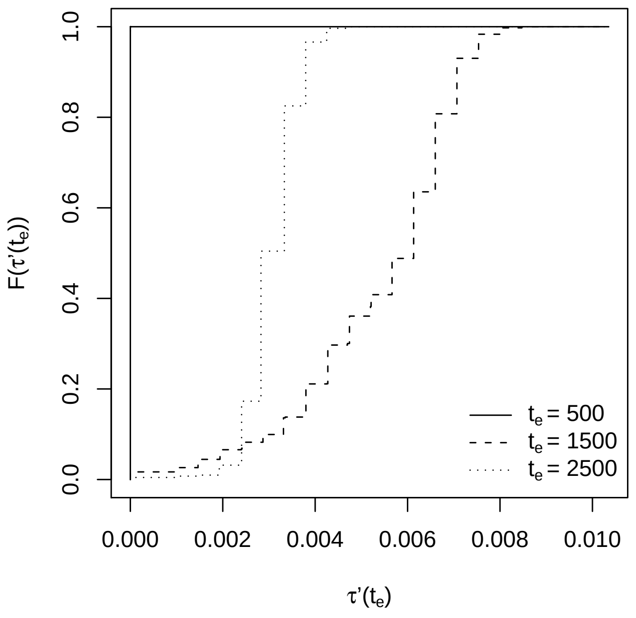


Figure 7: Cdf of the optimal rejuvenation rate for different testing durations

that $\beta^I/\tilde{\beta} = \phi^I/\tilde{\phi}$ can be interpreted as the testing compression factor (TCF) of testing approach #I, i.e., the ratio of the execution times needed in the operational phase and the testing phase using this approach, respectively, to cover the entire input domain of the software [30, p. 178]. In fact, we chose the parameter values $\tilde{\beta}$ and $\tilde{\phi}$ based on those for testing approach #I as β^I/TCF and ϕ^I/TCF, making use of the TCF $= 1.72$ determined by Huang and Lin [19] in their analysis of an electronic switching software system.

Figure 7 shows the cdf of the optimal rejuvenation rate τ' for three different testing durations.

At a testing duration of $t_e = 500$ $[h]$ the entire probability mass of the distribution is located very close to zero; the expected value of τ' amounts to $1.35 \cdot 10^{-11}$ $[h^{-1}]$. For all practical purposes, software rejuvenation should never be triggered. The reason for this result is the fact that with high probability many non-aging-related bugs will remain in the software after such a short testing phase, frequently causing crashes during operations. Therefore, aging effects cannot build up sufficiently for creating a high risk of aging-related failures. Software rejuvenation thus has hardly any positive effects, but it mainly causes the system to be unavailable until rejuvenation has been completed.

However, after testing the software for $t_e = 1500$ $[h]$ it is likely that many of the non-aging-related bugs have been removed. It is thus indeed beneficial to regularly rejuvenate the system in the operational phase in order to prevent failures due to the aging-related bugs remaining. The expectation of the optimal rejuvenation rate is 0.00538 $[h^{-1}]$, which means that the running system should on average be rejuvenated every 7.7 days.

If testing is to last $t_e = 2500$ $[h]$, then not only many non-aging-related bugs but also many of the aging-related bugs will have been removed with high probability. Therefore, rejuvenation should again be triggered at a lower rate: The expected optimal rejuvenation rate is now 0.00306 $[h^{-1}]$, relating to a mean time of 13.6 days until rejuvenation is triggered in the up state. From Figure 7 we can also see that the probability mass is spread over a smaller range of values of τ' than in the case $t_e = 1500$, indicating a higher certainty about the outcome of the longer testing phase.

This example suggests that software rejuvenation should be triggered most frequently during the usage of "semi-tested" software. For both insufficiently-tested as well as highly mature software the optimal rejuvenation rate tends to be lower.

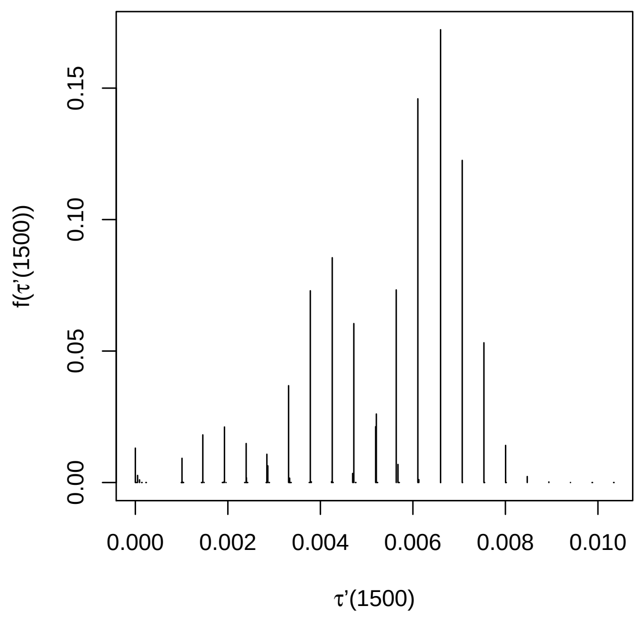


Figure 8: Pmf of the optimal rejuvenation rate for a test duration of $t_e = 1500$ $[h]$

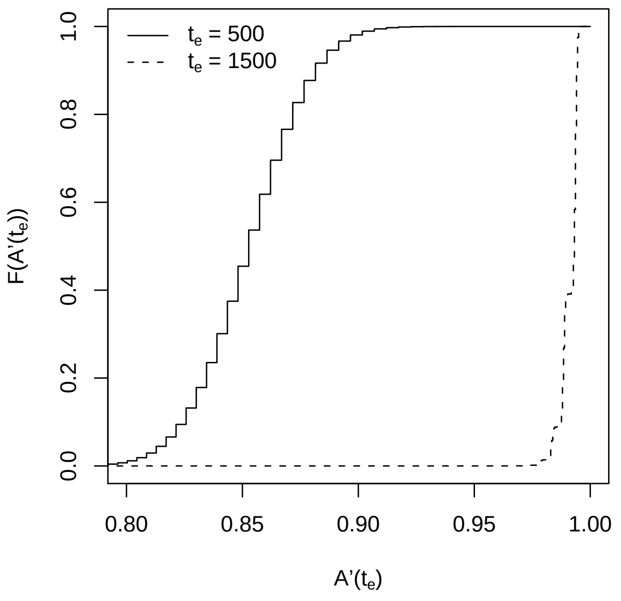
Unlike the cdfs of the optimal rejuvenation rate for various test durations, the related pmfs can hardly be plotted in the same diagram, because the bars tend to overlay. However, from the pmf representation the expected value of a random variable may be seen more easily, and it may also yield further insight.

As an example we show the pmf of the optimal rejuvenation rate for $t_e = 1500$ $[h]$ in Figure 8. For $\tau' > 0.0005$, the distribution mainly seems to be a mixture of three individual unimodal distributions with local maxima at around 0.0019, 0.0042, and 0.0065. In fact, these individual distributions are related to the events of two, one and zero non-aging-related bugs remaining in the software in the operational phase, respectively. Within each one of these distributions, the τ'-values with non-zero probabilities are not exactly evenly-spaced, but the distances between them often range between 0.00045 and 0.00050; starting with a given number of aging-related bugs remaining ($\tilde{m} > 0$) and non-aging-related bugs remaining ($0 \leq \tilde{n} \leq 2$), removing one more of the aging-related bugs tends to decrease the optimal rejuvenation rate by such an amount.

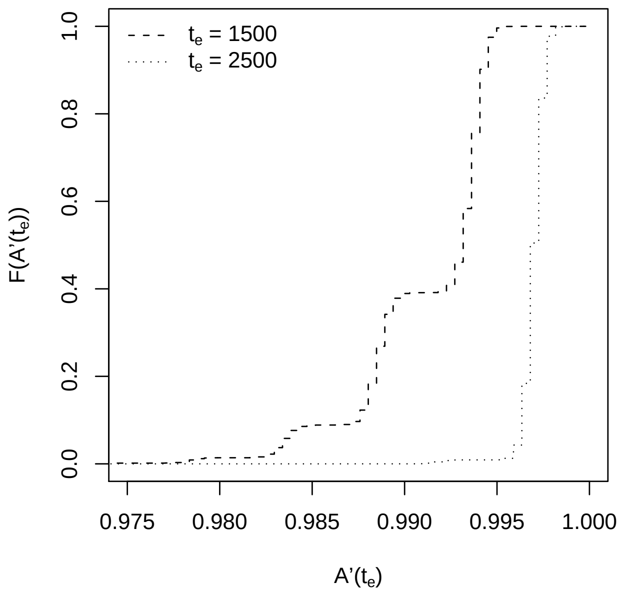
The cdfs of the maximum availability attainable for $t_e = 500, 1500$ and 2500 $[h]$ are depicted in Figure 9, separated into two plots due to the differences in magnitude between the three cases. Obviously, longer test durations are stochastically dominant with respect to the optimal availability: The longer the system has been tested, the higher the probability that a given target availability can be exceeded during operations.

This result is also reflected by the development of the expected maximum availability attainable $E[A'(t_e)]$ as a function of testing duration t_e, shown in Figure 10. The values $E[A'(500)] = 0.8530$, $E[A'(1500)] = 0.9911$ and $E[A'(2500)] = 0.9970$ in this plot are simply the expected values of the three distributions in Figure 9. Not surprisingly, the function is strictly increasing and converges to an availability of 100% for an infinite test duration. While the expected maximum availability attainable grows almost linearly for the first 500 hours, the slope of the function then quickly decreases, indicating a diminishing effect of each further hour spent on testing.

The fact that the parameters of the failure distributions are neither observed nor known with certainty, but estimated from experience or collected data, begs the question: How much are these results affected if the parameter values change? For example, what happens if the aging-related and non-aging-related bugs are more (less) virulent than expected? It seems rea-



(a) Testing durations 500 [h] and 1500 [h]



(b) Testing durations 1500 [h] and 2500 [h]

Figure 9: Cdf of the maximum availability attainable for different testing durations

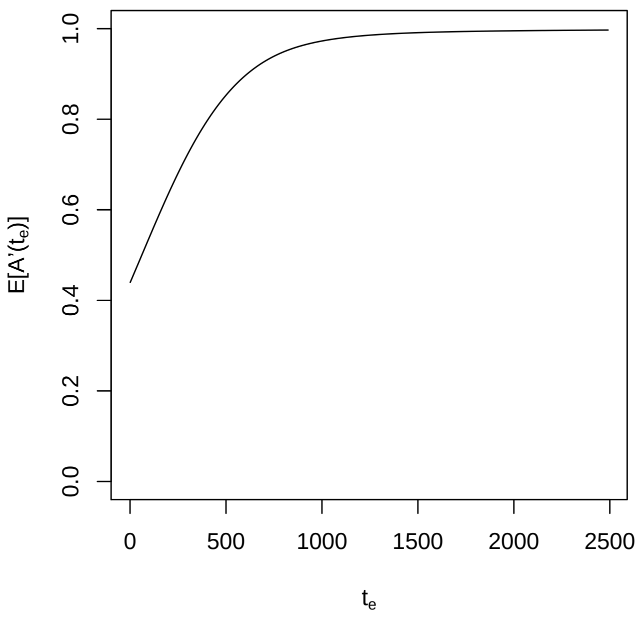


Figure 10: Expected maximum availability attainable as a function of testing duration

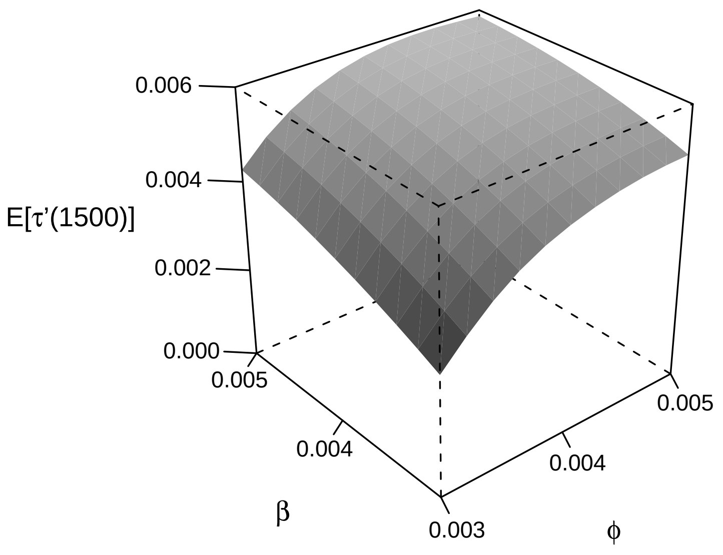


Figure 11: Expected optimal rejuvenation rate after $t_e = 1500$ hours of testing as a function of β and ϕ

sonable to assume that more (less) virulent faults tend to get activated at a higher (lower) rate during both the testing and the operational phase. We thus keep the testing compression factor constant at 1.72, and study by how much the expectations of the optimal rejuvenation rate and the maximum availability attainable change for a testing duration of $t_e = 1500$ $[h]$ as β and ϕ are varied within the interval $[0.003, 0.005]$, marking a decrease or increase of up to 25 percent from the original value 0.004.

Figure 11 shows the results for the expected optimal rejuvenation rate, which tends to exhibit positive relationships with the activation rates of both fault types in the region analyzed. As ϕ increases, more of the non-aging-related bugs are expected to be detected during testing; therefore, aging effects can build up to a higher degree during operations, making more frequent rejuvenation worthwhile. Keeping β fixed at 0.004 and varying ϕ within the given interval leads to values of $E[\tau'(1500)]$ between 0.00364 and 0.00560. While the most extreme decrease from the original expected optimal rejuvenation rate 0.00538 is slightly overproportional (-32.3%), the maximum increase is substantially underproportional at a mere $+4.1\%$.

For a fixed ϕ, within the interval $[0.003, 0.005]$ an increase in β leads to a higher expected optimal rejuvenation rate. Although the aging-related bugs become more virulent only relatively few additional ones tend to be found during testing due to the masking effect of the non-aging-related bugs; instead, this higher virulence means stronger aging effects in the operational phase, requiring more frequent rejuvenation. At $\phi = 0.004$, a 25-percent decrease or increase in β results in underproportional variations in $E[\tau'(1500)]$, which ranges from 0.00455 (-15.4%) to 0.00580 ($+7.8\%$).

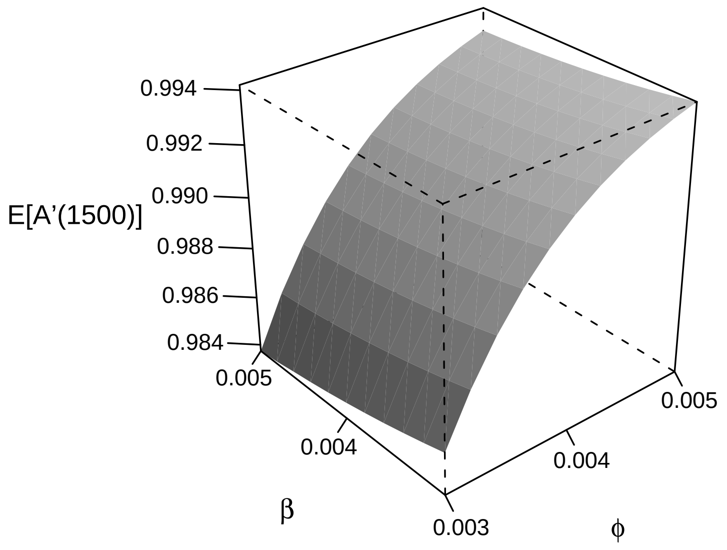


Figure 12: Expected maximum availability attainable after $t_e = 1500$ hours of testing as a function of β and ϕ

Figure 11 indicates that these effects may aggravate. If both β and ϕ happen to be 25 percent smaller than assumed, then the expected optimal rejuvenation rate amounts to 0.00269, which is 50 percent less than the value originally obtained. However, in the $[0.003, 0.005] \times [0.003, 0.005]$-region studied for β and ϕ the maximum resulting value of $E[\tau'(1500)]$ is 0.00592, corresponding to an increase of only 10.0%. Moreover, if the directions of deviation are different for the activation rates of the two fault types, then the effects tend to cancel.

From Figure 12 we can see that for the range of parameter values analyzed a higher virulence of the non-aging-related bugs leads to a higher expected maximum availability attainable (because more of these bugs tend to get removed during testing), whereas $E[A'(1500)]$ gets smaller as the virulence of

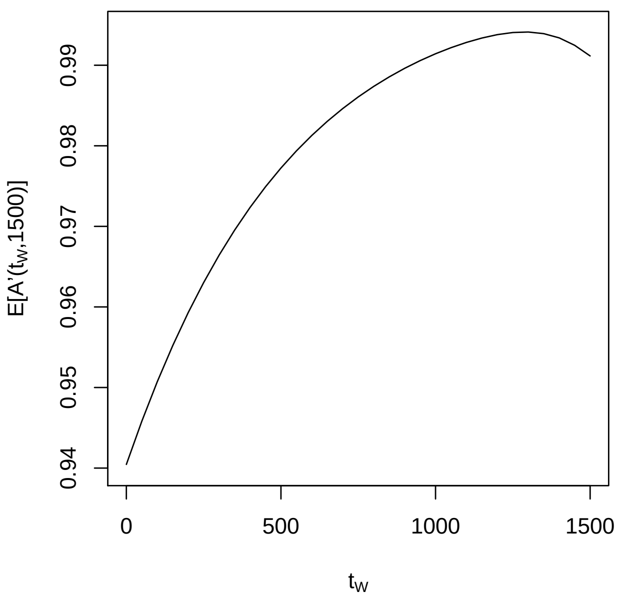


Figure 13: Expected maximum availability attainable as a function of switching time

the aging-related bugs increases (since the masking effect prevents detection of these bugs during testing from improving considerably, while the aging effects experienced during operations become more severe). In terms of the expected maximum availability attainable, the extent of variation due to changes in β and ϕ is less pronounced than the one for $E[\tau'(1500)]$: Within the region studied, the smallest $E[A'(1500)]$ obtained is 0.9837 (-0.7%), and the largest value amounts to 0.9942 ($+0.3\%$).

Let us now assume that there is a time budget of $t_e = 1500$ hours available for testing, which can be allocated to the two approaches already discussed in Section 2.4. When, if at all, should we ideally switch from approach #I to approach #II during the testing phase?

In Figure 13 it can be seen how the expected maximum availability attainable, $E[A'(t_W, 1500)]$, changes with the switching time over its entire possible range from $t_W = 0$ to $t_W = 1500$. The value $E[A'(1500, 1500)]$ relates to the expected outcome if solely testing approach #I is employed for the entire 1500 hours; from the above, we already know that $E[A'(1500, 1500)] = 0.9911$. Clearly, the function has a unique maximum, which is attained for a switching time of $t_W = 1287.2$ $[h]$: Using approach #I for 1287.2 hours instead of 1500 hours, and spending the remaining 212.8 hours on testing the software following approach #II increases the expected maximum availability attainable from 0.9911 to 0.9941.

5. Conclusions

In this paper we have developed semi-Markov models for the testing phase and the operational phase which (unlike existing models) explicitly account for aging-related bugs and non-aging-related bugs. From the assumptions of the operational model, we have derived a closed analytical expression for steady-state availability. For the testing model, we have formulated closed analytical expressions for the Laplace transforms of the transient state probabilities, and implemented an appropriate algorithm for numerical Laplace transform inversion. Based on these models we have detected a masking of software aging by non-aging-related failures for both the testing phase and the operational phase. While the existence of such a masking effect may be obvious (at least at hindsight), to the best of our knowledge it has never been discussed in the literature. What is more, we have seen that neglecting the masking effect will lead to wrong conclusions and suboptimal decisions.

We have also shown how the two models can be used in combination for examining the influence of testing on the aging behavior during usage. We do not know any previous work investigating such effects. In particular, we have seen that the analysis requires models for both phases that explicitly keep track of aging-related and non-aging-related bugs. Moreover, before the testing phase has ended the optimal rejuvenation rate and the maximum availability attainable in the operational phase are random variables; the analysis should thus be based on the distributions and moments of these random variables. The numerical example discussed by us suggests that software rejuvenation should be triggered most frequently for "semi-tested" systems rarely suffering from crashes due to non-aging-related bugs but still containing a moderate number of aging-related bugs.

While our models are based on some simplifying assumptions we are confident that the general insights obtained will not be affected if any of these assumptions is refined. In fact, our approach allows to easily make such adaptations due to its generic and modular nature.

We hope that our demonstration that non-aging-related bugs should not be neglected when studying software aging will induce practitioners and researchers to collect and publish data sets containing both failure times and the type of each underlying fault.

Acknowledgments

This research was supported by the Hans Frisch Foundation and by the Dr. Theo and Friedl Schoeller Research Center for Business and Society. We wish to thank the three anonymous reviewers, whose comments have helped us to improve this paper.

References

[1] Abate, J., Valkó, P. P., 2004. Multi-precision Laplace transform inversion. International Journal for Numerical Methods in Engineering 60 (5), 979–993.

[2] Carrozza, G., Cotroneo, D., Natella, R., Pecchia, A., Russo, S., 2010. Memory leak analysis of mission-critical middleware. Journal of Systems and Software 83 (9), 1556–1567.

[3] Chandra, S., Chen, P. M., 2000. Whither generic recovery from application faults? A fault study using open-source software. In: Proc. 2000 International Conference on Dependable Systems and Networks. pp. 97–106.

[4] Chen, Y., Singpurwalla, N. D., 1997. Unification of software reliability models by self-exciting point processes. Advances in Applied Probability 29 (2), 337–352.

[5] Cotroneo, D., Orlando, S., Russo, S., 2006. Failure classification and analysis of the Java Virtual Machine. In: Proc. 26th IEEE International Conference on Distributed Computing Systems.

[6] Dingle, N. J., 2004. Iterative transient state distribution calculation in semi-Markov processes. In: Proc. Third Workshop on Process Algebras and Stochastically Timed Activities. pp. 1–9.

[7] Dohi, T., Goševa-Popstojanova, K., Trivedi, K. S., 2000. Analysis of software cost models with rejuvenation. In: Proc. International Symposium on High Assurance Systems Engineering. pp. 25–34.

[8] Dohi, T., Goševa-Popstojanova, K., Trivedi, K. S., 2000. Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule. In: Proc. International Pacific Rim Symposium on Dependable Computing. pp. 77–84.

[9] Dohi, T., Goševa-Popstojanova, K., Trivedi, K. S., 2001. Estimating software rejuvenation schedules in high assurance systems. Computer Journal 44 (6), 473–482.

[10] Garg, S., Puliafito, A., Telek, M., Trivedi, K. S., 1995. Analysis of software rejuvenation using Markov regenerative stochastic Petri net. In: Proc. 6th International Symposium on Software Reliability Engineering. pp. 24–27.

[11] Garg, S., Puliafito, A., Telek, M., Trivedi, K. S., 2001. Analysis of preventive maintenance in transactions based processing systems. IEEE Trans. Computers 47 (1), 96–107.

[12] Gaudoin, O., 1990. Outils statistiques pour l'évaluation de la fiabilité des logiciels. Thèse de doctorat, Université de Joseph Fourier - Grenoble 1, Grenoble.

[13] Grottke, M., 2012. Prognose von Softwarezuverlässigkeit, Softwareversagensfällen und Softwarefehlern. In: Mertens, P., Rässler, S. (Eds.), Prognoserechnung, 7th Edition. Springer/Physica, Berlin, pp. 585–619.

[14] Grottke, M., Matias Jr., R., Trivedi, K. S., 2008. The fundamentals of software aging. In: Proc. 1st International Workshop on Software Aging and Rejuvenation.

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

[16] Grottke, M., Trivedi, K. S., 2005. A classification of software faults. In: Supplemental Proc. 16th International Symposium on Software Reliability Engineering. pp. 4.19–4.20.

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

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

[19] Huang, C.-Y., Lin, C.-T., 2010. Analysis of software reliability modeling considering testing compression factor and failure-to-fault relationship. IEEE Trans. Computers 59 (2), 283–288.

[20] Huang, Y., Kintala, C., Kolettis, N., Fulton, N., 1995. Software rejuvenation: Analysis, module and applications. In: Proc. 25th International Symposium on Fault-Tolerant Computing. pp. 381–390.

[21] Jelinski, Z., Moranda, P., 1972. Software reliability research. In: Freiberger, W. (Ed.), Statistical Computer Performance Evaluation. Academic Press, New York, pp. 465–484.

[22] Ledoux, J., 2003. Software reliability modeling. In: Pham, H. (Ed.), Handbook of Reliability Engineering. Springer, London, pp. 213–234.

[23] Littlewood, B., 1981. Stochastic reliability-growth: A model for fault-removal in computer-programs and hardware-design. IEEE Trans. Reliability 30 (4), 313–320.

[24] Littlewood, B., Verrall, J. L., 1973. A Bayesian reliability growth model for computer software. Journal of the Royal Statistical Society, series C 22 (3), 332–346.

[25] Lyu, M. R. (Ed.), 1996. Handbook of Software Reliability Engineering. McGraw-Hill, New York.

[26] Macêdo, A., Ferreira, T. B., Matias Jr., R., 2010. The mechanics of memory-related software aging. In: Proc. 2010 IEEE Second International Workshop on Software Aging and Rejuvenation.

[27] Matias Jr., R., Barbetta, P. A., Trivedi, K. S., Freitas Filho, P. J., 2010. Accelerated degradation tests applied to software aging experiments. IEEE Trans. Reliability 59 (1), 102–114.

[28] Matias Jr., R., Freitas Filho, P. J., 2006. An experimental study on software aging and rejuvenation in web servers. In: Proc. 30th Annual International Computer Software and Applications Conference. pp. 189–196.

[29] Matias Jr., R., Trivedi, K. S., Maciel, P. R. M., 2010. Using accelerated life tests to estimate time to software aging failure. In: Proc. 2010 IEEE 21st International Symposium on Software Reliability Engineering. pp. 211–219.

[30] Musa, J. D., Iannino, A., Okumoto, K., 1987. Software Reliability - Measurement, Prediction, Application. McGraw-Hill, New York.

[31] Pham, H., 2006. System Software Reliability. Springer, London.

[32] R Development Core Team, 2011. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0. Online: http://www.R-project.org.

[33] Salfner, F., Wolter, K., 2010. Analysis of service availability for time-triggered rejuvenation policies. Journal of Systems and Software 83 (9), 1579–1590.

[34] Schick, G. J., Wolverton, R. W., 1978. An analysis of competing software reliability models. IEEE Trans. Software Engineering 4 (2), 104–120.

[35] Suzuki, H., Dohi, T., Goševa-Popstojanova, K., Trivedi, K. S., 2002. Analysis and estimation of multi-step failure models with periodic software rejuvenation. In: Artalejo, J. R., Krishnamoorthy, A. (Eds.), Advances in Stochastic Modelling. Notable Publications, Neshanic Station, pp. 85–108.

[36] Trivedi, K. S., 2001. Probability and Statistics with Reliability, Queuing, and Computer Science Applications, 2nd Edition. Wiley, New York.

[37] Vaidyanathan, K., Trivedi, K. S., 2005. A comprehensive model for software rejuvenation. IEEE Trans. Dependable and Secure Computing 2 (2), 124–137.