Reliability Analysis in Telecommunications

Veena B. Mendiratta

Introduction

Reliability modeling and analysis of telecom networks and systems is a broad subject and includes many techniques, ranging from reliability block diagrams and other combinatorial methods at one end to Markov models, Petri nets, Stochastic Activity Networks (SANs), etc., for more complex models. Both analytical and numerical methods are used. In this paper our goal is to provide a characterization of telecom systems with respect to reliability and to present a few case studies to illustrate the types of mathematical modeling and analyses that are done in an industrial setting in the process of building a reliable telecom system. The analyses are presented in terms of the types of models that are developed in different phases of the product realization process. We discuss telecommunications reliability modeling from a systems perspective to include both hardware and software components.

As telecommunications systems are considered a critical infrastructure they have stringent requirements, typically an availability of 99.999%, often referred to as *5-Nines*. In this context, reliability modeling is an important part of the development process, starting from the design of the architecture where models are used to evaluate alternative architectures to predicting the expected field reliability at the end of the testing phase based on the lab testing results.

Very broadly, a telecommunications network system is an arrangement of computing and telecommunications assets—a group of nodes and links—that is capable of carrying audio, visual, and data communications. The main function of such a network is to provide for the efficient transmission of information from a point of origin to a point of termination. Figure 1 shows a high-level telecom network reference architecture where CDN refers to Content Distribution Network.

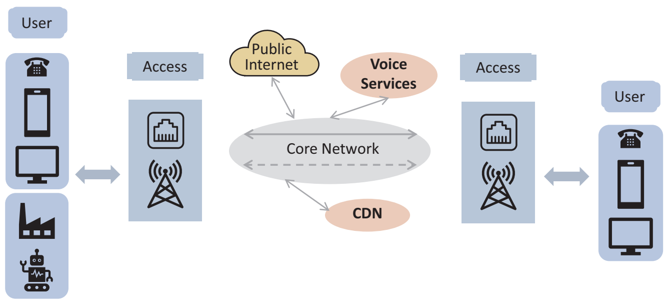

Figure 1. Telecom network example reference architecture.

In the following section we present an overview of the different types of reliability models developed in the telecom domain followed by a few selected example case studies.

Reliability Modeling

In an industrial setting, the type of reliability model developed depends on the phase of the product realization process for which the model is used, namely: concept/explore; architecture/design; post-development, pre-testing; testing; and operational phase. Figure 2 illustrates the different types of models that can be developed in the various phases of the product realization process.

In the concept and explore phases, the results from simple reliability models can be used to compare architecture alternatives. In the architecture and design phases, the models are used to predict system reliability and perform sensitivity analysis with respect to model parameters (the model on clustered systems in the next section is an example). The model can also be used, if given a system

Veena B. Mendiratta is an applied researcher in network reliability and analytics at Nokia Bell Labs. Her email address is veena.mendiratta@nokia-bell-labs.com.

Communicated by Notices *Associate Editor Richard Levine.*

For permission to reprint this article, please contact: reprint-permission@ams.org.

DOI: https://doi.org/10.1090/noti2095

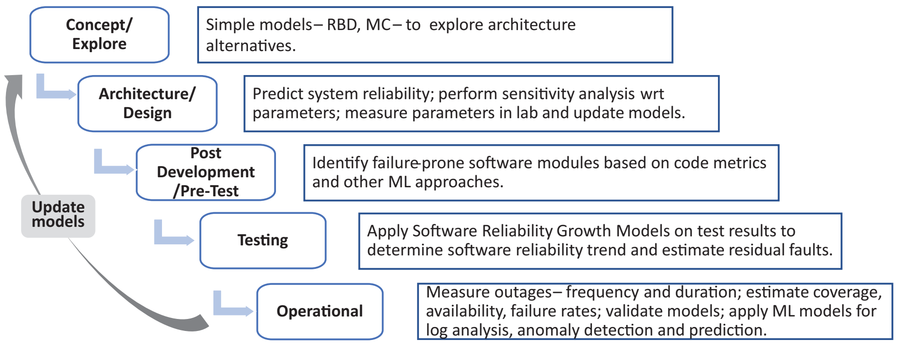
Concept/ Explore
Simple models– RBD, MC– to explore architecture alternatives.
Architecture/ Design
Predict system reliability; perform sensitivity analysis wrt parameters; measure parameters in lab and update models.
Post Development /Pre-Test
Identify failure-prone software modules based on code metrics and other ML approaches.
Testing
Apply Software Reliability Growth Models on test results to determine software reliability trend and estimate residual faults.
Operational
Measure outages– frequency and duration; estimate coverage, availability, failure rates; validate models; apply ML models for log analysis, anomaly detection and prediction.
Update models

Figure 2. Reliability modeling in the product realization process.

reliability objective, to determine the values of parameters such as error detection and recovery time, which can then be used as design goals. As the system is developed, model parameters can be measured and the reliability model can be updated with these values. After field deployment, the model can be validated with field failure data, which can also be used for modeling future releases of the product and for modeling the reliability of similar products. The example on availability in the next section discusses how field outage data can be interpreted, and used to compare the outage characteristics against the design specs. With the emergence of sensor networks and advances in network telemetry, large volumes of data are available, often in near real-time, during the operational phase. Using machine learning (ML) approaches (discussed in the next section), this data can be used for anomaly detection and prediction to proactively mitigate reliability issues. When problems do occur, ML approaches can be applied for log analysis to determine root cause as well as create failure signatures.

The basic inputs for a reliability model are the following: hardware failure rate, software failure rate, fault recovery coverage factor, and error detection and error recovery duration. Of these inputs, the most difficult to predict is the software failure rate, and it is the focus of much of the work in software reliability modeling. The modeling techniques appropriate for 5-Nines telecommunication systems are availability models for repairable systems with the following assumptions: constant failure rate; the availability of each component is independent; and redundant structures are designed for online repair where the repair of a failed unit does not impact the operation of the working units.

The exponential distribution is most commonly used in reliability models to represent the failure rate. For hardware systems, the failure rate λ is usually assumed to be a constant greater than zero, though, strictly speaking, it is a function of time, as shown in the *bathtub-shaped* curve [21] in Figure 3.

The failure rate is high during early life and is referred to as infant mortality due to the failure of weaker

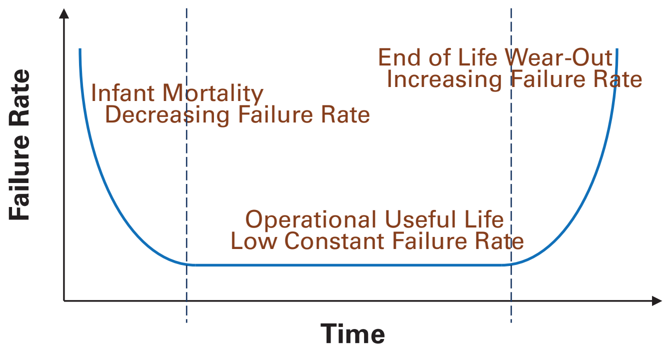

Figure 3. Bathtub curve for hardware failure rate.

components typically introduced in the manufacturing process. Once these defects are eliminated the failure rate is approximately constant and typically represents the stable operational phase of the system. Most systems spend the major part of their lifetimes operating in this flat portion of the bathtub curve. The failure rate starts increasing at the end of the operational life as the system enters the wear-out phase; this occurs as materials wear out and degradation failures occur at an ever increasing rate. Telecommunication systems are usually deployed only during the stable phase, and, therefore, in reliability modeling a constant failure rate assumption is made. If a time-varying failure rate assumption is made, the Wiebull distribution is commonly used to represent the failure rate where the shape factor of the distribution directly influences the failure rate [21].

Several reliability modeling techniques are used for telecommunication systems depending on the project phase, and also on the inputs available for the models. We classify the modeling techniques in three broad categories, namely:

- Classic (and simpler) techniques such as reliability block diagrams, fault trees, and Markov models, typically assuming exponential failure and repair rates.
- Advanced (and some newer) techniques such as nonexponential models, stochastic Petri nets, and

Bayesian belief networks.
- Machine learning (ML)-based techniques used when there is a large volume of data available such as from automated testing, or from telemetry data and log data for deployed systems.

In the early phases, for evaluation of alternatives, simpler models may be adequate, as typically the model inputs are less precise in this phase—the focus is more on the comparison of alternatives rather than the absolute model results. Markov models are a good choice for architecture-based reliability models, the results from which can provide insights into system failure and recovery behavior. The advanced techniques can be applied in the later phases of the project when more detailed inputs (architecture, failure and recovery behavior, parameter values based on lab measurements) are available. ML techniques can be applied to testing data as well to field data for deployed systems.

There are no books that are specific to telecommunications system reliability modeling; however, there are some classic books that are essential for reliability modeling. The book by Trivedi [21] is a classic text for system reliability modeling and provides a comprehensive guide to the analysis, modeling, and evaluation of the reliability of computer and communication systems along with the theoretical concepts of probability, stochastic processes, and statistics required for the analysis. The more recent book by Trivedi et al. [20] covers the advanced and newer techniques as well as evaluating the various approaches for reliability modeling. The survey article by Ahmad et al. [2] provides a detailed survey of the application of various reliability modeling and analysis techniques for telecommunications networks.

Hardware reliability modeling is a well-established discipline, and many textbooks, for example Tobias et al. [19], provide a good introduction to the subject. Software reliability modeling, on the other hand, is a (relatively) newer field and, to some extent, is more art than science. The text by Musa et al. [15] is a classic text for software reliability modeling. The handbook by Lyu [12] is a comprehensive collection of articles on various aspects of software reliability modeling. Software reliability models developed in the architecture and design phase are used for prediction and to identify software modules that are likely to be error-prone. The survey paper by Rathore et al. [18] provides an extensive survey of the software fault prediction techniques aimed at identifying fault-prone modules. Models developed in the testing phase, namely, software reliability growth (SRG) models, are based on the parameters estimated from system test data and are used to predict the residual faults in the system and to determine when to stop testing. The models on software reliability in the next section are examples of models used in the testing phase.

Modeling Examples: Case Studies

A few selected example case studies are presented in this section.

Availability. Availability is the commonly used measure and requirement in the telecommunications industry. Consequently, the models developed typically predict the expected availability or, equivalently, the expected downtime in minutes per year. When a system is said to have a reliability of 5-Nines what is meant is that the availability of the system is 0.99999. What does 0.99999 availability mean? Mathematically it means that, on average, a system is down for less than 5 minutes per year. For systems in the field, the average downtime per year is calculated as the cumulative downtime of all the deployed systems over a year divided by the number of deployed systems. The average value is skewed by the large duration outages, since very few systems fail at all in any given year for a high-availability system. The few systems that do fail are typically down for the duration of the time to repair. The Mean-Time-To-Repair (MTTR), for modeling purposes, is generally assumed to be two to four hours; however, in practice, it can vary significantly based on the outage event.

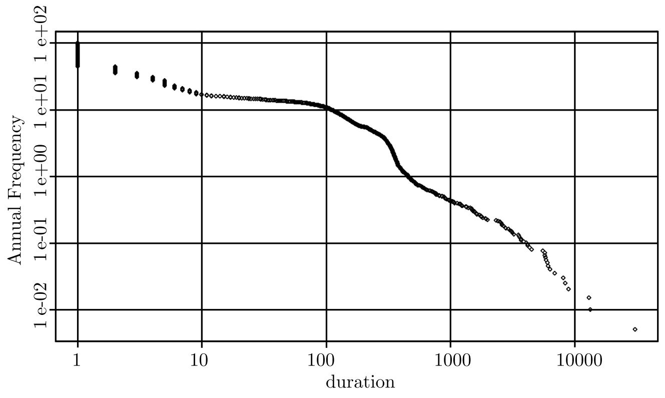

Figure 4. Pareto plot of field outage data.

For this reason, the Pareto plot [1] is a useful method to track a product's field availability performance over time. The cumulative distribution function (cdf), $F(d)$ of the outage duration d can be defined as

$$F(d) = Pr(D \leq d)$$

where D is a random variable of d. Using data from a widely deployed telecom product, normalized to have $\lambda = 100$ outage events per year and unitless time dimension, we compute the empirical cdf, $\lambda(1 - \hat{F}(d))$, which is plotted against d to obtain the plot shown in Figure 4 [9], where the x-axis is in a log scale. From the graph we see that one can expect ten events greater than 100 time units annually and one annual event greater than 300 time units. On the original scale, the area under the curve is approximately proportional to the product's availability. However, unlike the measured availability, which can be greatly influenced by a single large outage event, this graph gives a view of the true availability experienced over time. Further, we find that the graphs are very stable over time in the sense that the duration cdf is fairly unchanging once the product is designed, and improvements in availability are achieved via reduced frequency of outages modulo extreme outage duration events. Note that systems with lower successful recovery rates will have duration cdf plots similar to Figure 4 that are more spread out, that is, involve more long outage events. The designer can estimate this graph based on an understanding of the recovery mechanisms implemented and their expected parameters.

Clustered systems reliability modeling. A cluster is a collection of processing nodes in which any member of the cluster is capable of supporting the processing functions of any other member configured in a redundant $n + k$ configuration, where n processing nodes are actively processing the application and k processing nodes are in a standby state, serving as spares for the n active nodes. In the event of a failure of an active node, the application that was running on the previously active node is failed over to one of the standby nodes. The simplest redundant configuration is active/standby, in which one node is actively processing the application and the other node is in a standby state. In a configuration with n active nodes, the applications which were running on a failed processing node are redistributed among the other active nodes. The clustered system is modeled as an irreducible Markov chain with working and failed states and intermediate recovery states [14]; see Figure 5.

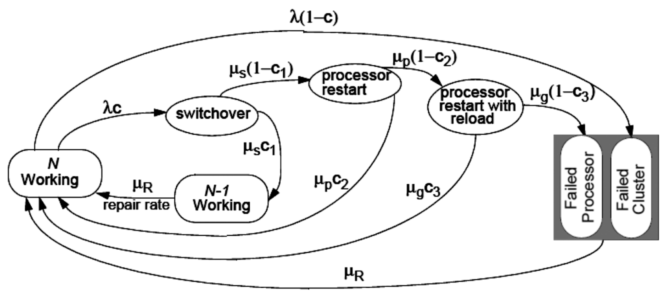

Figure 5. Cluster failure model.

The state transitions represent processor failures represented by failure rate λ, recovery and repair rates represented by μ, and the probabilities of state transitions represented by the fault recovery coverage factor c, where μ and c take on different values depending on the type of recovery and repair. The *coverage factor* is the conditional probability, given that an error has occurred, that the system recovers automatically within the designed recovery interval. From a practical perspective, the coverage factor may be computed as a product of the terms *Prob*(*successful error detection*) and *Prob*(*successful error recovery*). The key point to note here is that coverage is never perfect; it is always less than one. For a high reliability *5-Nines system*, most errors that do occur are recovered with the lowest level of recovery, for example, task restart. It is difficult and costly to test for coverage. Furthermore, for redundant systems the coverage factor is the single most important factor impacting the availability of the system. The example on testing later in this section describes a model for estimating the number of tests required to achieve a specified coverage level.

The model in Figure 5 includes only two working states: $N working$ and $(N - 1) working$. Several other working states such as $(N-2) working$, $(N-3) working$, etc., could be included in the model. However, given the high processor recovery and repair rates relative to the processor failure rates, the probability of the system entering these states is small, and the exclusion of these states has minimal impact on the model results. Since the goal here is to model the failure and recovery behavior of a processor, the failure rate λ represents the hardware and software failures for a single processor. The system failure rate is $N\lambda$, where N is the number of processors.

When an error is detected in the processor, typically the software is programmed to try a sequence of recovery actions—from least severe, such as process restart, to most severe, such as processor restart after data reload from disk—until recovery is successful. The general principle is to try the lowest-level recovery first to minimize service impact and continue escalation to the next higher-level (more severe) recovery if the lower-level recovery does not work, and models of this type are often referred to as escalating recovery models. The recovery sequence specified in the model shown in Figure 5 is hardware switchover and, if that is not successful, then processor restart. The model is run with representative values for the input parameters—λ, μ, and c—to obtain the expected frequency and duration of recoveries and outages for a single processor in the cluster and for the entire clustered system.

In field operation, there is considerable variation in the duration of outages for such systems, and the distribution of downtime duration, therefore, provides a good characterization of the failure behavior of the system. This was also illustrated above in the discussion of the availability example and the related figure (Figure 4). Figure 6, based on model results, shows a plot of the normalized frequency versus outage duration distributions for the processor where both axes are in a log scale. The plots in the figure are interpreted as follows. If the fault recovery coverage factor is 0.90—$c_1 = 0.90, c_2 = 0.90, c_3 = 0.90$—then 10% of the outages are expected to be greater than 5 minutes and 90% are less than 5 minutes; 1% of the outages are greater than 20 minutes and 99% are less than 20 minutes; and so forth. c_1, c_2, and c_3 are the probabilities of successful recoveries from the states *switchover*, *processor restart*, and *processor restart with reload*, respectively, in Figure 5. The plots for coverage factor values of 0.99, 0.75, and 0.50 can be interpreted similarly. Note that as the fault recovery coverage factor is decreased from 0.99 to 0.5 the plot flattens out—the slope decreases. This means that more of the recoveries escalate to longer duration recoveries. A steep slope of the plot represents a system where most of the outages are very short and even the recoveries that escalate to higher levels are of short duration; it is the preferred system behavior. Results from the model can be used in various ways—the single processor view provides the information necessary for evaluating whether the architecture provides the required reliability for particular applications; the cluster view provides the reliability predictions for the system. And, given an availability requirement, the designer can evaluate the sensitivity to various parameters to gain insights into the design changes needed to meet the given requirement.

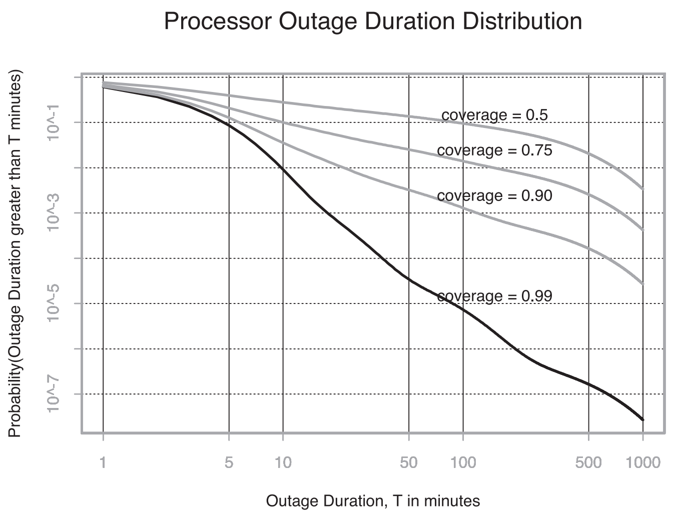

Figure 6. Processor outage duration distribution.

Software reliability modeling.

Learning from software failure data. Failure detection and fault correction are vital to ensure high-quality software. During the development and deployment phases detected failures are commonly classified by severity and tracked to meet quality and reliability requirements. Besides tracking failures, this data can be analyzed and used to qualify the software and to control the development and maintenance process. This example is focused on failure data collected during the development phase, in particular during the testing phase, and explores what we can learn by analyzing this data [4].

Change management systems log the failures detected during testing and the code fixes made to correct the underlying software defects. By applying software reliability models and statistical techniques to this defect data, we can address questions such as the following.

- Is the maintenance process increasing the software reliability?
- Is the maintenance process under control?
- How many failures are expected to occur in the field?
- What is the expected time remaining to meet the reliability requirement?

The methodology is based on trend analysis, control charts, and software reliability growth models. A time period for failure analysis is defined and trend analysis applied to detect reliability growth or decay. When reliability decay is observed, standard control chart techniques are used to detect the occurrence of assignable causes of process shifts to take necessary corrective actions. In the case of reliability growth, software reliability growth models are applied to predict the number of residual failures and the failure intensity. The following tools and techniques are used:

- For quality process control—control charts (u chart) and reliability growth trend analysis (Laplace trend).
- For reliability modeling and forecasting—reliability growth models (exponential and S-shaped models).

Control charts. A process displays variation when measured over time or over a set of items. Control charts quantitatively categorize the sources of variation into two categories: variation due to phenomena that are natural to the process and out-of-control variations that have assignable causes that could be prevented. The goal is to identify critical periods representing out-of-control variations in the number of software defects detected. Let $n(i)$ be the number of defects detected in day i with system utilization measure $\rho(i)$, and the average number of defects is $n(i)/\rho(i)$. Consider that $n(i)$ is a Poisson variable. The parameters of the control chart are

$$UCL_i(LCL_i) = \overline{\mu} \pm 3\sqrt{\frac{\overline{\mu}}{\rho(i)}},$$

$$CL_i = \overline{\mu} = \frac{\Sigma n(i)}{\Sigma \rho(i)},$$

where CL is the center line, and UCL and LCL are the upper and lower control limits, respectively.

The observations $n(i)/\rho(i)$ are plotted on a timeline against the control limits, and observations that are outside the limits are considered to be out of control. If the process is in control the center line value can be used as an estimate of the failure intensity (an estimate of the long-term process mean). The μ chart is appropriate to deal with rate measures like the daily failure intensity, considering that generally the system utilization is not uniform. When the system utilization is uniform, $\rho(i) = 1, i = 1, ..., m$. In Figure 7, based on real data, the time periods *19 to 37* and *185* are considered out of control, and would be flagged for further investigation to determine the cause of the problem.

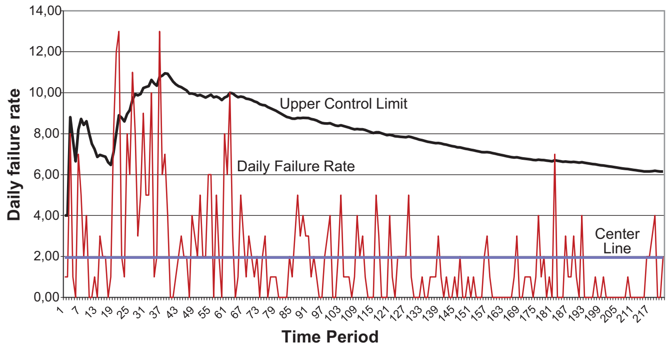

Figure 7. Control chart: Daily failure rate.

Laplace trend test. If the failure detection and fault correction process is not under control it is not possible to forecast the number of remaining defects in the software. The Laplace trend test [6] is a powerful statistical tool to monitor the reliability growth trend both over specific intervals (local trend) and over the total period of interest. The following trend regions, which correspond to a confidence level of 95%, are used: $[-2, +2]$ suggests no particular trend; $[> 2]$ suggests a reliability decay in the observed time period; and $[< -2]$ suggests reliability growth in the observed time period. The identification of local trends is important, as they correspond to inflection points in the cumulative number of failures. The chart in Figure 8, based on real data, shows a system where, after a period of no local trend, the system exhibits a local reliability growth trend by period 22 and a clear reliability growth trend after period 34.

Software reliability growth (SRG) models. Since the test and deployment phases have strict schedules and quality goals there is a need to predict reliability metrics such as faults remaining and the failure intensity. The basic approach is to calibrate an SRG model using the failure data collected during the testing phase. The calibrated model is

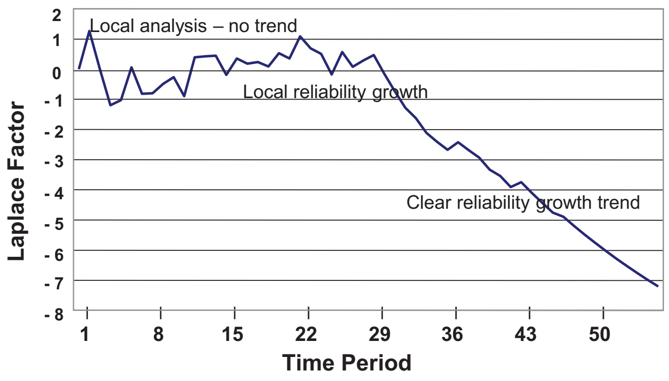

Figure 8. Laplace trend test showing reliability growth.

then used to predict the future reliability of the software, and to determine when to stop testing. SRG models can be used once the system exhibits reliability growth. These models typically assume that software failures display the behavior of a nonhomogeneous Poisson process (NHPP) where the instantaneous failure intensity of the software is time dependent.

The system represented in Figure 8, based on real data, starts exhibiting local reliability growth by period 22, so the failure data collected up to this period can be used for prediction. Figure 9 shows the predicted cumulative software failures based on an *S-shaped* nonhomogeneous Poisson process (NHPP) model [15] given by $H(t) = a[1-(1+bt)e^{-bt}]$, where $H(t)$ is the cumulative number of faults detected by time t, a is the random variable representing the initial number of faults in the software, and b is the fault rate (failure occurrence rate per fault or the rate at which the individual faults manifest themselves as failures during testing). The model has two parameters, a and b, which are estimated from the test failure data. The supposition underlying the models is that the fault rate tends asymptotically to a constant value b. The objective of the model is to predict failure intensity for each new system release with the expectation that the software failure intensity decreases with time as faults are discovered (through testing) and repaired.

For comparison, Figure 9 also shows the prediction using the failure data up to period 14, when the system does not exhibit reliability growth; note that the prediction deviates significantly from the observed data. The prediction using data up to period 22 fits quite well with the observed failures. Here, trend analysis to detect the period of local reliability growth was sufficient to provide a good estimate of the prediction period. However, this is not always the case, and discussions with developers and testers can help with determining if the observed trend is sustainable to properly select the *detection* period.

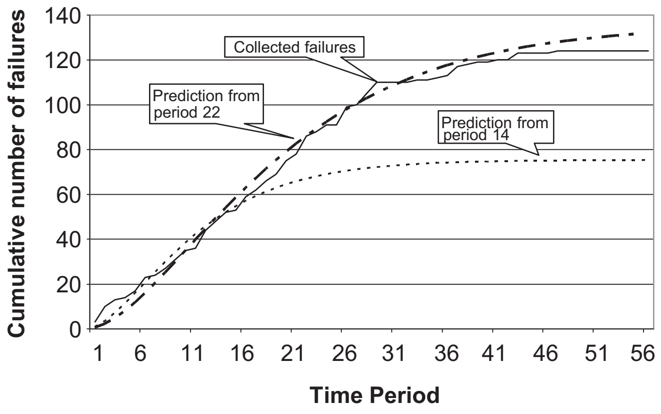

Figure 9. Prediction using S-shaped SRG model.

This methodology provides a well-structured quality/reliability framework to analyze software failure data, and can be used to certify the system in its capability to fulfill the reliability requirements.

Testing for high reliability. It can be difficult to determine whether a system meets its reliability requirements after some level of successful lab testing. In fact, the validation of ultra-high reliability in systems relying on complex software is often not possible, because the dependability requirements may lie near the limit of the current state of the art, or beyond, in terms not only of the ability to satisfy them, but also, and more often, of the ability to demonstrate that they are satisfied in the individual operational products (validation) [11].

As the model for clustered systems described above showed, the coverage factor is an important factor in determining system availability. Consequently, based on modeling, there are requirements on the minimum value of the coverage factor to meet the system availability requirements. Fault injection testing can be used to obtain estimates of the coverage factor. In this section we present an approach based on hypothesis testing that allows us to estimate the reliability of a system with a specified confidence level based on the number of tests run successfully.

Given a requirement that the coverage factor $c \geq C$, the obvious question is the following: how can we determine in the lab if the system has the specified coverage factor? The approach used is to run a series of random tests. If a test fails, that is, the fault recovery was ineffective, the error is fixed and random testing is started again. The testing should continue until the data collected is sufficient to show that the coverage factor is above the minimum requirement. Since only a small fraction of the possible tests can be actually executed, we cannot be absolutely certain that the coverage factor is adequate. What can be calculated, however, is the probability that a system without the minimum required coverage factor would have passed the proposed system test. If N randomly selected tests are conducted, the probability that there will be no failure encountered in the testing is given by $P_{nf} = C^N$, where C is the system coverage factor or the probability of successful automatic recovery on a test and N is the number of randomly selected tests. The tests are chosen, with replacement, from a distribution that is representative of the actual usage of the program. Therefore, if the coverage factor is required to be at least equal to C, and N tests were run without failure, then the probability that an unacceptable product would pass the test is no higher than P_{nf}. The objective is to continue testing, without failure, until N is large enough to make P_{nf} acceptably low. As an illustration, a plot of $P_{nf} = C^N$ versus N, ranging from 0 to 1, for three values of C is shown in Figure 10.

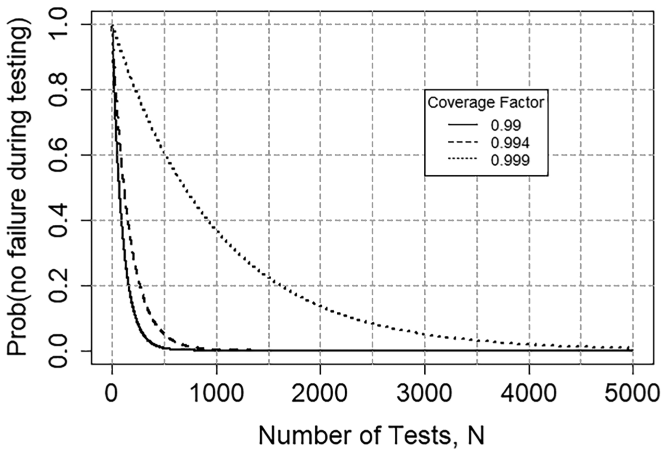

Figure 10. Probability of automatic recovery for N tests.

From Figure 10, we observe, for example, that if the coverage factor requirement is ≥ 0.994, running 700 randomly selected tests without failure indicates that the probability of an unacceptable product passing the test is about 1.5%, and for 400 tests the corresponding value is 9%. If the coverage factor objective is ≥ 0.99, a less stringent requirement, then only 400 tests are required so that the probability of an unacceptable product passing the test is less than 2%, while for a coverage factor requirement of ≥ 0.999, a very stringent requirement, the number of tests needed is 4,000 for a comparable confidence level. This methodology provides a framework for testing as well as guidelines for the number of tests that should be run for a required level of confidence in the results. For example, given c and P_{nf}, the number of test runs required can be calculated.

The procedure begins with a fully operational, nonfaulty machine. Testing is continued repeatedly until the required number of tests has been run and tests have passed consecutively, where the objective is to continue testing, without failure, until N is large enough to make P_{nf} acceptably low. It is emphasized that the validity of this approach depends strictly on the distribution of the test set being representative of the distribution of the events that will be encountered in the field.

Machine learning approaches. Data-centric approaches for reliability modeling are gaining traction given the increasing volume and diversity of data availability from sensors, telemetry, and system log data. This topic spans wide areas of machine learning, and there is a large volume of work in this area; it is beyond the scope of this paper to cover the area comprehensively. To this end, in this section, we provide a few examples and links to survey papers for readers wishing to get more information on work in this area. We believe that the greatest potential for using ML approaches for reliability modeling are in the testing and operational phases.

Some of the early work on applying ML methods in reliability modeling focused primarily on the testing phase. Examples include the paper by Gokhale et al. [7], where the authors apply regression tree models to predict the number of faults by software module based on the software complexity metrics. Also, Ma et al. [13] present a methodology for predicting fault-prone modules using a modified random forest algorithm. These types of models are developed prior to the testing phase, and the outputs are used to direct testing efforts to modules that are predicted to be more error prone; such models are especially valuable for testing of large-scale systems. Durelli et al. [5] review the state-of-the-art of how ML has been applied to automate and streamline software testing. Their results highlight that ML algorithms have been used mainly for test case generation, refinement, and evaluation where the issues were formulated and solved as supervised or semi-supervised learning problems.

In the operational phase, the application of ML techniques for reliability modeling most commonly includes:

- anomaly detection—find undesired patterns in the data;
- root cause analysis—investigate what caused the anomalous behavior;
- failure prediction—monitor metrics to predict failures based on knowledge of abnormal patterns and their causes; and
- preventive maintenance—prevent failures before they occur based on predictions.

The specific techniques used can depend on the types of data available. System logs provide a rich source of data for anomaly detection and prediction and root cause analysis where supervised, unsupervised, and reinforcement learning methods are used. Increasingly, data from automated smart sensors is used for preventive maintenance and, coupled with log data, can be used for root cause analysis. It is pertinent to note that since telecommunication systems are designed for high reliability, failures are infrequent and often may be subtle rather than catastrophic. And, as the systems are characteristically complex, detecting and diagnosing anomalous behavior can be challenging.

In recent work Kim et al. [10] developed an anomaly detection algorithm using log data from a 4G telecommunications network. The techniques used were: multivariate unsupervised learning with Principal Component Analysis (PCA) for anomaly detection, and finite state machines for root cause analyses. PCA was used for dimensionality reduction, as the feature space was large. PCA was applied on the normal data to find the subspace where the variation of the normal data was small. By characterizing the variation of the normal data in the subspace, they derived a boundary of the normal data, and developed an anomaly detection model based on it. After detecting an anomaly, the message patterns of the anomaly data were compared to those of the normal data to determine where and why the problems were occurring. Root cause analysis, which is application specific, typically relies on correlating information from different types of log sources—for example, correlating log data with resource consumption data or data from IoT sensors. In this work the error codes related to the anomalous messages provided additional detail for the root cause analysis.

Using log data He et al. [8] evaluated six different algorithms (three supervised and three unsupervised machine learning methods) for anomaly detection. They found that supervised anomaly detection methods present higher accuracy when compared to unsupervised methods; that the use of sliding windows (instead of a fixed window) can increase the accuracy of the methods; and that methods scale linearly with the log size. In practice, the data available for analysis is often unlabeled or weakly labeled, thereby precluding the use of supervised learning techniques. The paper by Candido et al. [3] presents a good survey of the different log analysis techniques using machine learning for anomaly detection and prediction and root cause analysis. The tutorial paper by Rafique and Velasco [17] includes an example of applying ML for preventive maintenance in an optical network. The tutorial paper by Musumeci et al. [16] presents an overview of ML approaches for network failure management by introducing automated methods for failure detection and prediction, and localization and identification (root cause analysis).

Summary

In this paper we provided a characterization of telecom systems with respect to reliability, and presented examples to illustrate the types of mathematical modeling and analyses that are done in an industrial setting in the process of building a reliable telecom system. The models were presented in terms of the different phases of the product realization process. As telecommunications systems have stringent reliability requirements, modeling is an important part of the development process, starting with models to evaluate alternative architectures, to predicting the expected field reliability at the end of the testing phase. A range of techniques is used, ranging from classic (and simpler) techniques, such as reliability block diagrams, fault trees, and Markov models, to more complex approaches, such as stochastic Petri nets and Bayesian belief networks. An overview of ML-based approaches was presented which can be used when there is a large volume of data available, such as from automated testing or from telemetry data and log data for deployed systems.

References

[1] Lada A. Adamic, *Zipf, power-laws, and Pareto-a ranking tutorial*, Xerox Palo Alto Research Center, Palo Alto, CA, `https://www.hpl.hp.com/research/idl/papers/ranking/ranking.html` (2000).

[2] Waqar Ahmad, Osman Hasan, Usman Pervez, and Junaid Qadir, *Reliability modeling and analysis of communication networks*, Journal of Network and Computer Applications **78** (2017), 191–215.

[3] Jeanderson Candido, Maurício Aniche, and Arie van Deursen, *Contemporary software monitoring: A systematic literature review*, preprint, arXiv:1912.05878 (2019).

[4] K. De Herdt, V. B. Mendiratta, J. M. Souza, and G. H. Zimmerman, *What can we learn from software failure data*, International Symposium on Software Reliability Engineering, Chicago, Illinois, 2005.

[5] V. H. S. Durelli, R. S. Durelli, S. S. Borges, A. T. Endo, M. M. Eler, D. R. C. Dias, and M. P. Guimarães, *Machine learning applied to software testing: A systematic mapping study*, IEEE Transactions on Reliability (2019), 1–24.

[6] S. S. Gokhale and K. S. Trivedi, *Log-logistic software reliability growth model*, Proceedings Third IEEE International High-Assurance Systems Engineering Symposium (cat. no. 98ex231), 1998, pp. 34–41.

[7] Swapna S. Gokhale and Michael R. Lyu, *Regression tree modeling for the prediction of software quality*, Proceedings of the Third ISSAT International Conference on Reliability and Quality in Design, 1997, pp. 31–36.

[8] S. He, J. Zhu, P. He, and M. R. Lyu, *Experience report: System log analysis for anomaly detection*, 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), 2016, pp. 207–218.

[9] D. A. Hoeflin and V. B. Mendiratta, *Characterizing switching system outage durations*, Proceedings Microsoft Workshop on Reliability Analysis of System Failure Data, Cambridge, UK, 2007.

[10] C. Kim, V. Mendiratta, and M. Thottan, *Unsupervised anomaly detection and root cause analysis in mobile networks*, 12th International Conference on Communication Systems Networks (COMSNETS), January 2020, pp. 1–15.

[11] Bev Littlewood and Lorenzo Strigini, *Validation of ultrahigh dependability for software-based systems*, Predictably dependable computing systems, 1995, pp. 473–493.

[12] Michael R. Lyu et al., *Handbook of software reliability engineering*, Vol. 222, IEEE Computer Society Press, CA, 1996.

[13] Yan Ma, Lan Guo, and Bojan Cukic, *A statistical framework for the prediction of fault-proneness*, Advances in machine learning applications in software engineering, 2007, pp. 237–263.

[14] Veena B. Mendiratta, *Reliability analysis of clustered computing systems*, Proceedings Ninth International Symposium on Software Reliability Engineering (cat. no. 98tb100257), 1998, pp. 268–272.

[15] John D. Musa, A. Iannino, and K. Okumoto, *Software reliability: measurement, prediction, application*, Vol. 30, McGraw-Hill, New York, 1990.

[16] Francesco Musumeci, Cristina Rottondi, Giorgio Corani, Shahin Shahkarami, Filippo Cugini, and Massimo Tornatore, *A tutorial on machine learning for failure management in optical networks*, J. Lightwave Technol. **37** (2019), no. 16, 4125–4139.

[17] Danish Rafique and Luis Velasco, *Machine learning for network automation: Overview, architecture, and applications*, J. Opt. Commun. Netw. **10** (2018), no. 10, D126–D143.

[18] Santosh S. Rathore and Sandeep Kumar, *A study on software fault prediction techniques*, Artificial Intelligence Review **51** (2019), no. 2, 255–327.

[19] Paul A. Tobias and David C. Trindade, *Applied reliability*, 3rd ed., CRC Press, Boca Raton, FL, 2012. MR3058437

[20] Kishor S. Trivedi and Andrea Bobbio, *Reliability and availability engineering: modeling, analysis, and applications*, Cambridge University Press, 2017.

[21] Kishor Shridharbhai Trivedi, *Probability and statistics with reliability, queuing, and computer science applications*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1982. MR657943

Veena B. Mendiratta

Credits

Figures 1–4 and 7–10 and author photo are courtesy of Veena B. Mendiratta.

Figures 5 and 6 ©1998 IEEE. Reprinted, with permission, from V. B. Mendiratta, "Reliability analysis of clustered computing systems," IEEE Proceedings, Jan. 1, 1998.