Using software reliability growth models in practice

Citation for published version (APA):

DOI:
10.1109/MS.2007.182

Document status and date:
Published: 01/01/2007

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 16. Dec. 2019
he amount of software in consumer electronics has grown from thousands to millions of lines of source code over the past decade. Up to a million of these products are manufactured each month for a successful mobile phone or television. Development organizations must meet two challenging requirements at the same time: be predictable to meet market windows and provide nearly fault-free software.1 Before introducing a product to the market and starting mass production in a timely manner, one should assess the software’s quality at the time of the project’s completion. Software reliability is an important criterion of software quality because it directly correlates to the user experience and inversely correlates with field failure costs.2

Software reliability is the probability of failure-free operation for a specified period of time in a specified environment.3 A commonly accepted metric for quantifying a product’s reliability is the number of faults you expect to find within a certain time. Failures are the result of a fault in the software code,4 and several failures can be the result of one fault. The process of finding and removing faults to improve the software reliability can be described by a mathematical relationship called a software reliability growth model (SRGM).

We appreciate theoretical discussions about such models5 and recent refinements of basic models6 (see the sidebar “Related Work in Software Reliability”). Our goal is to assess the practical application of SRGMs during integration and test and compare them with other estimation methods. We empirically validated SRGMs’ usability in a software development environment. We also compared the model prediction capabilities to experts’ predictions. Thus, we demonstrate that it’s possible to obtain solid reliability estimations before product release using SRGMs and failure data collected by test scripts that simulate operational behavior in a controlled test environment. These reliability estimations were used to guide management through the software project’s uncertain maturity phase.

Stability testing and model selection

We selected test data from three software development projects (TV2003, TV2004, and TV-2005) for failure analysis and reliability prediction (see figure 1). All three projects concerned...
Software stability testing

In the TV software projects’ maturity phase, software test engineers ran many stability tests to assess the software’s behavior under operational usage. The failure data from these tests formed the basis of our analysis of the software’s operational reliability.

Stability tests are automated user interaction scripts that perform actions like switching channels and random audio/video use case switching. After we abandoned or completed each test run, we analyzed the test data, noted the time of failure, and classified the failures. Software engineers worked on resolving most of the identified failures before the next test run.

Failure data and model selection

We consolidated each project’s failure logs into a database and excluded all failures related to testscript errors and hardware problems. We used the Laplace trend test and log-log plots of the data sets to determine which SRGMs were useful to investigate. These graphical estimates acted as a quick check to validate a model’s usefulness before estimating its parameters (figure 2).

The Laplace trend test shows a reliability growth trend in all data sets after 20 percent to 30 percent of the test period had elapsed. This implies that the use of non-homogeneous Poisson process SRGMs for failure prediction was validated for the TV2003, TV2004, and TV2005 data sets.

Non-homogeneous Poisson process models

The NHPP model class can predict the software’s expected number of faults and future reliability (expressed as MTBF, or mean time between failures) using test data from the later test stages. The models use observed failure history data (for example, time between failures) to estimate the residual number of faults in the software and the test time required to detect them. These models can cope with interval and point data and assume that the failure intensity declines when failures are detected and the underlying faults are removed. The general concave shape of the mean value function is linked to the fact that remaining faults in the software are more subtle and often more difficult to detect and correct.

Furthermore, finite and infinite failure models can be distinguished in this class. Finite failure models assume the possibility of finally developing a fault-free product (an asymptotic approach to a finite value), whereas the infinite failure models assume that the number of faults observed is finite, which means that the mean value function is unbounded. We selected the following finite and infinite failure models from the NHPP model class (see table 1).

Related Work in Software Reliability

Authors such as John Musa and Michael Lyu compiled the basic theory on software reliability engineering in the late ‘80s and ‘90s. Software reliability engineering aims to assess and provide feedback on the development process itself. When coupled with reliability measures and analytical tools, it would provide considerable value to customers and developers. Developers have created many software reliability growth models for different types of software projects. Various authors have discussed software reliability models’ applications, limitations, and underlying assumptions, but few have written about applying SRGMs in practice.

SRGM practitioners have applied reliability prediction models during several stages of a software development project—design and coding, integration and testing, and after release. Applying SRGMs during design and coding won’t give acceptable results because the software is often unstable and doesn’t contain all the required functionality. Making reliability predictions after release in the field is possible, but high-quality data is difficult to collect, and we consider it too late for effective project management.

References


Figure 1. Three software development projects for TV sets.
GO model. In 1979, Amrit Goel and Kazuhira Okumoto presented this model for describing a software failure process, assuming that an NHPP could describe the cumulative failure process.

Delayed S-shaped model (DSS). Shigeru Yamada, Mitsuru Ohba, and Shunji Osaki adjusted the GO model by considering test-efficiency improvement during the first test phases. This resulted in an S-shaped curve of the cumulative number of software failures over time.

Log Poisson execution time model (PET). In 1984, John Musa and Okumoto introduced this model, which aimed to develop a SRGM that could predict the expected number of failures better than existing models. The PET model could cope with nonuniform operational profiles and worked well with relatively small sample sizes.

Log power model (LP). M. Zhao and M. Xie developed a log power software reliability model with a clear graphical interpretation and simple forms of the maximum likelihood estimators for easier calculation.

We applied the maximum likelihood method to estimate the unknown parameters of the mean value function on the basis of the available failure data. After estimating the parameters, we used the mean value function to predict the expected number of failures at time \( t = \{1, 2, 3, \ldots\} \).

Applying the models to the TV2003 and TV2004 failure data

We first applied the four selected SRGMs to the stability failure data sets that were collected in the maturity phase of the previous two TV software projects completed in 2003 and 2004. The TV2003 software was subjected to 1,030 stability test hours, locating 48 faults. The TV2004 software was tested for 980 hours, locating 68 faults. This preliminary study showed that the failure predictions of the NHPP models at a quarter of the way through the total test time deviated on average 20 percent from the total number of faults found (see figure 3).

### Table 1

<table>
<thead>
<tr>
<th>Mathematical expressions of four software reliability growth models</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NHPP model</strong></td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>Finite</td>
</tr>
<tr>
<td>GO model</td>
</tr>
<tr>
<td>Delayed S-shaped model</td>
</tr>
<tr>
<td>Infinite</td>
</tr>
<tr>
<td>Log power model</td>
</tr>
<tr>
<td>Log Poisson execution time model</td>
</tr>
</tbody>
</table>
It wasn’t possible to make reasonable predictions before 20 percent of the elapsed test time because the Laplace trend test didn’t indicate a significant growth trend. In addition, the infinite failure models appeared to overestimate the total number of faults. Patrick Hartman pointed out that the infinite models behave this way because they don’t prematurely lock their predictions and rely on the assumption that as long as testing continues, faults will be found.11

The models’ differences in predicting the number of faults became negligible once we fed more than 70 percent of the stability failure data into them. The trend test and the log-log plot correctly indicated that NHPP SRGMs would prove to be useful for software reliability estimation.

**Using SRGMs during the TV2005 software project**

The next step was applying the selected SRGMs to a running project and using them to help management estimate the software’s reliability. The TV2005 project began in mid-2003 and entered its maturity phase in January 2005. The project consisted of 18 project teams at 11 sites around the world. The total development effort was more than 100 person-years. The stability tests were executed from January 2005 through April 2005. SRGMs are based on assumptions that often aren’t met in practice. Several authors have proved that it’s still possible to obtain acceptable results even if the data partly violates the model’s assumptions.12,13 Table 2 shows how well our TV2005 stability test environment corresponded to the ideal test scenario based on the four SRGMs’ main assumptions.

**Predicting faults with the help of five experts**

Models aren’t the only way to predict the number of faults. Using other methods is recommended, especially the opinions of involved experts.14 At the start of the stability test period, we asked five experts (a system architect, two test engineers, and two test managers) to estimate the total number of faults to be found. All experts were involved in the previous TV software development projects and had access to the TV2005 stability test results and failure logs. They made their first estimate on the total number of faults to be found at the start of the stability test period using only a brief summary of the previous projects’ stability tests results (that is, results from TV2003 and TV2004). During the stability test phase, we asked the experts to revise their estimate three times based

---

**Table 2**

<table>
<thead>
<tr>
<th>NHPP SRGM assumption</th>
<th>TV2005 stability tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faults are repaired immediately when they are discovered.</td>
<td>About 60 percent of the faults were removed before the next test run.</td>
</tr>
<tr>
<td>Fault repair is perfect.</td>
<td>Failure logs didn’t give information about imperfect repair.</td>
</tr>
<tr>
<td>Faults are independent of each other.</td>
<td>Most likely not, but no information was available.</td>
</tr>
<tr>
<td>No new product features were introduced during test.</td>
<td>No new product features were added.</td>
</tr>
<tr>
<td>The test script remained constant over the stability test period.</td>
<td>The test script got a minor update in March 2005.</td>
</tr>
<tr>
<td>All reliability improvements are caused by solving the open stability faults.</td>
<td>Other nonstability faults were also solved in the stability test period.</td>
</tr>
<tr>
<td>An NHPP can describe the fault detection and removal process.</td>
<td>True, according to the Laplace trend test.</td>
</tr>
<tr>
<td>Each unit of test execution time is equally likely to find a failure.</td>
<td>Assumption holds for stability test scripts.</td>
</tr>
<tr>
<td>Testing follows an operational profile.</td>
<td>Software engineers estimated a functional profile coverage of 90 percent.</td>
</tr>
</tbody>
</table>
Figure 4. Five experts’ fault estimations during the TV2005 project.

Figure 5. Failure predictions the (a) GO model and (b) LP model made during the stability test phase.

on the current TV2005 stability test results.

Because the trend test didn’t indicate a significant growth trend before that time. Figure 5 shows the GO and the LP models’ failure predictions at three moments during the stability test phase (that is, after 260, 570, and 950 hours). At each estimation moment, we fed the models the currently available failure data.

Figure 5b shows the LP model’s solid performance in matching the true failure occurrences, especially at t = 260. The results also indicate that once enough data was available, the model’s predictions improved. The blue dots represent the actual number of failures known at the last model estimate, which was at 71 percent of the test time (t = 950 hours).

The graphs don’t give an indication of the number of faults (that is, unique failures). An estimate of the total number of faults to be found in the test period will help management decide how much resources they must allocate to resolve them. We used the following formula to convert the predicted number of failures into the number of faults:

Uniqueness ratio\(_i\) = \[\frac{\text{unique failures}}{\text{total failures}}\]

These predictions supported management decision making regarding the TV2005 project’s release time and resource allocation (see table 3).

Comparing estimation methods

We subjected the TV2005 project to 191 stability test runs lasting 1,342 hours total. In this period, 103 stability failures were located, 57 of which were unique and classified as faults. The stability test script covered an estimated 90 percent of the software’s functions.

Estimates versus reality: Models

The models’ prediction results were remarkable because the LP and PET model already predicted the total number of faults found after 1,340 test hours at t = 260 within a 10-percent range (see table 3). To further compare the models’ predictive capabilities, we subjected all estimations to the following two performance criteria: mean square error (MSE), which expressed the model’s “goodness of fit,” and accuracy of estimation (AE), which expressed the prediction error. A lower value indicated a better score on both performance criteria. Figure 6 shows the superior performance of the infi-
nite models class at matching the true failure occurrences, especially at 20 to 40 percent of the elapsed test time.

The model performance on all criteria improved over time, except for the DSS model because the cumulative failures did not follow an S-shaped curve over time (figure 5). The AE criterion assesses the model’s ability to make long-term predictions. The figure confirms that both infinite models can predict the total number of faults found within a 10-percent margin after only 25 percent of the elapsed test time. The convergent behavior of SRGMs on both performance criteria increases confidence in the models.

Nevertheless, it’s hard to explain why a particular SRGM outperforms another. SRGMs are designed for a special purpose and rely on certain assumptions about the test environment. In this case, the PET and LP model closely matched the situation at hand.

Estimates versus reality: Experts and models

To compare the expert predictions with the models, we averaged the predictions of both the experts and the two model classes. The experts made their first estimation at the start of the stability tests. Most experts initially predicted that the total number of stability faults would be at least twice as high as in previous projects owing to the TV2005 software project’s increased complexity.

Figure 7 shows that experts gave conservative estimates compared to the models. When more failure data became available, the experts gradually adjusted their predictions until they reached an average of 60 stability faults. The log-like infinite SRGMs outperformed both the finite models and the experts.

What made using SRGMs in this project so successful was the timely feedback the engineers and managers received. Once they saw the models’ benefits, they took the time to provide the data and interpret the models’ results. To support the use of SRGMs, we developed a Matlab-and Excel-based tool that let project managers feed their test data into a spreadsheet and obtain a fault prediction in return. Once a proper software reliability engineering prediction tool was in place, using prediction models for project management required little effort: approximately two hours a day for one person to perform the analysis for a project of 100 engineers.

Although the trend test and log-log plot helped us select suitable SRGMs beforehand, it’s not a guarantee that all selected models will perform equally well. We
are of the opinion that the first estimation on the total expected number of faults in the software should come from the project’s experts. They’ll consider other factors, such as adapted development processes, staff changes, improved test methodology, and employee motivation. The models outperformed the experts in the projects we investigated, but it is too early to declare victory and conclude that models always perform better than experts. SRGMs should be applied to many more projects before we can arrive at such a conclusion.

We encourage practitioners to apply software reliability models and techniques in practice, but do assess the software environment in which you will be using them beforehand. Management can use all the support they can get in assessing their software’s reliability and estimating the required testing effort. Software reliability engineering practitioners and scientists can contact Vincent Almering at vincent.almering@nxp.com for the three failure data sets and Matlab/Excel tools.

References