

Progressive Software Reliability Modeling

Samuel Keene
s.keene@ieee.org

Software is typically the long pole in the tent in terms of its total contribution to system unreliability. Measuring software reliability or, more so, predicting software reliability is a challenge. This is especially true before the code is fielded. This paper recommends a practical approach to this challenge. A prediction model developed by Sam Keene, hereafter referred to as the Keene model, can be used to predict the software failure rate prior to code development [1]. It predicts the software reliability based upon characteristics of the software and the capability of the developing organization. The model is then progressively refined throughout development as defect data is accumulated.

Level 1 (historical approach)

Companies, who have metrics on the fault discovery rate of their predecessor products, can use these fault rates, normalized to the current code extent, i.e. KSLOC, to project the reliability of their current product. The company typically assumes some level of improvement of the current product over its predecessor product. For example, they would assume a 10% reduction in faults in the new product vis-à-vis the predecessor product.

But there were limitations,

- Often the developer did not have SW reliability history
- The supposed 10% improvement was usually unfounded (too much exuberance)
- There was no way to properly account for process initiatives as to whether there was any improvement and the expected size of the improvement.

Level 2 (Process based reliability prediction)

The Keene model projects fault/density based upon empirical data correlating fault density to the development organization's Capability Maturity Level (CMM level). The model also transforms the latent fault density into an exponential reliability growth curve over time, as

failures surface and the underlying faults are found and removed.

The original software reliability modeling used Keene's Development Process Based Model. The key inputs to this model are the:

- SEI level for the developing organization,
- Number of months for the software failure rate to stabilize,
- Number of failure replications expected prior to fault removal,
- Fault activation rate,
- Percentage of all failures that are critical failures, and the
- Projected run time per month.

The reliability of the code has been observed to stabilize after four years in the field [2]. At that point, the software failure rate plateaus at a stable level. Subsequent releases of the code plateau after two years in the field.

Level 3 (The Raleigh Model Fault Density Predictor)

The Raleigh Model is the best tool to predict software reliability during development. The Raleigh model is a member of the Weibull distribution with its shape parameter, $m = 2$. The exponential distribution is also a special case of the Weibull with the shape parameter, $m = 1$. The Raleigh prediction model forward chains all the defect discovery data, collected throughout the development phases, to predict the software's latent fault density. (Latent errors are those errors that are not apparent at delivery but manifest themselves during subsequent operations. This is contrasted to patent errors, which are obvious by inspection.) The inputs to the Raleigh model are the defect discovery rates found in the following design phases:

High Level Design,
Low Level Design,
Code and Unit Test,
Software Integration and Unit Test,
System Test,

Table 1. Defect Removal Patterns and Raleigh Projections

| Project | Language | LOC | First Yr Defect Density Faults/KSLOC | Total Latent Defect Density Faults/KSLOC | Raleigh Estimate Defect Density Faults/KSLOC |
|---------|----------|-------|--------------------------------------|--|--|
| A | Jovial | 680K | 0.3 | 0.6 | 0.6 |
| B | PL/1 | 30K | 3.0 | 6.0 | 6.0 |
| C | BAL | 70K | 0.2 | 0.4 | 0.3 |
| D | Jovial | 1700K | 0.4 | 0.8 | 0.9 |
| E | ADA | 290K | 0.3 | 0.6 | 0.7 |
| F | ----- | 70K | 1.1 | 2.2 | 2.1 |
| G | ADA | 540K | 0.6 | 1.2 | 1.1 |
| H | ADA | 700K | 0.2 | 0.4 | 0.4 |

The defect density found in System Test completes the development phase input data. The Raleigh model uses the profile of defect discovery, by development phase, to project the latent fault density at delivery.

Kan has reported good agreement between Raleigh project defect densities and field measured performance as measured across eight programs [3]. He reported:

Notice that Kan’s data shows excellent agreement with the Raleigh prediction, without any bias in the estimator, to actual field experience. Also, the reliability growth shown in Table 1 for first year defects shows that 50% of the defects are removed the first year. Comparatively, Keene’s model projects 56% of all latent defects will remain after 1 year. So Keene’s model is a slightly conservative predictor of reliability growth vis-à-vis the actual data shown in Table 1.

Best Predictive practice.

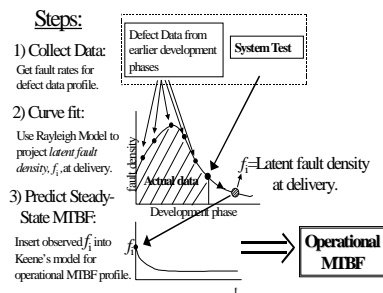
First use Keene’s Performance Based Prediction Model to estimate the software reliability. Use the Raleigh model over the development cycle to refine the latent fault density estimate. Factor this into the Keene model to transform the latent software fault density into an operational reliability.

Note software reliability has traditionally been predicted using execution time models [cf 4]. This is also a good practice, but it is more limited when the test time is shorter than the projected MTBF as is often the case on high reliability systems. The Keene Performance Based Prediction model also allows a prediction to be made prior to code development and smoothly integrates all of the development defect experience to refine its prediction. Note: George

Box, former President of the American Statistical Association, said “All models are wrong, but some are useful” This progressive software reliability modeling procedure is illustrated in Figure 1. The authors believe it will prove useful to the everyday practitioner.

Figure 1

Progressive Software Reliability Prediction



References

1. S.J. Keene, “Modeling Software R&M Characteristics,” ASQC Reliability Review, Part I and II, Vol 17, No.2&3, 1997 June, pp.13-22.
2. Ram Chillarege, Shriram Biyani, Jeanette Rosenthal, “Measurement of Failure Rate in Widely Distributed Software,” Fault Tolerant Computing Symposium (FTCS), 1995, page 424-433].
3. Stephan H. Kan, “Metrics and Models in Software Quality Engineering”, Addison-Wesley Publishing, Reading, Mass. 1995, p. 192.
4. John Musa et al, “Software Reliability, Measurement, Prediction, Application”, McGraw-Hill, New York, 1988.