Annex F

(informative)

Software reliability prediction tools prior to testing

Software failure rates are a function of the development process used. The more comprehensive and better the process is, the lower the fault density of the resulting code. There is an intuitive correlation between the development process used and the quality and reliability of the resulting code as shown in Figure F.1. The software development process is largely an assurance and bug removal process, and 80% of the development effort is spent removing bugs. The greater the process and assurance emphasis, the better the quality of the resulting code, which is shown in Figure F.1. Several operational, field data points have been found to support this relationship. The operational process capability is measured by several techniques (see Capability Maturity Model® (CMM®) [B6][9, 10, 11]). Process measures of operational process capability can be used to project the latent fault content of the developed code.

F.1 Keene's development process prediction model (DPPM)

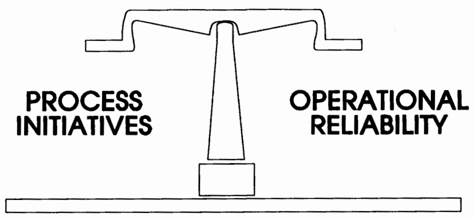


Figure F.1—Illustrating the relationship between process initiatives (capability) and operational reliability

Figure F.2 illustrates the defect density rate improves (decreases) as the development team's capability improves. Also, the higher the maturity level, development organizations will have a more consistent process, resulting in a tighter distribution of the observed fault density and failure rate of the fielded code. They will have less outliers and have greater predictability of the latent fault rate.

The shipped defects are removed as they are discovered and resolved in the field. It has been shown fielded code can be expected to improve exponentially over time (Cole and Keene [3], Keene [5], Keene [6], "New System Reliability Assessment Method" [9]) until it reaches a plateau level when it stabilizes.[12] Chillarege has reported failure data on a large operating system revealed the code stabilized after four years of deployment on the initial release and two years on subsequent releases (Chillarege [2]).

The projection of fault density according to the corresponding Software Engineering Institute (SEI) level is now shown in Table F.1. These fault density settings are based upon the author's proprietary experience with a dozen programs. This spans all of the SEI categories, except for lacking a data point for SEI level-IV programs. The SEI level V is set based upon the Space Shuttle's published performance.

[9] Process improvement models that meet these criteria include the SEI CMM® model [B6].

[10] Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

[11] This information is given for the convenience of users and does not constitute an endorsement by the IEEE of these models. Equivalent models may be used if they can be shown to lead to the same results.

[12] In this annex, the numbers in brackets correspond to those of supporting published material in F.5.

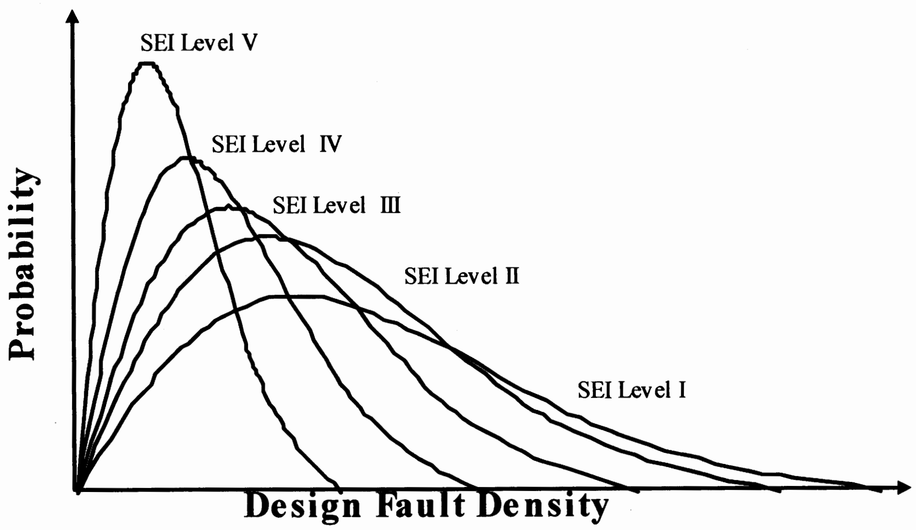


Figure F.2—Illustrating projected design defect density as a function of the development organization's design capability, as measured in terms of CMM capability

The latent fault densities at shipment are shown as a function of the SEI development maturity level in Table F.1.

Table F.1—Industry data prediction technique

SEI'S CMM level	Maturity profile Nov. 1996 542 Organizations	Design fault density faults/KSLOC (all severities)	Defect plateau level for 48 months after initial delivery or 24 months following subsequent deliveries
5	Optimizing: 0.4%	0.5	1.5%
4	Managed: 1.3%	1.0	3.0%
3	Defined: 11.8%	2.0	5.0%
2	Repeatable: 19.6%	3.0	7.0%
1	Initial: 66.9%	5.0	10.0%
unrated	The remainder of companies	>5.0	not estimated

Keene's development process prediction model (DPPM) correlates the delivered latent fault content with the development process capability. This model can be used in the program planning stages to predict the operational SR. The model requires user inputs of the following parameters:

- Estimated KSLOCs of deliverable code
- SEI capability level of the development organization
- SEI capability level of the maintenance organization
- Estimated number of months to reach maturity after release (historical)
- Use hours per week of the code
- *Percent fault activation (estimated parameter)* represents the average percentage of seats of system users that are likely to experience a particular fault. This is especially important (much less than

100%) for widely deployed systems such as the operating system AIX that has over a million seats. This ratio appears to be a decreasing function over time that the software is in the field. The early-discovered faults tend to infect a larger ratio of the total systems. The later removed faults are more elusive and specialized to smaller domains, i.e., have a smaller, and stabilizing, fault activation ratio. A fault activation level of 100% applies when there is only one instance of the system.

— *Fault latency* is the expected number of times a failure is expected to reoccur before being removed from the system. It is a function of the time it takes to isolate a failure, design and test a fix, and field the fix that precludes its reoccurrence. The default on this parameter is as follows:
    — SEI level V: Two reoccurrences
    — SEI level III and level IV: Three reoccurrences
    — SEI level I and level II: Five reoccurrences
— Percent severity 1 and severity 2 failures (historical)
— Estimated recovery time (MTTR) (historical)

F.2 Rayleigh model

The Rayleigh model uses defect discovery rates from each development stage, i.e., requirements review, high level design inspection, etc., to refine the estimate the latent defect rate at code delivery. This model projects and refines the defect discovery profile improving the projection of the estimated number of defects to be found at each succeeding development stage up to product release. One popular implementation of the Rayleigh model is the software error estimation procedure (SWEEP) released by Systems and Software Consortium, Inc. (SSCI).

NOTE—For example, the executable code for the Rayleigh model is provided in *Metrics and Models in Software Quality Engineering* [4].

The input data are the defect discovery rates found during the following development stages: high level design, low level design, code and unit test, software integration, unit test and system test. The defect discovery profile is illustrated in Figure F.3.

The SWEEP model refines the initial Keene model process-based estimate. The Figure F.4 shows the reliability growth curve from the Keene model can be beneficially applied to the latent error estimate of the SWEEP model.

NOTE 1—The reliability estimate provided by the Keene model gives an early (in development) reliability estimate to SR. This initial estimate can next be subsequently refined by the Rayleigh model incorporating actual development data defect rates collected at each process stage of development, i.e., requirements, high level design, low level design, code, software integration and test, system test.

NOTE 2—The Rayleigh model's projected latent defect density can then be extrapolated forward in time using the Keene model fault discovery and plateau profile. This is shown in Figure F.4 and explained in the following paragraph.

Figure F.4 illustrates the data fusion of the prediction models. Prior to the three steps in this process, there is a step involving an *a priori* estimate of the latent fault rate and its associated field failure rate of the code. This is accomplished by the Keene process based prediction model. Once the code is under development and subjected to inspections, reviews, and tests, the Rayleigh model can better map the actual projected defects. This prediction process is further tuned by applying the exponential fault discovery and removal profile of the Keene model.

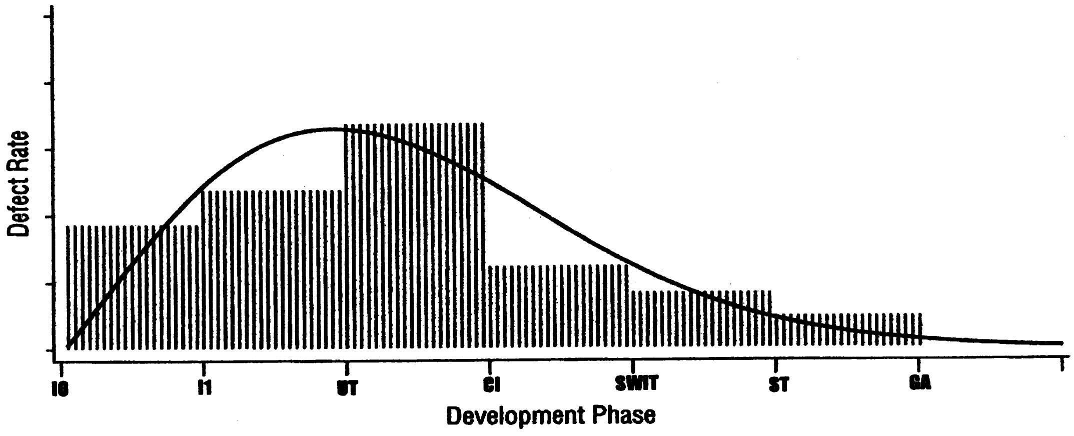


Figure F.3—Illustrative Rayleigh defect discovery profile over the development stages

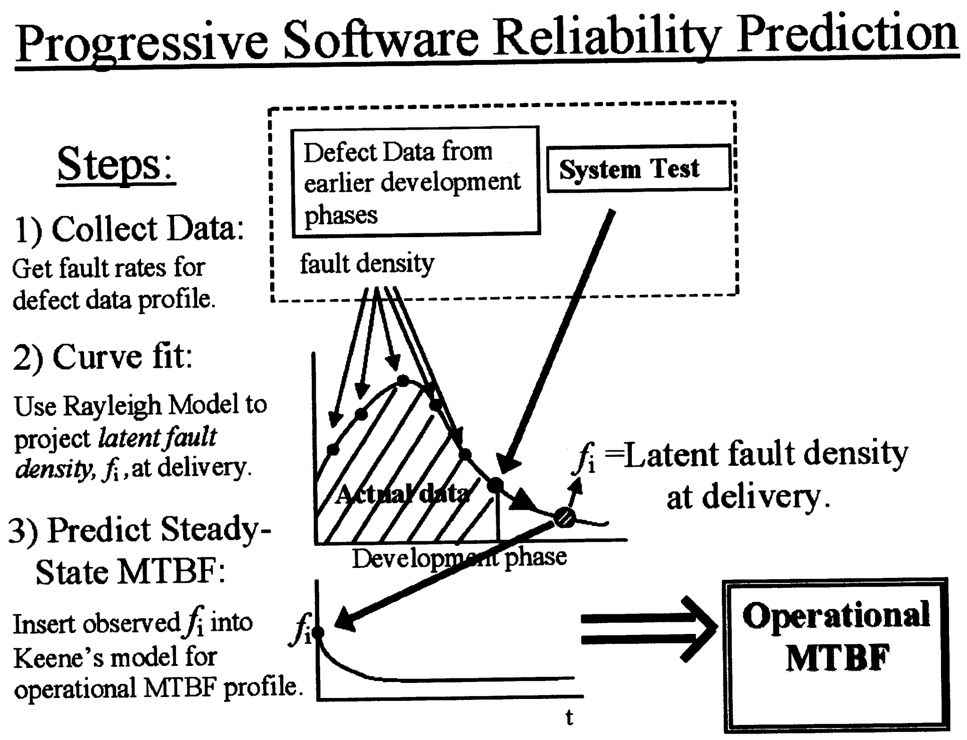

Progressive Software Reliability Prediction
Steps:
1) Collect Data:
Get fault rates for defect data profile.
2) Curve fit:
Use Rayleigh Model to project *latent fault density*, f_i, at delivery.
3) Predict Steady-State MTBF:
Insert observed f_i into Keene's model for operational MTBF profile.
Defect Data from earlier development phases
System Test
fault density
Actual data
Development phase
f_i =Latent fault density at delivery.
f_i
t
Operational MTBF

Figure F.4—Progressive SR prediction

F.3 Application of Keene and Rayleigh models

F.3.1 Software reliability estimates

The concerted application of the reliability models, DPPM, SWEEP, and CASRE, makes the best use of whatever data is available to predict the code reliability at the various stages of development. The reliability models in the CASRE tool suite are preferred for SR prediction, but these models require operational code. Their predictions are based upon intra fail times. The Keene DPPM provides an *a priori* reliability estimate that is largely based upon the capability level of the developing organization. It also provides a fault discovery and removal profile that converts the latent fault content at delivery to an observed user-experienced failure rate. The Rayleigh model correlates the rate of defects found throughout the development process to the expected latent error content. So the Keene model, the Rayleigh model, and the CASRE tool suite are progressively applied improving the failure rate estimate of the code.

The Keene DPPM gives the developing organization an early estimation of the risk that code will not meet reliability expectations or to assure the delivery of more reliable code. It applies more attention to the development history to better estimate the model parameters such as the expected failure latency of the code. The development organization can quantify its opportunity for delivering more reliable code by improving its development capability. Aerospace- and military-based companies usually know their SEI capability level so applying the Keene DPPM is a straightforward process. Commercial companies can look at the SEI CMM rating criteria and reasonably estimate their current capability level and also see what they can do to improve their process. There is the old axiom: "what gets measured, gets improved." The Keene DPPM just quantifies the expected return of investment in terms of reliability improvement for investing in process improvement.

The SWEEP model allows the developer to critically examine the quality of his process prior to test. It makes use of the discovery rate of defects found in reviews and inspections throughout the development process. The projected latent defect content will be best when there is a lower profile of defects found and when the rate of defect discovery is peaked earlier in the development process.

The CASRE reliability tool suite gives the best estimate of operational reliability of the code since it is based upon observing actually operating code. There is a caveat here. Software improves during test as defects are exposed and removed. It is often said that "software learns to pass its tests." This is true and also beneficial to the end user so long as the testing adequately represents the end user's actual application. So the quality of the testing experience depends on how well the customer's operational profile is known and replicated in the test suite. There is another limitation to basing the operational failure rate on actual test data. That is, the major problem of field failures lies in requirements deficiencies. The testing's purpose is to verify that the code meets the product specifications. Requirements problems that escape the specification will not be caught in test. So each reliability prediction method has its limitations as well as its application strengths.

There is benefit in using all of these reliability models and combining the results as depicted in Figure F.4. They provide a reliability focus at each point in the development process.

F.3.2 Development process model

Government contractors have made use of the Keene DPPM for over a decade and have reported favorable results (Bentz and Smith [1], Peterson [7], Peterson et al. [8], Smith [10]) This model is best used in conjunction with SWEEP and CASRE as previously stated. The use of the model is straightforward and intuitive. It calls out experience factors for the developer to pay attention in the development process, such as the failure rate latency. The failure rate latency is the expected number of times a failure is likely to occur before it is isolated to a fault and that fault removed.

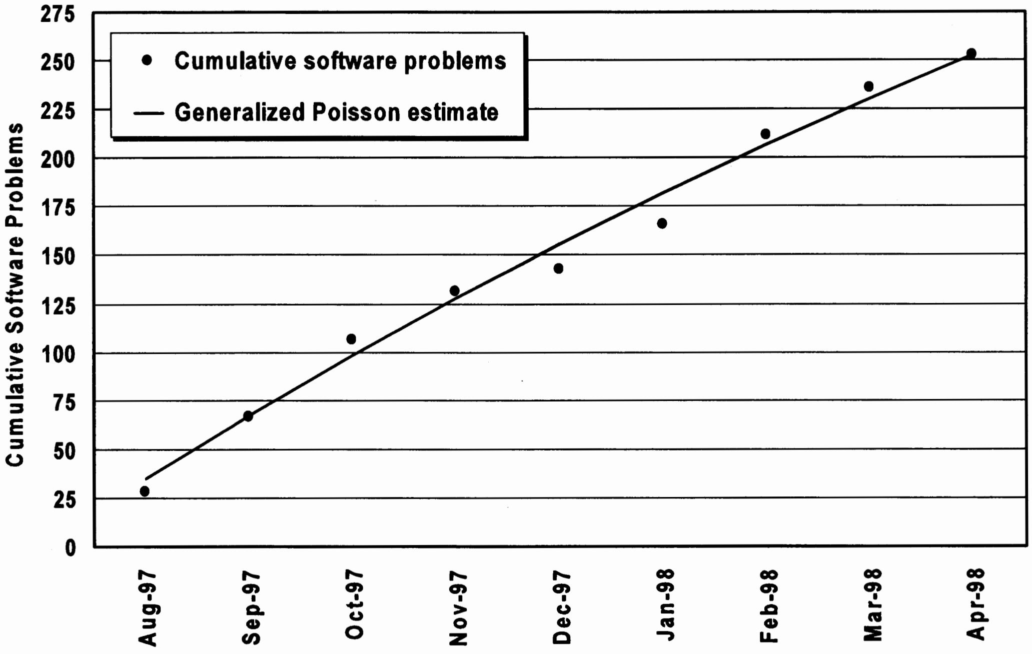


Figure F.5—CASRE results for DDS tactical software problems by month

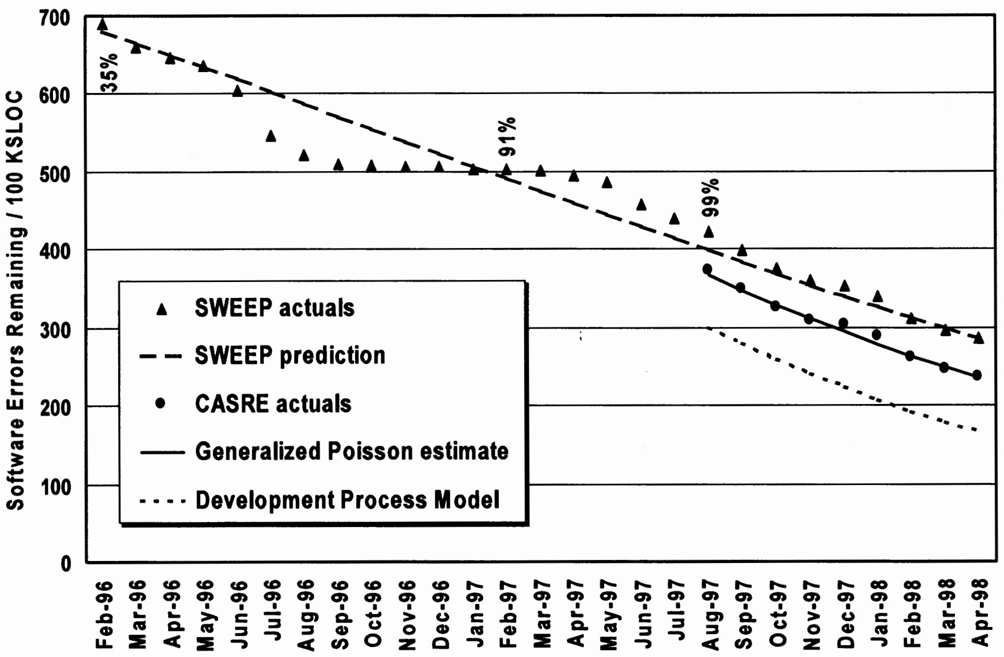
F.3.3 SWEEP predictions

The SWEEP model program, written by the Software Productivity Consortium, implements mathematical models to predict software fault rates. SWEEP uses software error data obtained from reviews, inspections, and testing throughout the development cycle to predict the succeeding number of errors that will be found later. It also estimates the latent fault content at delivery. SWEEP has the following three modes of operation: a time-based model, a phase-based model, and a planning aid. All three models use a two-parameter Rayleigh distribution to make predictions of the rate of discovery of the remaining defects in the software. SWEEP's time-based model was used for this analysis because software was being coded, tested, integrated, and retested simultaneously. After testing began, software error data was obtained from software problem reports in real time and grouped into months. As suggested in the SWEEP User Manual, the number of problems per month was normalized to errors per 100 KSLOC to account for the fact that the amount of software being tested was increasing.

F.3.4 Combined results

To easily compare and integrate all the model results, they are plotted on one graph. Since all three models could provide estimates of software errors remaining, it was decided to plot this on the vertical axis. For the CASRE curve, the errors remaining were calculated by subtracting the software problems from CASRE's estimate of the total number of errors. To match the SWEEP results, the CASRE results were also normalized to errors per 100 KSLOC. The development process model curve was easily plotted on the graph since it is based on errors per KSLOC. For the SWEEP curve, the errors remaining were calculated by subtracting the software problems from SWEEP's prediction of the total number of errors present. Figure F.6 shows the results of all three models. The percentages shown indicate the fraction of the total lines of code that were present during periods before September 1997. Note that the CASRE and SWEEP actuals differ only because the CASRE and SWEEP estimates of the total number of errors are different. The development process model curve would be even closer to the others if a few months of testing had been assumed before starting it. At least in this case, it seems clear that SR can be predicted well before system integration testing.

The graph in Figure F.6 shows that the reliability models are trending in a similar pattern. The DPPM model (illustrated as the "Development Process Model" in that graph, underestimates the remaining software error content by approximately 30%. This error content may vary between developing organizations and between projects. There is an opportunity for developing organizations to pay attention to their modeling results and compare these to actual field reliability results. Several models can then be normalized or refined for better prediction capability.



© 1999 IEEE. Reprinted, with permission, from the IEEE and Samuel Keene (author), for his paper presented at the *Tenth International Symposium on Software Reliability Engineering.*

Figure F.6—Combined results for DDS tactical software problems by month

F.4 Summary

The Keene development process model provides a ready model to estimate fault content and the resulting failure rate distribution at requirements planning time. It rewards better process capability of the developing organization with lower fault content and projected better field failure rates. This model requires the developer to know some things about his released code experience to fill in all the model parameters. It is now being popularly applied by several defense and aerospace contractors.

The Rayleigh SWEEP model is useful in projecting the number of defects to be found at each development stage. This helps in resource planning and in setting the expectation for the number of faults to be uncovered at each phase. It uses the defect data discovery profile to refine the initial Keene model prediction, for projected latent defects at delivery.

Both the Keene DPPM and the SWEEP Rayleigh model are useful additions with the CASRE test suite. They round out the reliability prediction tool kit and provide a continuing reliability focus throughout the development cycle. This continuing reliability focus throughout the development process will promote delivering and assuring a more reliable software product.

F.5 Supporting published material

[1] Bentz, R., and Smith, C., "Experience report for the Software Reliability Program on a military system acquisition and development," *ISSRE '96 Industrial Track Proceedings,* pp. 59–65.

[2] Chillarege, R., Biyani, S., and Rosenthal, J., "Measurement of failure rate in widely distributed software," *25th Annual Symposium on Fault Tolerant Computing,* IEEE Computer Society, June 1995.

[3] Cole, G. F., and Keene, S., "Reliability growth of fielded software," *ASQC Reliability Review,* vol. 14, pp. 5–23, Mar. 1994.

[4] Kan, S. H., *Metrics and Models in Software Quality Engineering,* Reading, MA: Addsion-Wesley Publishing, 1995, p. 192 (Rayleigh model discussion).

[5] Keene, S. J., "Modeling software R&M characteristics," Parts I and II, *Reliability Review,* June and September 1997.

[6] Keene, S. J., "Modeling software R&M characteristics," *ASQC Reliability Review,* Part I and II, vol. 17, no. 2 and no. 3, June 1997, pp.13–22.

[7] Peterson, J., "Modeling software reliability by applying the CASRE tool suite to a widely distributed, safety-critical system," *11th Annual ISSRE 2000,* practical papers, San Jose, CA, Oct. 8–11, 2000.

[8] Peterson, J., Yin, M.-L., Keene, S., "Managing reliability development & growth in a widely distributed, safety-critical system," *12th Annual ISSRE 2001,* practical papers, Hong Kong, China, Nov. 27–30, 2001.

[9] Reliability Analysis Center and Performance Technology, "New System Reliability Assessment Method," IITRI Project Number A06830, pp. 53–68.

[10] Smith, C., *NCOSE Symposium Proceedings,* St. Petersburg, FL, June 23, 1998.

[11] Software Productivity Consortium, "Software Error Estimation Program User Manual," Version 02.00.10, AD-A274697, Dec. 1993.

[12] Uber, C., and Smith, C., "Experience report on early software reliability prediction and estimation," *Proceedings of the 10th International Symposium on Software Reliability Engineering,* practical papers, Boca Raton, FL, Nov. 1–4, pp 282–284.