JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 4, January 2007

Experience with Risk-Based Software Defect Prediction Models

Norman Schneidewind*
Naval Postgraduate School, USA

We share with you our experience in developing, analyzing, and validating software requirements risk models. In addition to telling you about successes, we tell you about some of the problems in adapting such models to real world applications like the NASA Space Shuttle and Goddard Space Flight Facility. We believe you will find this article helpful because if requirements risk can be identified early in development, you could save your customers a lot of heartburn, once their software is delivered!

I. Purpose

OUR purpose is to demonstrate models for predicting risk (i.e., probability that defects exist on a software release), as a function of risk factors. (see definitions below). This objective is motivated by the fact that one of the software development functions of the NASA Space Shuttle Flight Software organization is to evaluate the risk of implementing requirements changes. These changes can affect the reliability and maintainability of the software. To assess the risk of change, the software development contractor uses a number of risk factors. The risk factors were identified by agreement between NASA and the development contractor based on assumptions about the risk involved in making changes to the software. This formal process is called a risk assessment. No requirements change is approved by the change control board without an accompanying risk assessment. During risk assessment, the development contractor will attempt to answer such questions as: "Is this change risky because the size of the change is large or there are conflicting requirements. If this were the case, a high-risk value would be assigned for this requirements change.[1]

In addition, we were motivated to attack this problem because, as explained by,[2] defects can remain undetected in the software a long time. Therefore, defect *prevention* is a far more practical solution to improving reliability than defect removal. Thus, with a model at hand, using prediction, defects can be *anticipated*, rather than reacting to them, after they have been detected. Furthermore, it is important to do a *quantitative* risk analysis because, as suggested in:[3] "Risk assessment is an essential process of every software risk management plan. Several risk assessment techniques are based on the subjective judgment of domain experts. Subjective risk assessment techniques are human intensive and error-prone. Risk assessment should be based on product (and *process*) attributes that we can quantitatively measure using product (and *process*) attributes."

In accordance with this principle, we made initial steps in developing quantitative measures of risk attributes in.[1] Then, in this paper, we take the next step to develop new models of risk-based defect prediction and compare their prediction accuracy with that of existing models. The risk variables—software size in source lines of code (*sloc*) and requirements conflicts (*issues*)—can be considered *both* product and process attributes, because product attributes, depend in part, on the characteristics of the process that produce the product.. In evaluating requirements risk, we are advised that requirements can be discovered throughout the life cycle.[4] Given this situation, these authors tell us that

Received 12 July 2006; revision received date month year; accepted for publication 7 November 2006. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States. Copies of this paper may be made for personal or internal use, on condition that the copier pay the $10.00 per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923; include the code 1542-9423/04 $10.00 in correspondence with the CCC. This material is a work of the U.S. Government and is not subject to copyright protection in the United States.
* Fellow of the IEEE, IEEE Congressional Fellow 2005, Professor Emeritus, Naval Postgraduate School, 2822 Raccoon Trail, Pebble Beach, CA 93953. ieeelife@yahoo.com

it is important to do a requirements risk analysis in every life cycle phase. While this is true, the most critical phase is when requirements are developed because it is then that we can have significant impact on improving software reliability by resolving requirements issues before they become a major reliability problem.

II. Major Problems in Software Requirements Engineering

The fact that the risk of using ambiguous, conflicting, and incomplete software requirements is not included in most software reliability models is unfortunate because, if requirements problems can be identified during the requirements phase, when the *cost of problem correction is low*, a significant improvement in reliability could be achieved. We hasten to add that it is understandable that most models do not include risk as a predictor because the classical reliability growth models were developed before the software engineering field recognized that bad requirements–as opposed to faulty code–may cause the code to be defective! Fortunately, requirements risk is now recognized in software engineering best practice documents, for example, in the IEEE/AIAA Recommended Practice on Software Reliability.[5]

A recognition of the relationship between requirements and reliability can be found in[6] that provides a prescription for a risk-based model: the model identifies a set of software reliability indicators or symptoms, and relates them to the expected behavior of the software in the field. Once risks have been identified, analyzed, and prioritized, a risk control strategy can be defined and implemented. Furthermore, we have to realize that our risk control decisions may fail to avert the risk.

Our recommended risk control strategy is to use defect prediction models to control by *prevention* (i.e., anticipating reliability problems by proactively predicting their occurrence and taking action to mitigating their effects).

III. Definitions

Some definitions that are central to understanding this article:

DPM: defect prediction model
ESFM: exponential forecasting model
LRM: linear regression model
SSRM: Schneidewind Software Reliability model
YSSM: Yamada S Shaped model
R^2: square of the correlation coefficient

software reliability growth model: a software reliability model that assumes that as testing and operation of the software increases, more defects will be detected and removed and reliability will increase

Risk variable examples:

sloc: *cumulative* source lines of code (sloc): (e.g., aggregation of number of source lines of code written in response to a Shuttle change requests).

issues: *cumulative* number of possible conflicts among requirements (e.g., an aggregation, across change requests of Shuttle software requirements conflicts, for example, providing greater thrust conflicts with lowering engine weight). An aggregation of conflicting requirements issues causes reliability risk.

r_i: risk variable i (e.g., risk of excessive size, excessive conflicting issues).

cumulative: the aggregation of defect and risk variable counts across requirements change requests (i.e., change requests generate the need for additional source lines of code, may result in requirements conflicts, and may cause defects to occur).

cumulative defects: predicted by risk variable i during operating time interval t (e.g., NASA Shuttle cumulative Discrepancy Reports) For example, a program executes an incorrect path due to excessive requirements conflicts and a Discrepancy Report is written.

D_i: *cumulative* defects as predicted by risk variable i during operating time interval t

t: operating time (e.g., execution, wall clock, calendar), either in test or operation.

mean relative residual error: mean of [(actual value- predicted value)/actual value].

Outlier: a value outside the range of recorded defect and risk data

IV. Approach to the Problem

We approached the problem of defect prediction by focusing on the risk that requirements changes will lead to defects in the software. Risk is a multifaceted issue and must be addressed with methods that are appropriate for the decisions being made. Historically, risk assessment and risk management professionals have focused on accident risks, natural hazard risks, business interruption risks, project risks, and financial risks. In these areas, organizations have used very systematic processes and tools to understand and prioritize these diverse risks (especially those with catastrophic consequences) so that limited resources can be effectively applied to reduce risk.[7] The general concept is given in the following equation:

$$\mathit{Risk} = \mathit{Frequency\ of\ Event} \times \mathit{Probability\ of\ Event} \times \mathit{Consequence\ of\ Event}$$

While the above equation is useful for organizing one's thinking about risk, it is difficult to quantify in the real world. The only easily quantified factor is *Frequency of Event*. How do you quantify *Probability of Event* and *Consequence of Event* in the above example? We have found it better to focus on *controlling* risk. A simple example of *controlling* risk is to apply a statistical control chart to the Shuttle Combustion Program defect groups (i.e., sets of defect identifiers) in Fig. 1 in order to identify a defect group (a range of defect identifiers) that is out of control (i.e., exceeds the mean plus three standard deviations). In the figure, *risk* is represented by defect group 8 being out of control and posing a threat (i.e., *consequence*) to the safety and reliability of this Shuttle system, with a *frequency* of 1 out of 20. This simple model could be used to alert Shuttle software managers to investigate the cause of the anomaly and correct the relevant module (s).

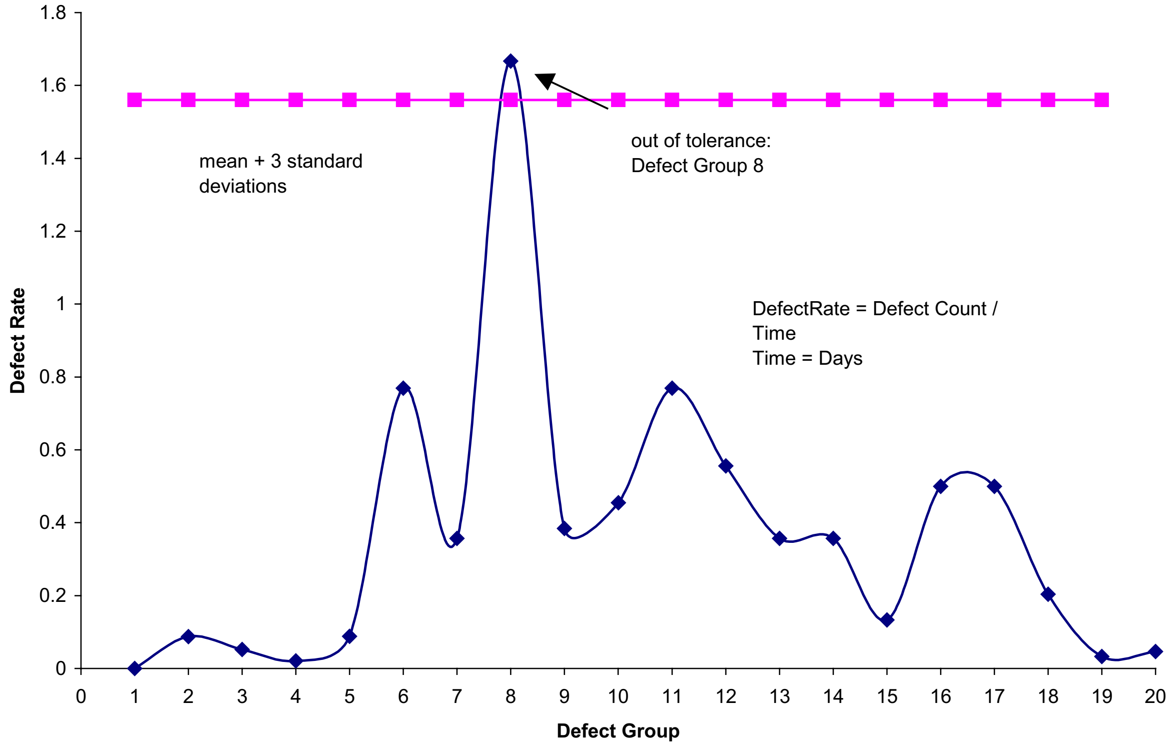

Fig. 1 Shuttle Combustion Defect Rate vs. Defect Group.

V. Defect Prediction Model (DPM)

The first model to be considered has the form $D_i = a\, r_i^b$, where a and b are parameters estimated from regression analysis. One desirable property of this model is that cumulative defects D_i should increase with time reflecting the fact that, as the incompleteness, imprecision, and ambiguity of requirement increase, the problem becomes worse over time. This characteristic can be seen in Fig. 2, where predicted defect count, for the NASA Shuttle Flight software, as a function of requirement risk variable issues, is plotted against time.

Another desirable property is that it should not be necessary to provide different prediction models for each type of defect because it was found in an operating system that 72 percent of reported field software failures were recurrences of known software faults and 70 percent of the faults had identical characteristics.[8] These authors argue that given the similarity of results, different models are not necessary to predict different types of defects. Thus, we have produced models that independent of the types of defects predicted.

A by-product of developing DPM is that rather than assuming an exponential form for D_i, a linear regression equation was fitted to the defect data. This model is referred to as LRM.

VI. Application of the Defect Prediction Model (DPM)

In this section, we take the reader through the step-by-step process of applying the DPM.

1. The objective is to predict defect count for the Space Shuttle flight software, as a function of risk variables, in order to assess this software's quality.
2. Starting with the assumed form of the model as given by $D_i = ar_i^b$, collect data to do the parameter estimation, prediction, and validation in later steps. The data are shown in Table 1. Note that data in the intervals $t = 1$, 16 are used to estimate parameters. The data in the intervals $t = 17$, 26 are used for comparing predictions with the actual defect count, The latter data were not known at the time of making the predictions.

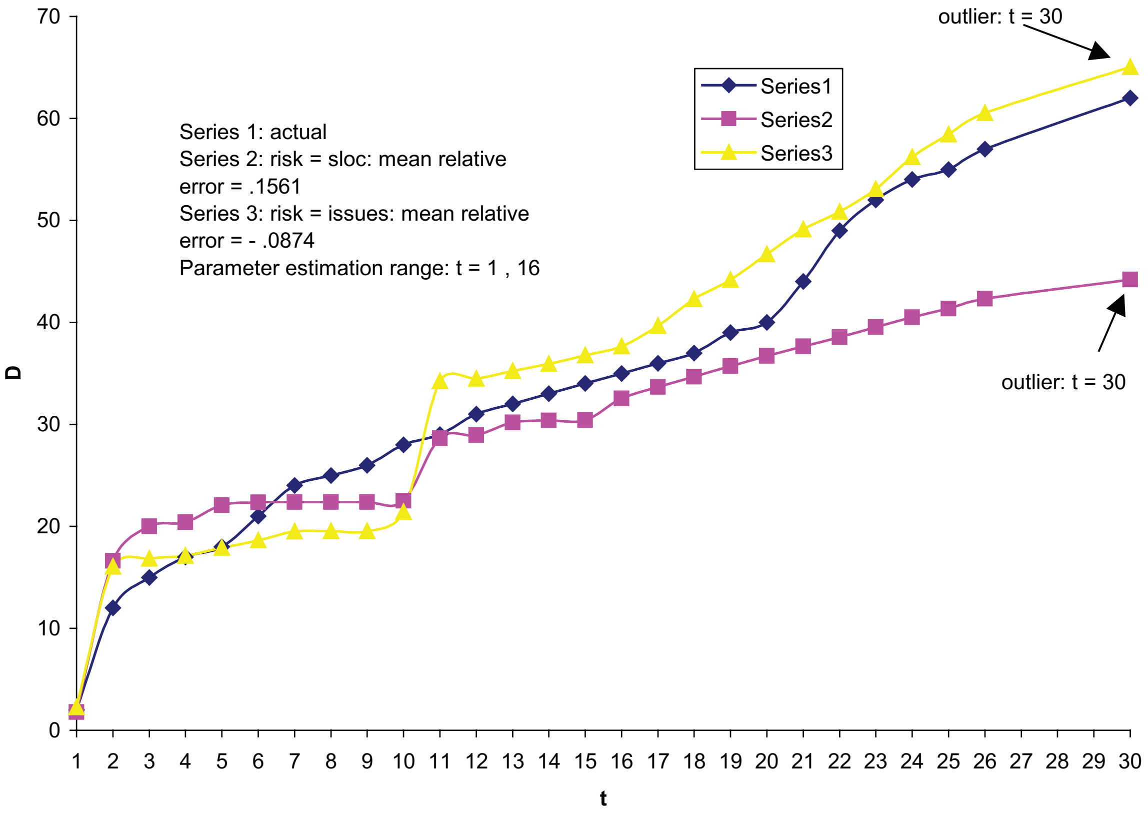

Fig. 2 Predicted Cumulative Defects D, as a Function of Cumulative Risk (sloc and issues) vs. Time Period t.

Table 1 NASA Shuttle and NASA Goddard Space Flight Center (GSFC) Data.

Shuttle sloc	Shuttle issues	Shuttle defects	GSFC defects
32	44	2	1
1933	37	10	6
807	41	3	7
100	33	2	3
447	35	1	7
77	50	3	5
10	43	3	0
7	36	1	3
0	40	1	1
32	43	2	3
1933	37	1	3
100	38	2	1
447	52	1	3
77	37	1	6
10	36	1	3
800	41	1	3
442	48	1	4
408	39	1	2
423	37	2	2
430	34	1	0
401	40	4	3
408	41	5	5
440	49	3	0
451	41	2	0
392	32	1	5
466	43	2	0
			2
			2
			4
			0
			1
			1
			2
			3

Notes: Each entry corresponds to a time interval. There are 26 intervals of data reported for the Shuttle and 35 for GSFC. The Shuttle data correspond to Part 1 of Table 2 and the GSFC data correspond to Part 2. There is no *sloc* and *issues* data available for the GSFC applications.

3. The next step is to estimate the parameters a and b. This is done by plotting $\log D_i = \log a + b \log r_i$ and noting that the $\log a =$ the Y intercept and $b =$ the slope of the plot. The equations for $r_i = sloc$ and *issues* are shown in Table 2, Part 1.
4. After the equations are obtained, predictions can be made in the time intervals $t = 17, 30$. As shown in Fig. 2, we can predict defect count at $t = 30$. Both the *sloc* and *issues* equations are used to bound the actual defect count.

   Note that the predictions in the intervals $t = 1, 16$ will be more accurate than those in the intervals $t = 1, 30$ because the former represents *retrospective* predictions compared with known defect data, while the latter represents *future* predictions compared with defect data that was unknown at the time of the predictions.
5. Then, we compare predictions produced by $D_i = a\, r_i^b$ with *actual defect count* in time intervals greater than the range of t of recorded data. We do this in Fig. 2, where the outlier time is $t = 30$. We do not have the

Table 2 Mean Relative Residual Prediction Error Comparison of Various Models. Cumulative Defect Count Predictions Parameter Estimation Sample = 16 (t = 1, 16). Prediction Sample = 10 (t = 17, 26).

Model	Data set	Error
	Part 1	
ESFM	NASA Space Shuttle (various Change Requests Discrepancy Reports)	0.0006
LRM	NASA Space Shuttle (various Change Requests Discrepancy Reports)	−0.0233 $D = 8.1446 + 1.8041\,t$ $R^2 = 0.9639$
DPM	NASA Space Shuttle (various Change Requests Discrepancy Reports)	−0.0874 risk variable = *issues* $D_\mathrm{i} = 0.0815^*\ (issues^{0.9425})$
DPM	NASA Space Shuttle (various Change Requests Discrepancy Reports)	0.1561 risk variable = *sloc* $D_i = 0.2753^*\ (sloc^{0.5407})$
YSSM	NASA Space Shuttle (various Change Requests Discrepancy Reports)	0.6624
SSRM	NASA Space Shuttle (various Change Requests Discrepancy Reports)	−0.8378
	Part 2	
ESFM	NASA Goddard Space Flight Center (Satellite System Defect Reports)	−0.0125
SSRM	NASA Goddard Space Flight Center (Satellite System Defect Reports)	−0.0137
LRM	NASA Goddard Space Flight Center (Satellite System Defect Reports)	−0.0640 $D = 3.775 + 3.2324\,t$ $R^2 = 0.9661$

ESFM: Exponential Smoothing Forecasting Model, LRM: Linear Regression Model, DPM: Defect Prediction Model, YSSM: Yamada S-Shaped Model, SSRM: Schneidewind Software Reliability Model.

actual defect count at $t = 30$, but we approximate it by computing the value of the regression equation, $D = 8.1446 + 1.8041\,t$, which was obtained from a fit of the actual defect count, at $t = 30$.

6. The purpose of this step is to interpret the results. Interestingly, it was found that step 5 did not produce the most accurate predictions because, when a linear curve was fitted to the data, the LRM equation shown in Part 1 of Table 2 produced more accurate predictions. Thus, if DPM and LRM were the only models considered, the latter would be the choice.
7. The last step is to validate the model. This is performed by computing the *mean relative residual error* for DPM and LRM in Part 1 of Table 2, where we see that *issues* is a better predictor of defect count than *sloc*, but LRM is better than either of these, yielding a low error of 2.33%. However, ESFM that does not require the use of risk variables, yields the most accurate predictions with an error of 0.06%!

VII. Software Reliability Growth Models

Now, we provide a brief introduction to the software reliability growth models that were evaluated. These models have been implemented in FORTRAN and C++ by the Naval Surface Warfare Center, Dahlgren, Virginia as part of the Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS). It can be run on an IBM PCs under all Windows operating systems. In addiction, the SSRM is one of the models recommended in the IEEE/AIAA P1633™/Draft 3, Draft Standard for Software Reliability Prediction.[5]

A. Schneidewind Software Reliability Model[9]

The basic philosophy of this model is that as testing proceeds with time, the defect detection process changes. Furthermore, recent defect counts are usually of more use than earlier counts in predicting the future. Three approaches can be employed in utilizing the defect count data, i.e. number of failures detected per unit of time. Suppose there are t intervals of testing and $d_{i\,\text{defects}}$ were detected in the ith interval, one of the following can be done:

— Utilize all of the defects for the t intervals.

— Ignore the defect counts completely from the first $s - 1$ time intervals $(1 \le s \le t)$ and only use the data from intervals s through t.

— Use the cumulative defect count from intervals 1 through $s-1$, i.e $D_{s-1} = \sum_{i=1}^{s-1} d_i$.

The first approach is used when it is determined that the failure counts from all of the intervals are useful in predicting future counts. The second approach is used when it is determined that a significant change in the defect detection process has occurred and thus only the last $t-s+1$ intervals are useful in future defect count forecasts. The last approach is an intermediate one between the other two. Here, the combined defect counts from the first $s-1$ intervals and the individual counts from the remaining are representative of the defect detection behavior for future predictions.

B. Space Shuttle Flight Software Application[10]

The SSRM has been used to predict the reliability of the Shuttle flight software. The Shuttle software represents a successful integration of many of the computer industry's most advanced software engineering practices and approaches. Beginning in the late 1970's, this software development and maintenance project has evolved one of the world's most mature software processes applying the principles of the highest levels of the Software Engineering Institute's (SEI) Capability Maturity Model (the software is rated Level 5 on the SEI scale) and ISO 9001.

Standards. This software process includes state-of-the-practice software reliability engineering (SRE) methodologies. Reliability predictions are used by the developer to add confidence to a formal software certification process comprised of requirements risk analysis, design and code inspections, testing, and independent verification and validation.

C. Yamada S Shaped Model (YSSM)[11]

The YSSM has the interesting characteristic of a defect rate that does not monotonically decrease with time, as is the case with most reliability growth models (e.g., SSRM). Instead, the rate initially increases, reflecting the immaturity of the software testing process to identify and remove defects. Subsequently, when the process matures, the defect rate peaks and then declines. Among other applications, this model has been implemented at the Naval Surface Warfare Center, Dahlgren, Virginia involving research in reliability prediction and analysis of the TRIDENT I and II Fire Control Software.

VIII. Exponential Smoothing Forecasting Model (ESFM)[12]

Our rationale for using ESFM is that, according to,[12] significant changes are possible in the data that is used in a model. Whenever there is a significant change in the pattern, computations should be made using only the most recent observations. Abrupt changes were observed in our data, for example in the defect rate, shown in Fig. 1. Therefore, we must be able to easily adjust the number of past observations that are included in our computations to model these changes.

IX. Solutions to the Defect Prediction Problem

In obtaining solutions to the defect prediction problem, we focused on DPM and an adaptation of ESFM–two new software defect prediction models. Both models use defect data from the NASA Space Shuttle flight software and the NASA Goddard Space Flight Center satellite projects. In addition, the first model uses requirements risk data from the Shuttle flight software. In order to *not* rely on the accuracy of my model predictions as the sole indicator of model validity, we experimented with other models—the Schneidewind Software Reliability Model and the Yamada S-Shaped Model[2]—comparing their accuracy with the developed models. As by-products, we found, surprisingly, that when the exponential smoothing model[13] was adapted to defect prediction, it produced the most accurate predictions of all models considered. In adapting this model to defect prediction, we developed a new and accurate method of statistically estimating the model's parameter. Other methods[12], select a value of the parameter without computing it, depending on the degree of smoothing that is desired. Another surprising result was that linear regression, with defect count as a function of defect occurrence time, produced the second best predictions. Thus, a lesson learned for the practitioner is that simple models can yield good results, and because they are simple, would be inexpensive to implement and easy to use.

X. Applicability of the Models

It is important to note two characteristics of these models. First, using cumulative values of both the independent and dependent model variables achieves a smoothing effect by aggregating the raw data to produce more accurate predictions than by using the raw data. Second, it is important to recognize that our models can only predict *known* defects because the models are based, in part, on historical *known* defects.[8] We cannot account for the defects that are present in the software but have not made their presence known!

XI. Problems that were Encountered in Obtaining a Solution

In developing these models, we must frankly admit specific deficiencies arise that are not the result of lack of understanding of the importance of certain model attributes, such as defect severity and the consequences to reliability of severity. Rather, the deficiencies occur because certain data are not available in our project's database! For example, the *perfect world* definition of risk is stated in the following: according to,[14] "risk is a function of: the possible frequency of occurrence of an undesired event, the potential severity of resulting consequences, and the uncertainties associated with the frequency and severity." In the real world, we were unable to differentiate the defects by severity in the NASA Shuttle database and to estimate the "uncertainties" associated with severity. However, we were able to predict undesired events (e.g., defects arising from requirements problems).

XII. Model Validation

While, of course, it is important to validate requirements–beyond the scope of this paper–it is also important to validate models that evaluate requirements! It is important to validate models before they are put into practice. The most important validation method is to compare model predictions against *future* actual values,[13] rather than relying on retrospectively fitting the model with historical values. One example is that we compared predictions produced by DPM ($D_i = ar_i^b$) with actual defect count in time intervals greater than the range of t for estimating the parameters a and b. We did this in Fig. 2 for the Shuttle flight software, where the outlier time is $t = 30$. Fig. 2 shows that the outlier values of *sloc* and *issues* still bound the actual defect count at $t = 30$. We are not suggesting that retrospective testing is not valuable. It is only to urge that model builders not assume that because they obtain a good fit with historical data, that their model is valid. Unfortunately, such "validation" will only be legitimate to the extent that the future is like the past!

Other validation methods involve tabulating and comparing prediction error statistics for the models. The summary of the error statistics for all models for cumulative defect predictions is shown in Table1. Table 1 also documents the LRM equations and their R^2 values.

Part 1 compares the prediction accuracy of DPM, ESFM, and LRM in Table 1, using NASA Space Shuttle flight software defect counts. Table 1 shows an amazing result: ESFM and LRM–both simple models–are more accurate than DPM, despite the fact that the latter is a function of the risk variable *issues*, whereas ESFM and LRM are only a function of time of occurrence of defects!.

Part 2 evaluates ESFM, SSRM, and LRM, using the NASA Goddard Space Flight Center (GSFC) defect counts.. One of the salient results is that, as in the case in *Part1*, ESFM comes out on top for the GSFC software.

The implication of the results is significant: the *simple models*—ESFM and LRM—can be used to accurately predict defect occurrence for the Shuttle flight software and the GSFC software and similar mission critical software.

XIII. Lessons Learned

Most important, the new risk-driven defect prediction models are superior to reliability growth models we tested for defect prediction. However, the exponential smoothing forecasting model and the linear regression model were even better for the Shuttle software; the exponential smoothing forecasting model was also superior for the GSFC software. This is particularly significant because these models do not depend on any variables other than time of defect occurrence. Thus, they are simple to implement and use. We cannot be certain that these results can be generalized to *all* software because, undoubtedly, there are counter examples from other applications, but we suggest that the process we have used for model development and evaluation can be applied to any application, *independent* of the particular result that may be attained.

References

[1]Norman F. Schneidewind, "Predicting Risk as a Function of Risk Factors", Innovations in Systems and Software Engineering, Springer-Verlag 2005.

[2]Y. Levendel, "Reliability Analysis of Large Software Systems: Defect Data Modeling" IEEE Transactions on Software Engineering, February 1990 (Vol. 16, No. 2) pp. 141–152

[3]Sherif M. Yacoub, Hany H. Ammar, and Tom Robinson, "A Methodology for Architectural- Level Risk Assessment using Dynamic Metrics, Proceedings of the 11 the International Symposium on Software Reliability Engineering, San Jose, California, 8–11 October 2000, pp. 210–221.

[4]Robyn R. Lutz and Ines Carmen Mikulski, "Ongoing Requirements Discovery in High-Integrity Systems", IEEE Software, March/April 2004, pp. 19–25.

[5]IEEE/AIAA P1633™/Draft 3, Draft Standard for Software Reliability Prediction, Prepared by the Software Reliability Engineering Working Group of the Definitions and Standards Committee of the Reliability Society, May 2006.

[6]Michael R. Lyu, editor, Handbook of Software Reliability Engineering, Computer Society Press, 1996.

[7]Charles Mitchell and Chris Decker, "Applying Risk-Based Decision-Making Methods and Tools to U.S. Navy Antiterrorism Capabilities", Journal of Homeland Security, February 2004.

[8]Norman F. Fenton and Shari Lawrence Pfleeger, Software Metrics: A Rigorous & Practical Approach, Second Edition, PWS Publishing Company, 1997.

[9]Norman F. Schneidewind, "Reliability Modeling for Safety Critical Software", IEEE Transactions on Reliability, Vol. 46, No.1, March 1997, pp. 88–98.

[10]Ted Keller and Norman F. Schneidewind, "Successful Application of Software Reliability Engineering for the NASA Space Shuttle", Software Reliability Engineering Case Studies, International Symposium on Software Reliability Engineering, November 3, Albuquerque, New Mexico, November 4, 1997, pp. 71–82.

[11]Min Xie, Software Reliability Modelling, World Scientific, 1991.

[12]Robert Goodell Brown, Smoothing, Forecasting and Prediction of Discrete Time Series, Prentice-Hall, 1963.

[13]Fredrick S. Hillier and Gerald J. Lieberman, Introduction to Operations Research, Seventh Edition, McGraw Hill, 2001.

[14]Software Safety, NASA Technical Standard, NASA-STD-8719.13A, September 15,1997.

Additional Reading

J. B. Bowles, "Caution: Constant Failure Rate Models May Be Hazardous to Your Design", IEEE Transactions on Reliability, Vol. 51, No 3, September 2002, pp. 375–377.

H. Remus and S. Zilles, "Prediction and Management of Program Reliability", Proceedings of the 4th International Conference on Software Engineering, Munich, Germany, September 17–19, 1979.

Daniel P. Siewiorek, Ram Chillarege, and Zbigniew T. Kalbarczyk, "Reflections on Industry Trends and Experimental Research in Dependability, IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 2, April–June 2004, pp. 109–127.

Wayne C. Turner, Joe H. Mize, and John W. Nazemetz, Introduction to Industrial and Systems Engineering, Third Edition, Prentice Hall, 1993.

About the author

Dr. Schneidewind was selected for an IEEE USA Congressional Fellowship for 2005 and worked with the Committee on Homeland Security and Government Affairs, United States Senate, focusing on homeland security and cyber security. He is a Fellow of the IEEE, elected in 1992 for "contributions to software measurement models in reliability and metrics, and for leadership in advancing the field of software maintenance". In 2001, he received the IEEE "Reliability Engineer of the Year" award from the IEEE Reliability Society.

Michael Hinchey
Associate Editor