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

Software Defect Process and Product Model for High Assurance Applications

Norman Schneidewind*
Naval Postgraduate School, Pebble Beach, CA

One reason we are motivated to do this research is to suggest that it is not necessary to wait until software is fielded before recording and analyzing defects. Assuming there is an effective data collection process in place, we can record actual defects and predict future defects for all software development phases. Our contribution to defect analysis research is the modeling of the process attributes and the product reliability needed for safety critical systems in NASA, using a queuing process approach that we believe is the first of its kind. We feel this approach is important because defect process attributes, such as the *time required to correct defects*, have a direct bearing on product attributes, such as the *number of old defects* that remain unrepaired. The model has the capability of computing probabilities of defect prediction and repair, and repair times This feature allows unrepaired defects to be fed back into the model queue input, along with the new defects. The result provides a comprehensive template of defect processing that the software engineer can use to assess both the efficacy of defect processing and the reliability of the product that is the outcome of the process. A major result was that the module with the highest probability of defect detection corresponds to the module with the lowest values of defect count, source lines of code, and cyclomatic complexity. Conversely, the module with the lowest probability corresponds to the module with the highest values of these attributes. Thus, the engineer would be encouraged by these results to use these relationships as a guide to predicting the probability of detection of defects on other software systems with similar attributes. A remaining challenge in this research, an issue that requires resolution, is that a coherent and comprehensive NASA defect and metrics database should be developed to support research. While the individual data items in the NASA IV&V Facility Metrics Data Program data repository are valuable, the defect data is not adequately correlated to the metrics data.

I. Introduction

In previous research, we introduced the concept of fault correction profiles. Our motivation was that, in general, software reliability models have focused on modeling and predicting the failure detection process and have not given equal priority to modeling the fault correction process.[1,2] We felt it was important to address the fault correction process in order to identify the need for process improvements. However, it is even more important to consider *defects*—a metric that can be obtained earlier that either faults or failures because it should be possible to have an even greater effect on process improvement and the achievement of reliability goals when a defect detection and repair process is utilized, as shown in Fig. 1. In our search of the literature we have not found articles that address the queuing and service aspects of defect analysis. We feel this approach is important because defect process attributes,

Received 26 June 2006; accepted for publication 13 February 2007. 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, USA. ieeelife@yahoo.com

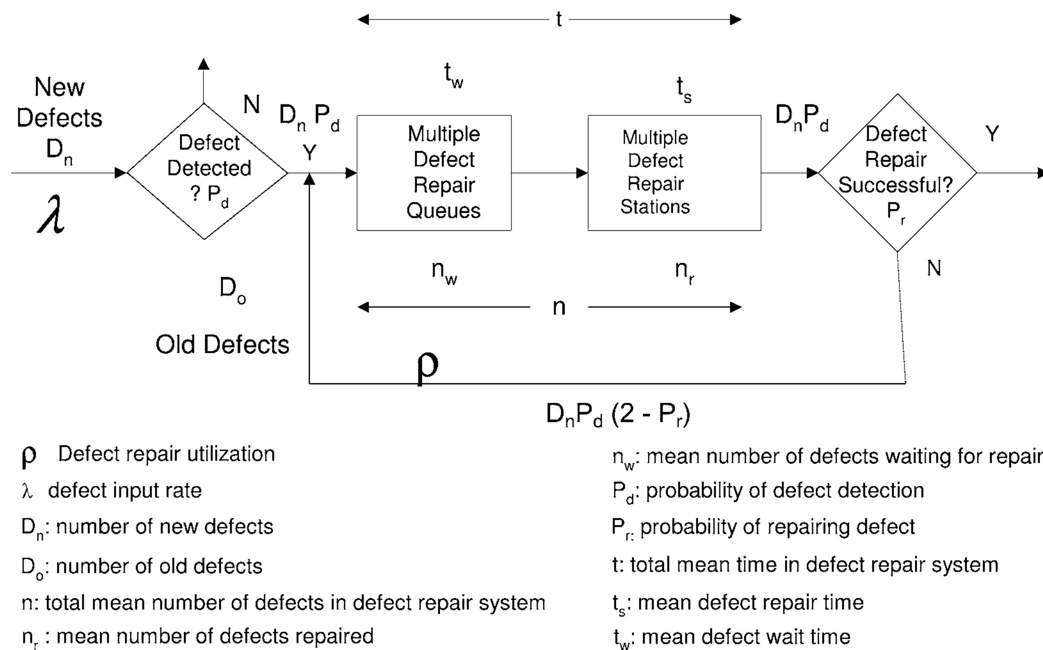

Fig. 1 Defect detection and repair process.

such as the *time required to correct defects*, have a direct bearing on product attributes, such as the *number of old defects* that remain unrepaired in Fig. 1.

The data used in this research was obtained from the NASA IV&V Facility Metrics Data Program data repository web site. Projects are given a pseudo name. It is not possible to identify the specific project because this information is not provided in the repository. The pseudo name of our data is the *JM1Data Set*. There are two types of data used. One is for a single module (see Table 1 for a summary of the results). In this case, defect processing data (e.g., time to correct defects) was collected for all the defects on a single module. The other type is comprised of data from 22 modules (see Table 2 for a summary of the results). In this case, defect count and metrics (source lines of code and cyclomatic complexity) data was collected for the 22 modules.

One reason we are motivated to do this research is to suggest that it is not necessary to wait until software is fielded before recording and analyzing defects. Assuming there is an effective data collection process in place, we can record actual defects and predict future defects for all software development phases. Unfortunately, there seems to be some confusion about how defects are processed. For example,[3] claims "that in industry, actual defect density of a software system cannot be measured until it has been put into production and has been used extensively by the end user. Actual defect density information as found by the end user becomes available too late in the software lifecycle to affordably guide corrective actions to software quality." This is not true in all cases because, as shown in,[4] the developers of the NASA Space Shuttle software analyze defects related to requirements changes throughout the life cycle. Although these developers focus on defects vice density, the latter could be computed from known or estimated software size. Furthermore, as described in,[5] defects can be defined and utilized for *all* development phases (see *Definitions* below).

A. Definitions

Software defect: This subtype includes all software defects that have been encountered or discovered by examination or operation of the software product. Possible values in this subtype are these:[5]

- Requirements defect: A mistake made in the definition or specification of the customer needs for a software product. This includes defects found in functional specifications; interface, design, and test requirements; and specified standards.

Table 1 Defect repair system attributes (JM1 Data Set)—Module 11181, except where noted.

Entity	New defects regression equation	New defects regression equation R^2	Values	Definition
sloc (Figure 2)	$D_n = .0121\ M_1 - 11.675$	0.9936	Correlation coefficient = 0.8449 for M_1 vs. M_2	Based on data for 22 modules
cyclomatic complexity	$D_n = .07411\ M_2 - 6.0303$	0.9919		Based on data for 22 modules
	How entity or value was determined			
$\bar{n}$	Result of data set analysis		17 defects	total mean number of defects in defect repair system
$\bar{n}_r$	Result of data set analysis		16 defects	mean number of defects repaired
$\bar{n}_w$	Computed from equation (1.12)		1 defect threshold = 10 defects	mean number of defects waiting for repair
$\Delta T_{min,max}$	Computed from data set data		1426 days	change in time from minimum time to next defect to maximum time to next defect
$\lambda = \bar{n}/\Delta T_{min,max}$	Computed from data set data		0.0119 defects per day	defect input rate
MTTD	Computed from equation (1.6)		117.75 days allowable value = 150 days	mean time to defect
P_r	Computed from equation (1.7)		0.9412	probability of defect repair
$\overline{\Delta Ci} = \bar{t}_s$	Computed from equation (1.9)		228.96 days threshold = 100 days	mean time required to repair defects
ρ	Computed from equations (1.10) and (1.14)		1.9444 using 1 station 0.6812 using 4 stations threshold =< 1.0	defect repair utilization
$\bar{t}_w$	Computed from equation (1.15)		84.03 days	mean defect wait time
$\bar{t}$	Computed from equation (1.16)		312.99 days	total mean time in defect repair system
$D_n P_d = 0.0121\ M_1 - 11.675$	Mean value of equation		48.23 defects	mean number of new defects detected
Mean [(Actual − D_n)/ Actual]			−0.0040 (sample size = 22)	Mean relative error of New Defects
Mean [(Actual − D_n)/ Actual]			−0.1530	Mean relative error of New Defects for the model in reference [MAR]
$D_o = D_n P_d (2 - P_r) = (0.0121 M_1 - 11.675) P_d)(2 - P_r)$	Mean value of equation		51.06 defects defects	mean number of old

- Design defect: A mistake made in the design of a software product. This includes defects found in functional descriptions, interfaces, control logic, data structures, error checking, and standards.
- Code defect: A mistake made in the implementation or coding of a program. This includes defects found in program logic, interface handling, data definitions, computation, and standards.

Table 2 Multiple module attributes for data set JM1.

Module j	Defect Count	Sloc M_{1j}	Cyclomatic complexity M_{2j}	Probability of detection for Module j (P_{dj}) $P_{dj} = 1 - \left(\frac{M_{1j}}{\sum_j M_{1j}}\right)$
11181	26	3442	470	0.8160
11182	20	1129	128	0.9397
11183	14	1824	268	0.9025
11184	10	222	19	0.9881
11185	15	844	404	0.9549
11186	11	1411	127	0.9246
11187	14	1532	263	0.9181
11188	10	466	94	0.9751
11189	13	1280	207	0.9316
11190	7	186	42	0.9901
11191	10	107	24	0.9943
11192	0	706	94	0.9623
11193	9	790	34	0.9578
11194	10	1882	286	0.8994
11195	12	657	104	0.9649
11196	8	322	82	0.9828
11197	7	128	20	0.9932
11198	9	725	173	0.9613
11199	3	334	33	0.9821
11200	0	20	2	0.9989
11201	4	621	25	0.9668
11202	6	82	11	0.9956
		$\sum_j M_{1j} = 18710$		Perfect linear relationship with M_{1j}

- Document defect: A mistake made in a software product publication. This does not include mistakes made to requirements, design, or coding documents.
- Test case defect: A mistake in the test case causes the software product to give an unexpected result.
- Other work product defect: Defects found in software artifacts that are used to support the development or maintenance of a software product. This includes test tools, compilers, configuration libraries, and other computer-aided software engineering tools.

Note: in the following definitions, some of the symbols are specific to defect repair processing (e.g., ΔC_i) and other are the usual symbols used in queuing analysis (e.g., t_s).

i: defect identification

j: module identification

M_k: metric k

M_1: sloc

M_2: cyclomatic complexity

D_n: number of new defects in defect repair system

D_o: number of old defects

T_i: time to next defect i

$\Delta T_{i,i+1}$: change in time from defect i to defect i + 1 = time to next defect

$\Delta T_{min,max}$: change in time from minimum time to next defect to maximum time to next defect

ΔC_i: time required to repair defect i (resolved date–opened date)

$\overline{\Delta Ci}$: mean time required to correct defects $= \bar{t}_s$
t_s: defect repair time
$\bar{t}_s$: mean defect repair time $= \overline{\Delta Ci}$
t_w: defect wait time (defect entry date–defect open date)
$\bar{t}_w$: mean defect wait time
$\bar{t}$: total mean time in defect repair system
MTTD: mean time to defect
λ: defect input rate
$\bar{n}$: total mean number of defects in defect repair system
$\bar{n}_r$: mean number of defects repaired
$\bar{n}_w$: mean number of defects waiting for repair
P_r: probability of defect repair
P_d: probability of defect detection (randomized for single module: 0, 1; computed for multiple modules)
ρ: defect repair utilization (for a stable defect repair system, $\rho < 1.0$)
In queuing parlance, ρ is the expected fraction of time a server is busy.[6]
N: number of repair stations (i.e., number of servers)

B. Rationale for Defect Detection and Repair Models

As stated in,[7] "software depends heavily on the defects in a software product and the repair activity undertaken to correct them." This is a succinct statement of the rationale for our development of a defect detection and repair model.

Furthermore, as reported in,[8] IBM took an important step in moving toward a goal and measurement-driven approach to defect elimination with a consistent use of defect models encompassing the entire software development life cycle. While they recognized that quality must be viewed as much more than merely an absence of defects, defect models allow them to view the entire process and to take a consistent view of gathering and tracking their data. It provided a means for project teams to track the data and, along with the project managers and release managers, set quality goals for each release. Creating a defect model allowed them to go back and take a more consistent view of the data so that they could evaluate their progress towards quality goals. They wanted to know how many defects they had and how they should evaluate their progress. They found that creating defect detection and repair models was a positive quality step in itself. It provided focus for the entire team on quality objectives, and gave reality to the magnitude of the quality challenge. As such, it readily became the basis for goal setting and concrete quality improvement initiatives.

Given that defect detection and repair models are such an important part of quality improvement in companies like IBM, we are encouraged to develop a model emphasizing the queuing aspects of defect detection and repair, using the NASA defect data to test our model.

C. Related Research

According to,[9] the "stopping rule" problem that involves determining an optimal release time for a software application has been addressed by several researchers. However, most of these research efforts assume instantaneous fault correction, an assumption that underlies many software reliability growth models, and hence provide optimistic predictions of both the cost at release and the release time. This researcher presents an economic cost model that takes into consideration explicit fault correction in order to provide realistic predictions of release time and release cost. As you can see in Figure 1 and in Table 1, we do *not* assume instantaneous fault correction. In fact, our model includes the time required to repair defects.

In,[10] models were considered that use the number of defects detected in the earlier phases of the development process as the independent variable. This number can be used to predict the number of defects to be detected later, even in modified software products. A strong correlation between the number of earlier defects and that of later ones was found. Using this relationship, a mathematical model was derived which may be used to estimate the number of defects remaining in software. This defect model may also be used to guide software developers in evaluating the effectiveness of the software development and testing processes. In contrast, our model focuses on the dynamics of the defect repair queuing process, as exhibited in Figure 1, to make explicit the delays involved in repairing defects

that queue up for service. Our purpose is to identify the need for possible faster repair in order to improve the quality of the software.

The aim of[11] was to develop a quality prediction model. This model, COQUALMO, predicts the defect density of the software under development where defects conceptually flow into a holding tank through various defect introduction pipes and are removed through various defect removal pipes. COQUALMO consists of 2 sub-models, namely the ‘Defect Introduction (DI)’ and the ‘Defect Removal (DR)’ models. The DI model is formulated using product, process, computer and personnel attributes (based on COCOMO) and predicts the number of requirements, design and coding defects that are introduced during various activities of the development life cycle. The residual number of defects is the difference between the number of defects introduced and the number of defects removed. We find this paper relevant and interesting because the author’s holding tank (our defect repair queue) and his pipes (our defect repair service) are analogous to a queuing system for defect processing (see Fig. 1).

This article[12] introduces the closed-loop defect removal model, a dynamic model that is used for software quality management and latent defect prediction. The author’s model considers injected defects and escaped defects separately. It gives the initial distribution of defects based on the organization’s baseline values for defect injection and goals for defect removal effectiveness across phases. Actual values for defects are fed into the model after every phase with additional indicators on defect injection and review effectiveness. Using statistical process control (SPC), specification limits are fixed for predicted values of defects. Estimates are revised when the actual defect count cross the defect specification limits. Causes of deviations are analyzed and preventive/corrective actions are taken. This enables better quality planning, control, and management.

Our model is consistent with the above approach in that we compare actual or predicted defect quantities with the desired limits, or thresholds, and if the bounds are exceeded, ascertain the cause and correct the process causing the discrepancy.

II. Model Equations

Using the above definitions, Figure 1, and Table 1, we develop the model equations:

$$\text{Defect input rate: } \lambda = \bar{n}/\Delta T_{min,max} = 0.0119 \text{ defects per day} \quad (1.1)$$

$$\text{New defects: } D_n = .0121M_1 - 11.675 \text{ (obtained from regression using } sloc\text{)} \quad (1.2)$$

$$\text{New defects: } D_n = .07411M_2 - 6.0303 \text{ (obtained from regression using cyclomatic complexity)} \quad (1.3)$$

$$D_nP_d = \text{number of new defects detected} = (.0121M_1 - 11.675)P_d \quad (1.4)$$

Note: henceforth, because the R^2 value for defects as a function of *sloc* is higher than for defects as a function of *cyclomatic complexity* (see Table 1), we will use equation (1.2) as a predictor of new defects. Another consideration for this choice is expressed in the following: “Size is the easily quantifiable software attribute that is most closely associated with the number of defects. The basic test of the effectiveness of complexity models and other indicators of defect-proneness is to ask, “Does this model show a significantly higher correlation with defects than just size (e.g., lines of code) alone?”.[13] What this statement means is that a defect prediction model using complexity as the predictor, may not have a significantly stronger relationship with defects than size alone.

Using Figure 1, we derive equation (1.5) for predicting old defects:

$$D_o = D_nP_d(2 - P_r) = (.0121M_1 - 11.675)P_d)(2 - P_r) \quad (1.5)$$

$$\text{Mean Time To Defect: MTTD} = \frac{\sum_{i=1}^{n} T_i}{n} = 117.75 \text{ days} \quad (1.6)$$

$$\text{Probability of defect repair: } P_r = \frac{\bar{n}_r}{\bar{n}} = 0.9412 \tag{1.7}$$

$$\text{Mean time to repair defects: } \overline{\Delta \text{Ci}} = \frac{\sum_{i=1}^{n} \Delta C_i}{n} = 228.96 \text{ days} \tag{1.9}$$

$$\text{Defect repair system utilization using 1 repair station: } \rho = \left(\frac{\sum_{i=1}^{n} \Delta C_i}{n}\right) \Big/ \text{MTTD}$$

$$= \left(\frac{\sum_{i=1}^{n} \Delta C_i}{n}\right) \Big/ \left(\frac{\sum_{i=1}^{n} T_i}{n}\right) = 1.9444 \text{ (infeasible since } \rho \text{ must be } <1) \tag{1.10}$$

Utilization is revised to $\rho = .6812$ using 4 repair stations (see Table 1).

Based on the concept that the probability of defect detection P_{dj}, for module j, in a set of multiple modules, will vary inversely with sloc, because it would be harder to detect defects in large modules than in small ones, we arrive at equation (1.11):

$$P_{dj} = 1 - \left(\frac{M_{1j}}{\sum_j M_{1j}}\right) \tag{1.11}$$

Of course, one could argue that if small modules are complex (e.g., high values of cyclomatic complexity), defects could be difficult to detect. However, in the case of our data, there is a high correlation between *sloc* and *cyclomatic complexity* (see Table 1).

Next, we wish to compute the mean number of defects waiting to be repaired. Figure 1 suggests how to do this. Equation (1.12) provides the answer:

$$\bar{n}_w = \bar{n} - \bar{n}_r = 17 - 16 = 1 \text{ defect} \tag{1.12}$$

III. Model Structure

The first aspect of the model structure is the form of the defect repair queuing system in Figure 1 and the key supporting equations that follow.

From queuing theory:[6]

For a multi server queuing system (see Figure 1):

$$\rho = \frac{(\lambda \bar{t}_s)}{N} = \frac{(\lambda \overline{\Delta \text{Ci}})}{N} \tag{1.13}$$

With N = 4, and substituting data from Table 1:

$$\rho = \frac{(\lambda \overline{\Delta \text{Ci}})}{N} = \frac{[(.0119)(228.96)]}{4} = .6812 \tag{1.14}$$

This is a reasonable value for the defect repair system utilization. If N = 1, $\rho = 1.9444$ in Table 1, which exceeds the allowable utilization for a stable queue system (i.e., if $\rho > 1$, the defect input rate exceeds the defect service rate).

Another key queuing equation involves the computation of the mean time that defects have to wait to be repaired, given by equation (1.15), which uses equation (1.12):

$$\text{Mean defect wait time} = \bar{t}_w = \frac{\bar{n}_w}{\lambda} = \frac{1}{0.0119} = 84.03 \text{ days} \tag{1.15}$$

Then, we can go on to compute the total mean time defects spend in the defect repair system by using equation (1.16):

$$\bar{t} = \bar{t}_s + \bar{t}_w = 228.96 + 84.03 = 312.99 \text{ days} \tag{1.16}$$

The second aspect of the model structure is the question of whether there should be a single probability of defect detection for all types of defects or whether there should be a different probabilities for each type. The former would be easier to model, but it would be unrealistic because as pointed out in,[14] some defects may be easier to detect than others, and, conversely, some may be more difficult to detect than others. A contrary view is expressed in,[15] as follows: “Defect insertion and detection rates tend to remain relatively constant as long as the project’s software processes remain stable. While the rates are not exactly constant, they perform within a recognized range.” The key to this statement is “remain stable”. Since we want our model to be flexible and to accommodate all defect detection events, we choose to use a variable defect detection probability. Therefore, we opt for randomizing the probability of defect detection P_d between 0 and 1 for a *single module*. This, then, allows us to predict the number of new defects detected $D_n P_d$ and the number of old defects Do in equations (1.4) and (1.5), respectively.

IV. Analysis of Prediction Results

Examination of prediction results allows software quality decision makers to judge the quality of defect prediction models and to use the models for anticipating events that could significantly affect the operation of their software quality process. Some examples follow.

1. A comparison was made of the accuracy of predicting *new defects* for our model versus the model reported in.[16] As can be seen in Table 1, our model fared much better. However, we hasten to add that this comparison

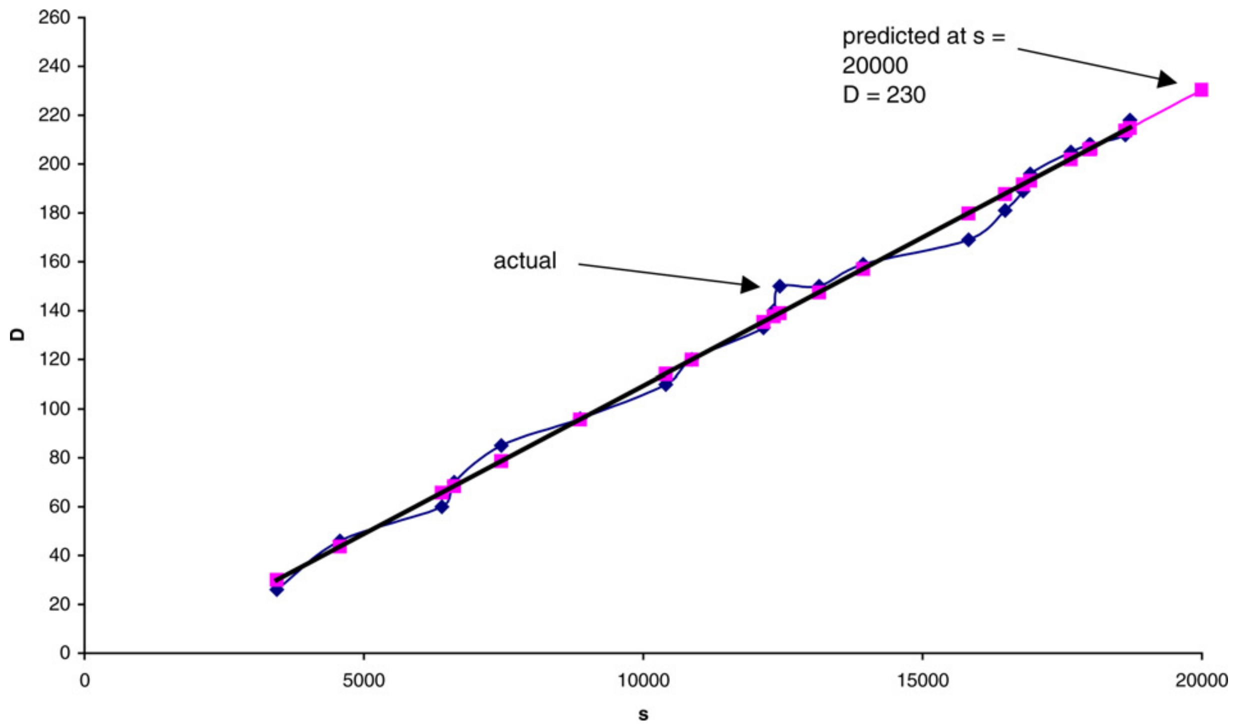

Fig. 2 Cumulative defects (D) vs. Cumulative sloc (s) (JM1 data set).

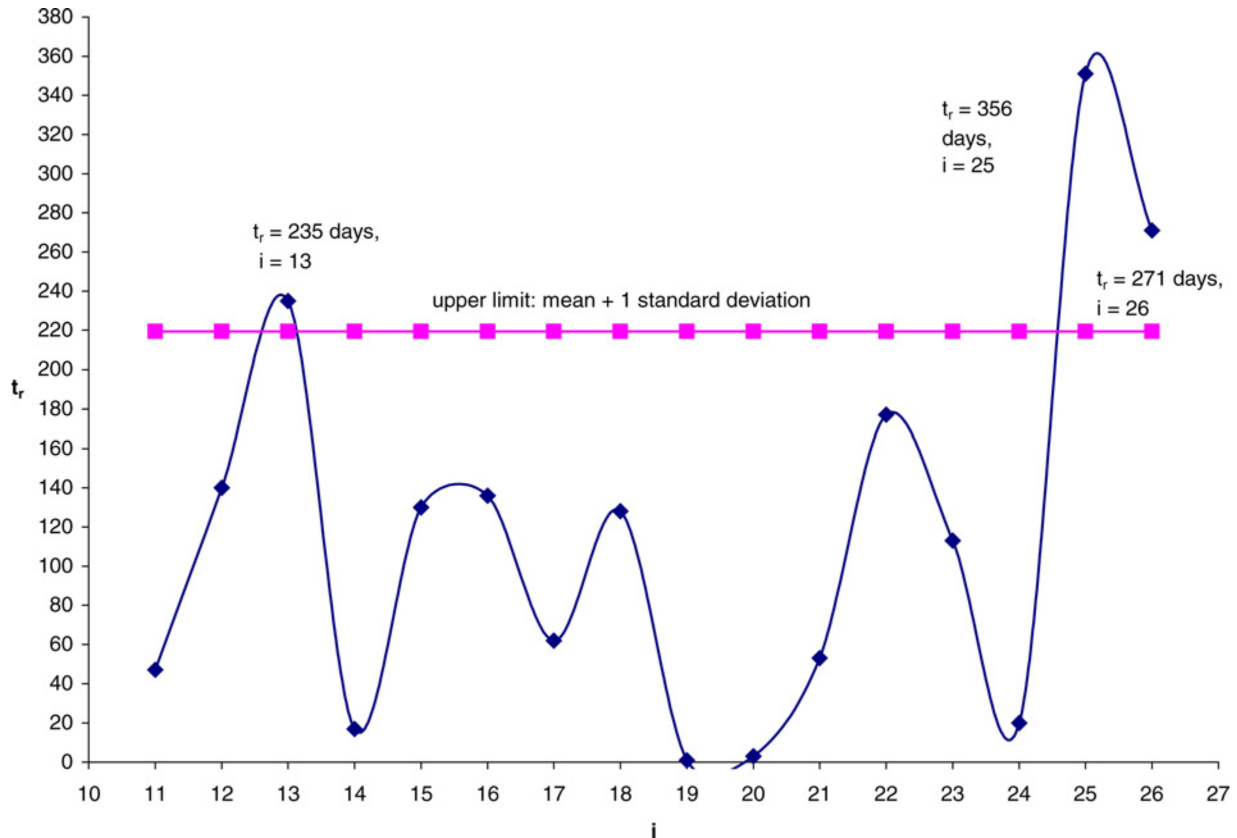

Fig. 3 Control chart for time to repair defects t_r vs. defect identification i (Module 11181, JM1 data set).

is only valid for *this particular* prediction. It should not be inferred that our model would be superior for all types of predictions.

2. A prediction was made of the number of new defects D_n for a cumulative value of $sloc = 20,000$ in Fig. 2. This value of *sloc* is used to predict the value of $D_n = 230$ that would ensue if a value of *sloc* were to occur that is not in the *JM1Data Set*. This type of prediction is important for *anticipating* rather than reacting to events. That is, in this case, we would not want to be surprised by an increase in *sloc* to 20,000 that could lead to *new defects* rising to 230 that might be unacceptable to NASA and its customers.
3. An analysis was made about how Statistical Process Control (SPC) could be applied to NASA software development processes. We used the concept, developed in[17], that a process has one or more outputs, such as the *time required to repair defects*. SPC is based on the idea that these outputs have two sources of variation: random and assignable cause. If the observed variability of the outputs of a process is within the expected range of random variation, the process is said to be under statistical control. The software engineer tracks the variability of the process to be controlled. When this variability exceeds the range to be expected from random variation, he or she then identifies and corrects assignable causes. We determined whether *time required to repair defects* is under statistical control. The result is shown in Fig. 3, where *times required to repair defects* that are not under control are identified as those values that exceed the upper limit. In this example, we would assign the cause as an insufficient number of defect repair stations, as previously mentioned, when it was determined that *mean time required to repair defects* was excessive. Another observation is that all three of the defects whose *time required to repair defects* exceed the control limit—$i = 13, 25, 26$—have a severity value = 3. This severity level adversely affects the performance of the software system. Thus, SPC can be

employed to identify both defect quantities that are out of bounds—*time to repair defects*—and associated data (defect severity).

V. Defect Processing Policy Issues

An application of our model is to illuminate the many issues that software engineers may want to consider regarding defect processing. To a great extent, this involves interpreting differences between predicted or actual defect counts and the thresholds (i.e., allowable values for achieving reliability goals). In addition to interpreting results, this analysis is geared to taking remedial action when actual or predicted quantities are out of bounds.

1. As explained in,[15] "During project execution, planned defect levels are compared to actual defect levels. Typically, this occurs at major phase transitions (milestones). Since real performance never exactly matches the plan, the differences must be investigated." For example, consulting the following question might arise: What is the mean number of defects waiting to be repaired? Since we see the answer is 1 and the threshold is 10 in Table 1, the software passes this quality check.
2. If we focus on defect repair utilization, we see a very unsatisfactory situation. Investigation of this problem would reveal that there is an insufficient number of defect repair stations—one! The solution is found in equation (1.14), where $n = 4$ stations was found to provide acceptable utilization. Another cause for concern is the excessive *mean time required to repair defects* of 228.96 days versus the threshold of 100 days shown in Table 1. A possible response to this problem would be to determine whether, for example, the delay is caused by an insufficient number of defect repair stations, or some other factor.
3. According to,[18] quality measurement models usually include, among other things, simple progress tracks of defects detected. Fig. 4 depicts a typical defects detected tracking chart for a single module. The software engineer would be interested in knowing whether the track is stabilizing over time or whether it is increasing at an increasing rate. We see that at later points in time, the track is increasing at a linear rate, which would suggest that the quality process for this application is under control. In addition to the control aspect, the figure tells the engineer that defect # 18 has not been repaired; this would trigger a repair action. Finally, as suggested by,[19] the knee of the curve shown in the figure serves as demarcation point for releasing the software (i.e., the defect detection process stabilizes). This policy is satisfactory as long as defect # 18—in this case—is repaired prior to release!
4. Another aspect of Fig. 4 that is of interest is the heights and lengths of the steps in the plot. According to,[20] the lengths represent the delays between defect detection and repair and the heights represent the number of defects delayed. In reviewing Fig. 4, we see that prior to the knee of the curve, there would be concern because there is a large step that includes defect # 18 that has not been repaired. Later, after the knee has been passed, we observe that the occurrence of *new defects* stabilizes.
5. A metric of great interest pertinent to the efficiency of the defect repair process is the probability of defect repair P_d, computed using equation (1.7). In Table 1, we see that this value is 0.9412. This metric can be considered to be the *defect removal effectiveness* that has been related to Capability Maturity Model levels. For example,[21,22] have found that *defect removal effectiveness* was 93% and 95% in level 4 and level 5 organizations, respectively. The significance of this point is twofold: 1) software organizations can use this metric as *one* of the goals for achieving high maturity levels and 2) the metric can be used as *one* of the criterions for assessing the stability of a software organization's development process. Thus, in the case of $P_d = 94.12\%$, there is *one* data point that suggests this NASA software organization has achieved a high level of maturity.
6. Heretofore, in 1–5 above, we have discussed policy questions, using data from a single module. Now, we shift the emphasis to analyzing data from 22 modules. Suppose the software engineer wants to know whether it would be easier to detect defects in simple, small modules as opposed to complex, large modules. Table 2 would serve as a guide. Using equation (1.11), we computed the probability of defect detection P_{dj} for each module. Table 2 reveals some very interesting results: the highest probability corresponds to the module with the lowest values of defect count, source lines of code, and cyclomatic complexity. Conversely, the module with the lowest probability corresponds to the module with the highest values of these attributes. Thus, the engineer would be encouraged by these results to use these relationships as a guide to predicting the probability of detection of defects in other software systems with similar attributes.

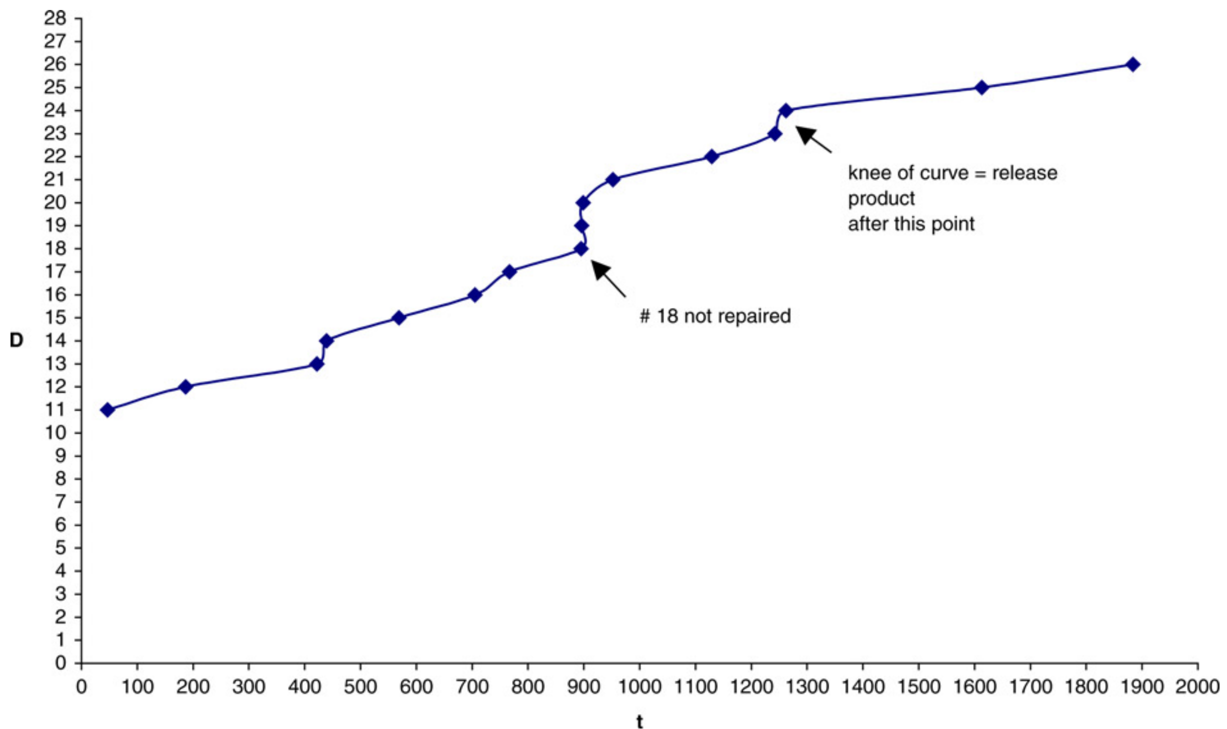

Fig. 4 Tracking of cumulative defects detected D as a function of cumulative numbers of days since detection *t* (Module 11181, JM1 data set).

VI. Conclusions and Future Research

The main benefit for the software engineer of using this model's predictions is for anticipating rather than reacting to events. Secondarily, the model's predictions can be used in statistical process control to determine whether defect processing variables are within limits or out of control. In addition, the model can be used to provide answers to questions like the following:

Does the mean number of defects waiting to be repaired exceed the threshold?

Does the mean time to repair defects exceed the threshold?

Does the track of defect detection stabilize over time or is it increasing at an increasing rate?

Does the occurrence of new defects stabilize over time?

Is the defect removable process satisfactory?

Can the probability of defect detection be predicted and can the methodology of prediction, developed for one project, be applied to other, similar projects? The answer to the first part of this question is "yes", but the answer to the second part would have to wait for the results from future research that will apply the model to other NASA software projects. Another important research effort will be to develop a comprehensive database of NASA defect data that correlates the various defect attribute for easy retrieval and analysis.

References

[1]Norman F. Schneidewind, "Modelling the Fault Correction Process", *Proceedings of The Twelfth International Symposium on Software Reliability Engineering*, Hong Kong, 27-30 November, 2001, pp. 185–190.

[2]Norman F. Schneidewind, "Applying Fault Correction Profiles," *Proceedings of the 28th Annual NASA Goddard Software Engineering Workshop* (SEW'03), 2003, pp. 185–192.

[3]Dolbec, J. and Shepard, T., "A Component Based Software Reliability Model," presented at Conference of the Centre for Advanced Studies on C, 1995.

[4]Norman F. Schneidewind, "Predicting Risk as a Function of Risk Factors", *The R & M Engineering Journal*, American Society for Quality, Vol. 25, No.1, 1 Mar. 2005.

[5]William A. Florac with the Quality Subgroup of the Software Metrics Definition Working Group and the Software Process Measurement Project Team, "Software Quality Measurement: A Framework for Counting Problems and Defects", Technical Report, CMU/SEI-92-TR-022, ESC-TR-92-022, Sep. 1992.

[6]Fredrick S. Hillier and Gerald J. Lieberman, *Introduction to Operations Research*, 7th ed., McGraw Hill, 2001.

[7]John D. Musa, "Anthony Iannino, and Kazuhira Okumoto", *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, 1987.

[8]David D. Brown and Jeffrey S. Poulin, "Metrics-Based Defect Prevention In MVS", International Business Machines Corporation, 15 Sep. 1993.

[9]Swapna S. Gokhale, "Optimal Software Release Time Incorporating Fault Correction," *Proceedings of the 28th Annual NASA Goddard Software Engineering Workshop* (SEW'03), 2003, pp. 175–184.

[10]T.-J. Yu, V.Y. Shen, and H.E. Dunsmore, "An Analysis of Several Software Defect Models", *IEEE Transactions on Software Engineering*, Vol. 14, No. 9, Sep. 1988, pp. 1261–1270

[11]Sunita Chulani, "Constructive Quality Modeling for Defect Density Prediction: COQUALMO", IBM Research, Center for Software Engineering, *International Symposium on Software Reliability Engineering*, Fast Abstracts, 1999.

[12]Achamma Jose, Anju, N. K., and Pillai, S. K., Network, Systems and Technologies (P) Ltd., "Software Processes: Closed-Loop Defect Removal Model Using Statistical Process Control", *Software Quality Professional*, Vol. 3, No. 1, Dec. 2000.

[13]Card, David, and William Agresti. "Resolving the Software Science Anomaly", *Journal of Systems and Software* Vol. 7, 1990, 29–35.

[14]Lionel C. Briand, Khaled El Emam, Bernd G. Freimut, Oliver Laitenberger, and Fraunhofer, "A Comprehensive Evaluation of Capture-Recapture Models for Estimating Software Defect Content", Institute for Experimental Software Engineering, International Software Engineering Research Network Technical Report ISERN-98-31.

[15]David N. Card, Software Productivity Consortium, "Managing Software Quality With Defects 1", *Crosstalk*, Mar. 2003 Issue.

[16]Martin Neil, Software Risk Assessment, Week 9–Software Defect Prediction using Advanced, Models, DCS337, ITEM044, AMCM055.

[17]"Software Acquisition Gold Practice, Statistical Process Control, Focus Area: Quality–Measurement", The Data and Analysis Center for Software.

[18]Kevin Domzalski, BAE Systems with assistance from David Card, Q-Labs, "The Measurement Challenge of High Maturity", The Dod Software Tech News, Vol. 9, No. 1, Mar. 2006.

[19]Ram Chillarege, "Orthogonal Defect Classification", *Handbook of Software Reliability Engineering*, edited by Michael R. Lyu, IEEE Computer Society Press, 1996, Chap. 9.

[20]John D. Musa, *Software Reliability Engineering: More Reliable Software Faster and Cheaper*, 2nd. ed., McGraw-Hill, 1999.

[21]Software Productivity Research, Quality, and Productivity of the SEI CMM, *Software Productivity Research*, 1994.

[22]C. Jones, *Software Assessments, Benchmarks, and Best Practices*, Addison-Wesley, 2000.

Michael Hinchey
Editor-in-Chief