Estimating the Number of Defects: A Simple and Intuitive Approach

Michael Naixin Li
Microsoft Corp.
One Microsoft Way
Redmond, WA 98052-6399
naixinli@microsoft.com

Yashwant K. Malaiya & Jason Denton
Computer Science Dept.
Colorado State University
Fort Collins, CO 80523
malaiya|denton@cs.colostate.edu

Abstract

The number of defects is an important measure of software quality which is widely used in industry. Unfortunately, accurate estimation of defect density can be a difficult task. Sampling techniques generally assume that the faults found are a representative sample of the all existing faults, which results in inaccurate estimates. Other existing techniques provide little information in addition to the number of faults already found.

Software test coverage tools can easily and accurately measure the extent to which the software has been exercised. Both testing time and test coverage can be used as measures to model the defect finding process. However test coverage is a more direct measure of test effectiveness and can be expected to correlate better with the number of defects found.

Here we describe a simple and intuitive procedure which can be used to estimate the total number of residual defects, once a suitable coverage level has been achieved. The technique is consistent with common testing approaches used. The method will be illustrated using actual data and is compared with existing approaches. Our results show that the method yields consistent estimates. An enhanced version of this approach is being implemented in a GUI tool.

1 Introduction

The number of residual defects is among the most important measures of software reliability. Leading edge software development organizations typically achieve a defect density of about 2.0 defects/KLOC [1] and some now use a lower target. The NASA Space Shuttle Avionics software with an estimated defect density of 0.1 defects /KLOC is regarded to be an example of what can be currently achieved by the best methods [5]. A low defect density can be quite expensive to achieve, the Space Shuttle code has been reported to have cost about $1,000 per line of code. The cost of fixing a defect later can be several orders of magnitude higher than during development, yet a program must be shipped by some deadline dictated by market considerations. This makes the estimation of the number of remaining defects a very important challenge. One conceivable way of knowing the exact defect density of a program is to actually find all remaining defects. This is obviously infeasible for any commercial product. Even if there are resources available, it will take a prohibitive amount of time to find all bugs in a large program [2]. Sampling based methods have been suggested for estimating the number of remaining defects. They assume that the faults *found* have the same testability as faults *not found*. However, in actual practice, the faults not found represent faults that are harder to find [13]. Thus such methods are likely to yield an estimate of faults that are relatively easier to find, which will be less than the true number.

It is possible to estimate the defect density based on past experience using empirical models like the Rome Lab model [7] or the model proposed by Malaiya and Denton [9]. The estimates obtained by such models can be very useful for initial planning, however these models are not expected to be accurate enough to compare with methods involving actual test data.

Another possible way to estimate the number of faults is by using the Exponential SRGM. In this model the parameter β_0 represents the total number of defects that would eventually be found. We can estimate the number

of remaining defects by subtracting the number of defects found from the value of β_0 obtained by fitting. We will evaluate this approach by comparing it with a new approach presented here.

An SRGM relates the number of defects found to the testing time spent. In actual practice, the defect finding rate will depend on the test effectiveness that can vary depending on the test input selection strategy. A software test coverage measure (like block coverage, branch coverage, P-use coverage etc.) directly measures the extent to which the software under test has been exercised. Thus we can expect that a suitably chosen test coverage measure can correlate better with the number of defects encountered. The relationship between test coverage and the number of defects found has been investigated by Piwowarski, Ohba and Caruso [15], Hutchins, Goradia and Ostrand [6], Malaiya et al [12], Lyu, Horgan and London [8] and Chen, Lyu and Wong [3].

In the next section, a method for estimating the defect count using test coverage is introduced and its applicability is demonstrated using test data. In section 3 we use the model to estimate the number of defects and compare the results with those obtained using the exponential SRGM. Finally we present some observations on this new approach.

2 Coverage and Defect Count

Software Reliability Growth Models (SRGMs) are well known and are used in many organizations. The SRGMs assume that there is a relationship between the total number of defects found and the time spent on testing. The SRGMs generally assume that testing is uniform and thus $\mu(t)$, the number of faults found increases smoothly. In actual practice testing can be uneven resulting in the behavior shown in Figure 1a. For a single test suite applied during time $(0,t_1)$ $\mu(t)$ starts showing saturation because the new tests are not effective in exercising untested parts of the code. When testing switches to a new test suite at time t_1, there is a jump in the number of defects being found, because the new suite tends to exercise some sections of code better than the first suite. Thus the actual behavior would vary with test effectiveness.

If we plot a coverage measure like branch coverage against time, then we would see a similar pattern. When testing is not very efficient, fewer new branches are covered. Again when a different test suite is applied starting at time t_1, we see a jump in new branches being covered as shown in Figure 1b.

Coverage is a more direct measure of the extent to which software has been exercised than the testing time. If we plot the number of defects against coverage, than we should expect to see a behavior that does not depend on test effectiveness, because this dependence is removed by directly considering actual coverage. Using coverage solves another practical problem, the measurement of testing time t. Because of holidays, multiple projects etc. it is not trivial to convert calendar time to actual testing time.

What would a plot of defects found against test coverage look like? Let us consider branch coverage as an example. Let us assume that there is some association of branches and defects such that when a specific branch is exercised, the related defects are likely to become exposed. If we assume a uniform distribution of defects among branches, then we should get a linear plot between defects found and branch coverage. However at any point in the test phase, the software has already been tested to some extent. It is likely that the defects associated with branches that are more reachable have already been found and removed. Thus at the beginning, even though coverage increases with time, very few defects would be found. At higher coverage, we will start finding defects resulting in the curve given in Figure 1c.

It is clear that for software with a lower defect density, the onset of defect finding will occur at higher coverage levels. That means that for lower defect densities, the curve in Figure 1c, will sink lower, leaving behind a smaller part of the curve to be encountered.

There is mathematical justification for this behavior. It has been found in some studies that the Logarithmic Poisson SRGM often has good predictive capabilities. If we assume that the model that is applicable for defects found is also applicable for the branches covered, then it can be shown [10] that we will see the behavior shown in Figure 1c. The value around which the curve exhibits a knee, has a significance as we will see below.

Applicability of this model is illustrated by the plots in figures 2, 3, and 4. This data was collected experimentally by Pasquini et al [14]. They tested a 6100 line C program by applying 20,000 test cases. The test coverage data was collected using the ATAC tool. Figure 2 shows a screen in ROBUST, an integrated software reliability evaluation tool suite [4] that has been developed at CSU. Further development of this tool suite is underway to include additional capabilities.

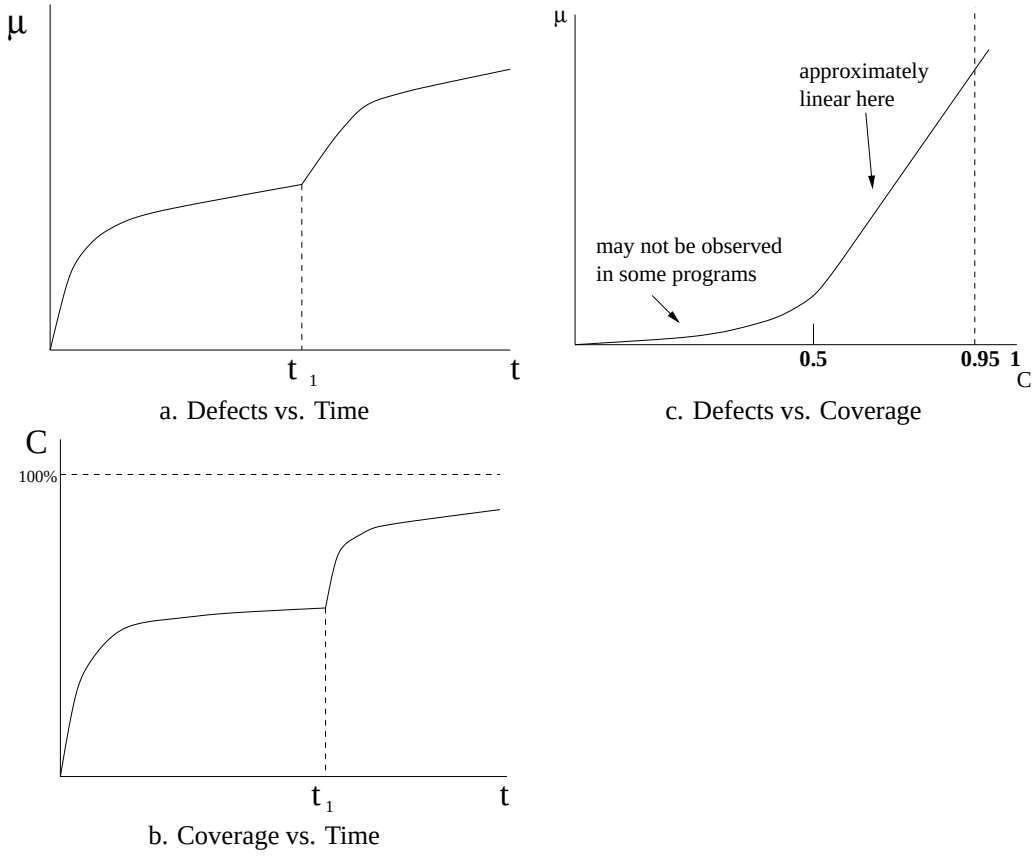

Figure 1: Defects found and Coverage relationship

For the 20,000 tests, these were the coverage values obtained: block coverage : 82.31% of 2970 blocks, decision coverage : 70.71% of 1171 decisions, and p-use coverage 61.51% of 2546 p-uses. This is to be expected since p-use coverage is the most rigorous coverage measure and block coverage is the least. Complete branch coverage guarantees complete block coverage, and complete p-use coverage guarantees complete decision coverage.

Also we note that the fitted model becomes very linear after the knee in the curve. Let us define C_{knee} as the knee, where the linear part intersects the x-axis. For block, branch and p-use coverage it occurs at about 40%, 25% and 25% respectively. Below we see the significance of this value.

The linear coverage model provides us a new way to estimate the total number of defects. As we can see in Figures 2, 3 and 4, which use the data obtained by Pasquini et al., the data points follow the linear part of the model rather closely. If we assume that all of the code is reachable, then the total number of defects would be given by the point in the curve where coverage equals unity. Because of linearity of the model, the projected number of total defects can be very stable. For example, the plot in Fig. 2 suggest that 100% block coverage would uncover 40 defects. If all of the code is reachable, we can expect that the actual total number of faults to be more than 40. In actual practice we need to take into account the existance of unreachable code as discussed below.

3 Estimation of Defect Count

The linear part of the curve for the number of defects found can be given as

$$\mu(C) = A_0 + A_1 C, \;\; C > C_{knee} \tag{1}$$

where A_0 and A_0 are parameters. Using the interpretation of the parameters through the Logarithmic model, it can be shown that [10, 11],

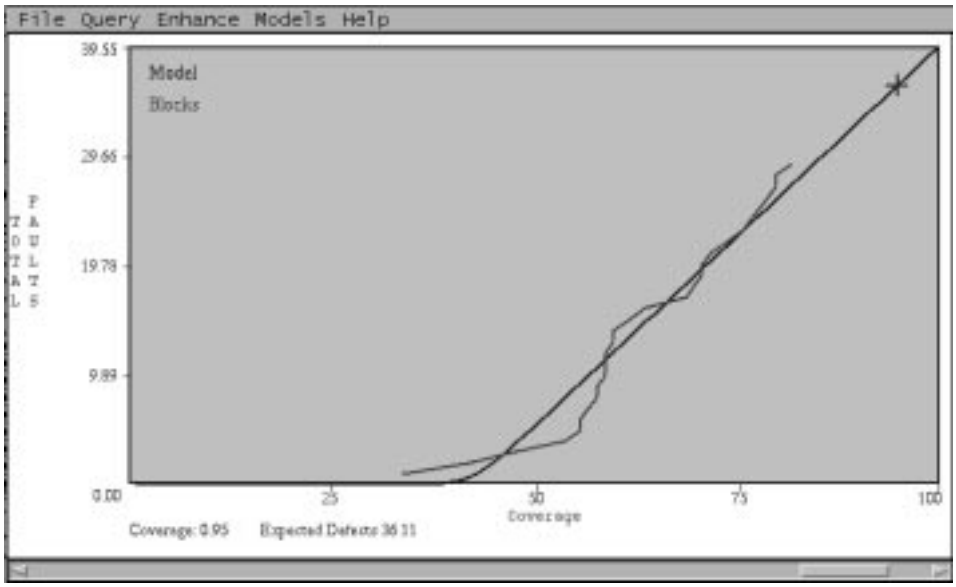

Figure 2: ROBUST: Block Coverage data (Pasquini et al.)

$$C_{knee} = 1 - (\frac{E_{min}}{D_{min}E_0})D_0 \tag{2}$$

Where E_{min}, D_{min}, E_0 are parameters and D_0 is the initial defect density. Thus for lower defect densities, the knee occurs at higher test coverage. This has a simple physical interpretation. If a program has been previously tested resulting in a lower defect density, it is likely that the enumerables with higher testability have already been exercised. This means that testing will start finding a significant number of additonal defects only after higher test coverage values are achieved. The problem of estimation of the parameters A_0 and A_1 is considered in detail in [10, 11].

In large programs, some small parts of the code are ofen unreachable. A DEC study suggests that in C programs it can be about 5% of the total code [16]. If we take the view that the dead code can be ignored, complete coverage of the reachable code corresponds to 95% coverage of the overall code in such a case. For accurate estimates, the dead code should be eliminated or the fraction of such code should be carefully estimated.

Let us consider the data reported by Pasquini [14]. Here for the purpose of illustration, let us assume that the software contains 5% unreachable code. Both the experimental data and the model suggest that the 95% coverage eventually achieved should uncover the number of faults as given in Table 1 below. The numbers in the last column have been rounded to nearest integer.

Table 1: Projected number of total defects with 95% coversage

Coverage measure	Total defects found	Coverage achieved	Defects expected with 95% coverage
Block Coverage	28	82%	36
Branch Coverage	28	70%	44
P-uses Coverage	28	67%	48

It should be noted that for this project 1240 tests revealed 28 faults. Another 18,760 tests did not find any additional faults, even though the presence of at least 5 more faults were known. The data suggests that the enumerables (blocks, branches etc.) not covered by the first 1240 tests were very hard to reach. They perhaps belong to sections of the code intended for handling special situations.

There is no coverage measure such that 100% coverage will assure detection of all the defects. Thus in the above table, the entry in the last column is a low estimate of the total number of defects actually present. Using a

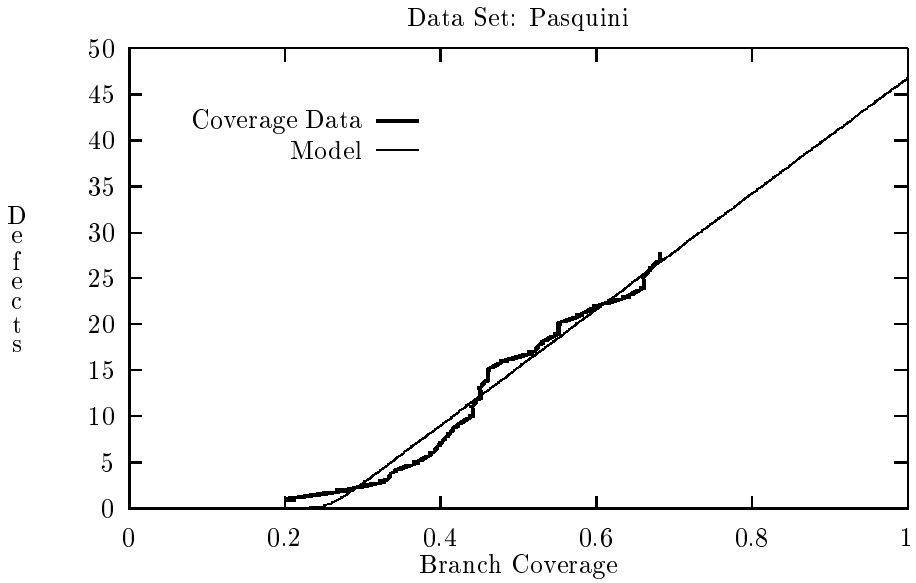

Figure 3: Defects vs. % Branch Coverage

more strict coverage measure raises the low estimate closer to the actual value. Since among the three measures the P-use coverage measure is most strict, the estimate of 48 faults using P-use coverage should be closer to the actual number than the estimates provided by block coverage.

4 Evaluation of Predictive Capability

As mentioned before, it is possible to obtain an estimate of defect density from the exponential SRGM. The parameter β_0 of this model represents the total number of faults in the system, and can be determined by fitting the available data to the model. Figure 5 shows the fitted value of β_0 for Pasquini et al's P-use data as testing progresses, assuming there is no unreachable code.

First, towards the end of testing the exponential model consistently predicts that the total defects present is the same as the number of defects found. As figure 5 shows, defects are still being found after the point where the model predicts zero residual defects. This means that in the later stages of testing, the exponential model provides no useful information about the remaining defects. Second, the predictions made by the exponential model never stabilize. If the estimates produced by the exponential model where accurate, then they should eventually begin to converge to some value, but this does not seem to be the case.

Figure 6 shows the estimates for total defects made by our model as testing progresses, based on Pasquini et al's data. The extended model [10] is used to handle the early points. Like the exponential model, the predictions initially made by our model rise quickly. After about 20 test cases however, they begin to take on stable values, maintaining consistent estimates of the total defects as more data comes in. We see similar results with other data, as shown in [10]. This stability is quite remarkable considering that the defect finding rate fluctuates considerably.

5 Conclusions and Observations

We have presented a new method for estimating the number defects based on coverage. At sufficiently high coverage, the linear model provides a very good description of actual data. Our method provides stable projections early in the development processes. The choice of coverage measure has an effect on the projections made. Results show that a strict coverage measure such as P-uses gives the most accurate results.

Further data collection and analysis is needed to evaluate accuracy of this approach at very low defect densities. There are no strong reasons to think that a deviation from the linear behavior would occur. It is unlikely that

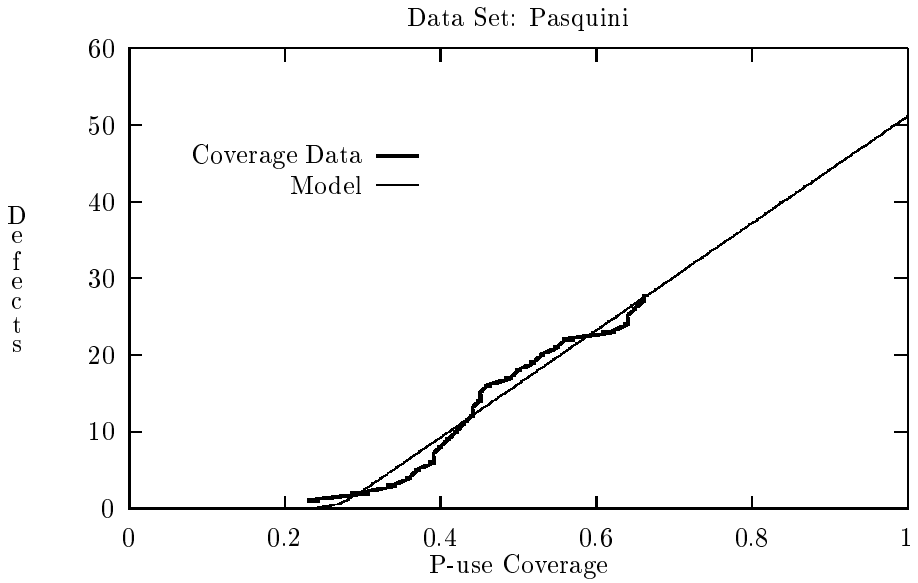

Figure 4: Defects vs. % P-use Coverage

the hardest to reach branches would have fewer faults associated with them, thus we should not expect to see a saturation of the linear plot.

The data sets used in this paper are for programs that are not evolving and thus no new defects are being added. The variation in the estimate of the total number of faults arise due to use of additional test data. For evolving programs, the method presented here would need to be extended to take into account the introduction of additional defects.

6 Acknowledgement

This work was supported in part by a BMDO funded project monitored by ONR, and in part by an AASERT funded project. We would like to thank Mladen Vouk for providing us some of the data used in this study.

References

[1] R. V. Binder. Six sigma: Hardware si, software no! *http://www.rbsc.com/pages/sixsig.html*, 1997.

[2] R. W. Butler and G. B. Finelli. The infeasibility of quantifying the reliability of life-critical real-time software. *IEEE Transactions on Software Engineering*, 19(1):3–12, January 1993.

[3] M. Chen, M. R. Lyu, and W. E. Wong. An empirical study of the correlation between coverage and reliability estimation. In *Proc. IEEE International Symposium on Software Metrics*, Berlin, Germany, March 1996.

[4] J. A. Denton. *ROBUST, An Integrated Software Reliability Tool*. Colorado State University, 1997.

[5] L. Hatton. N-version design versus one good design. *IEEE Software*, pages 71–76, Nov./Dec. 1997.

[6] M. Hutchings, T. Goradia, and T. Ostrand. Experiments on the effectiveness of data-flow and control-flow based test data adequacy criteria. In *Proc. International Conference on Software Engineering*, pages 191–200, 1994.

[7] P. Lakey and A. Neufelder. *System and Software Reliability Assurance Notebook*. Rome Laboratory, Rome, New York, 1997.

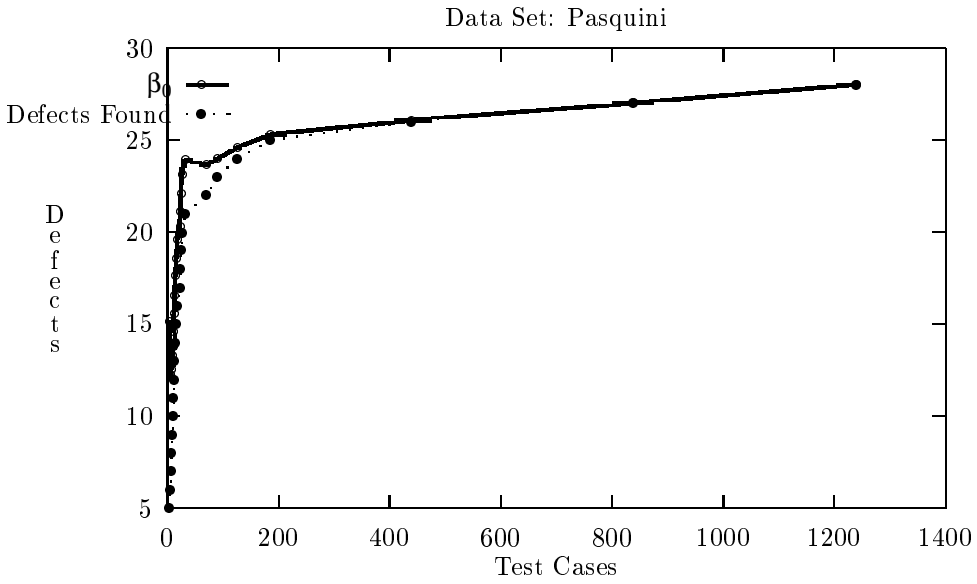

Figure 5: Estimated total defects using exponential model (Pasquini data)

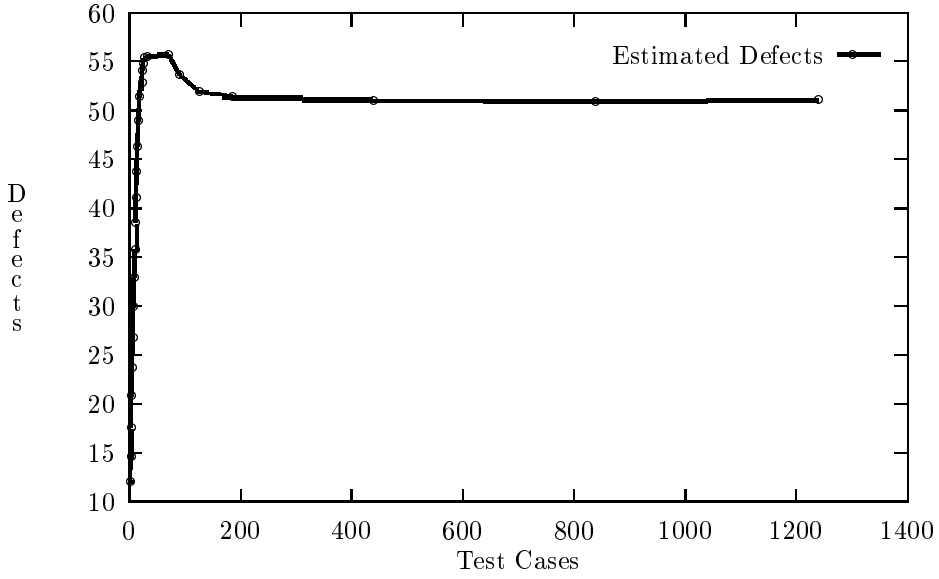

Figure 6: Estimated total defects with new approach (Pasquini data)

[8] M. R. Lyu, J. R. Horgan, and S. London. A coverage analysis tool for the effectiveness of software testing. In *Proc. IEEE International Symposium on Software Reliability Engineering*, pages 25–34, 1993.

[9] Y. K. Malaiya and J. A. Denton. What do software reliability parameters represent? In *Proc. International Symposium on Software Reliability Engineering*, pages 124–135, Albuquerque, NM, November 1997.

[10] Y. K. Malaiya and J. A. Denton. Estimating defect density using test coverage. Technical Report 98-104, Colorado State University, Ft. Collins, CO, 1998.

[11] Y. K. Malaiya and J. A. Denton. Estimating the number of residual defects. To appear in *Proc. IEEE High Assurance Systems Engineering Symposium*, Nov. 1998.

[12] Y. K .Malaiya, N. Li, J. Bieman, R. Karcich, and B. Skibbe. The relationship between test coverage and reliability. In *Proc. International Symposium on Software Reliability Engineering*, pages 186–195, November 1994.

[13] Y. K. Malaiya and S. Yang. The coverage problem for random testing. In *Proc. IEEE International Test Conference*, pages 237–245, October 1984.

[14] A. Pasquini, A. N. Crespo, and P. Matrella. Sensitivity of reliability growth models to operational profile errors. *IEEE Transactions on Reliability*, pages 531–540, December 1996.

[15] P. Piwowarski, M. Ohba, and J. Caruso. Coverage measurement experince during function test. In *Proc. 15th International Conference on Software Engineering*, pages 287–300, May 1993.

[16] A. Srivastava. Unreachable procedures in object-oriented programming. *LOPLAS*, Vol. 1, No. 4, pages 355–364, 1992.