# Exploration into Costs and Mitigation Strategies for Invalid Defects in Systems Development

Keith N. Mort

Marist College

Department of Information Systems

Poughkeepsie, NY

keith@mort.net

## Abstract

Organizations that are developing information systems not only have to deal with real defects, but they must also deal with a large number of invalid defects, which are not actually the fault of the software that the defect was reported against. Studies show that a vendor can expect an average of two invalid defect reports for every valid defect and 70% of all bugs will be found by more than one user, resulting in duplicate defect report. The cost of dealing with invalid defect reports can exceed 25-52% of the total cost of fixing defects in addition to causing delays in the test cycle and damaging user satisfaction. This paper discusses how even relatively unsophisticated machine learning classification models are easily able to surpass 81% accuracy in identifying invalid defects and explore the benefits of leveraging these models to triage new defect reports. Ongoing considerations are discussed which hopefully can inspire additional research in the area of invalid defect mitigation.

## 1 Introduction

The goal of the testing phase in the SDLC is to eliminate as many high impact residual defects as possible in order to ensure that the software is production ready [10,12]. According to Boehm and Basili, maintenance costs consume over 70% of the total cost of software development projects due to debugging and fixing residual defects [3].

"A surprisingly large amount of time and effort goes into dealing with invalid defects although this topic is seldom discussed in the quality literature…Companies that produce commercial software are aware that they not only must deal with real bugs but also must deal with enormous quantities of bugs that are not really the fault of the software against which the bug was reported" [6, 8]. A software stack may be comprised of hardware, hypervisors, operating systems, multivendor middleware and user applications. When a bug occurs the root cause may not be immediately evident and may be mistakenly identified. For example, a user or tester may open a defect report against a piece of middleware when the root cause may be in the operating system or the hardware.

Another problem is when a development team receives duplicate reports of the same defect. For example, a defect in the installation software for WordPerfect 5.0 caused over 10,000 users to call in on the same day. There were so many telephone calls that it temporarily shut down phone service into Utah! "As a rule of thumb, about 70 percent of all commercial software bugs will be found by more than one user… about 15 percent will be found by many users." [6]

In this paper an Invalid Defect report is defined as any reported defect that does not result in a unique fix for the product it was reported for. It may be a problem whose root cause lies elsewhere in the software stack, a user error including environmental misconfiguration, a defect report that is working as designed or finally a defect report that has already been found, reported and fixed.

## 2 The Cost of Invalid Defect Reports

Table 1 shows the results of a study by Capers Jones while at IBM which illustrates significant variation in defect repair hours by the severity level of the defect [7,8]. With that chart we can compute an estimate on how long we expect it to take to handle each category type of defect report as shown in Table 2. This illustrates that we can expect a severity 1 defect report to take on average 10.34 hours of effort to resolve. A severity 2 defect we expect will take 16.56 hours, severity 3 will take 6.9 hours, severity 4 will take 2.6 hours and finally an invalid defect report will take an estimated 4.98 hours to repair.

**Table 1.**

| | Severity 1 | Severity 2 | Severity 3 | Severity 4 | Invalid | Average |
|---|---|---|---|---|---|---|
| > 40 hours | 1.00% | 3.00% | 0.00% | 0.00% | 0.00% | 0.80% |
| 30 - 39 hours | 3.00% | 12.00% | 1.00% | 0.00% | 1.00% | 3.40% |
| 20 - 29 hours | 12.00% | 20.00% | 8.00% | 0.00% | 4.00% | 8.80% |
| 10 - 19 hours | 22.00% | 32.00% | 10.00% | 0.00% | 12.00% | 15.20% |
| 1 - 9 hours | 48.00% | 22.00% | 56.00% | 40.00% | 25.00% | 38.20% |
| > 1 hour | 14.00% | 11.00% | 25.00% | 60.00% | 58.00% | 33.60% |
| TOTAL | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% |

Severity 1: The software has stopped working.
Severity 2: Major features of the software have stopped working.
Severity 3: Minor defects.
Severity 4: Cosmetic defects.
Invalid: Non-defects.

**Table 2.**

| Hrs | Sev1 | | Sev2 | | Sev3 | | Sev4 | | Invalid | |
|---|---|---|---|---|---|---|---|---|---|---|
| 45 | 1.00% | 0.45 | 3.00% | 1.35 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 |
| 35 | 3.00% | 1.05 | 12.00% | 4.2 | 1.00% | 0.35 | 0.00% | 0 | 1.00% | 0.35 |
| 25 | 12.00% | 3 | 20.00% | 5 | 8.00% | 2 | 0.00% | 0 | 4.00% | 1 |
| 15 | 22.00% | 3.3 | 32.00% | 4.8 | 10.00% | 1.5 | 0.00% | 0 | 12.00% | 1.8 |
| 5 | 48.00% | 2.4 | 22.00% | 1.1 | 56.00% | 2.8 | 40.00% | 2 | 25.00% | 1.25 |
| 1 | 14.00% | 0.14 | 11.00% | 0.11 | 25.00% | 0.25 | 60.00% | 0.6 | 58.00% | 0.58 |
| | | **10.34** | | **16.56** | | **6.9** | | **2.6** | | **4.98** |

**Table 3.**

| Severity | Repair Hrs | Probability | Est. Repair Effort (Hrs) | Est. Repair Effort (%) |
|---|---|---|---|---|
| 1 | 10.34 | 8.25% | 0.85 | 13.59% |
| 2 | 16.56 | 8.25% | 1.37 | 21.76% |
| 3 | 6.9 | 8.25% | 0.57 | 9.07% |
| 4 | 2.6 | 8.25% | 0.21 | 3.42% |
| **Invalid** | **4.89** | **67.00%** | **3.28** | **52.18%** |
| Total | | 100.00% | 6.28 | 100.00% |

When the estimated effort results from Table 2 are combined with Capers Jones' claim that "software vendors can expect to receive two invalid defect reports for every valid defect report that is opened" [6] we can combine the two to create an estimated repair effort as shown in Table 3. Using these weighting criteria, it is shown that if invalid defects reports occur with 67% frequency with the remaining 33% evenly distributed between severities, we can conclude that about 52.18% of the total repair effort is consumed by Invalid Defects.

When we combine Boehm and Basili's figures with the estimates produced from Capers Jones' studies we can estimate that out of the 70% total cost of debugging and fixing residual defects 52.18% of the effort is expended on invalid defects.

This gives us a final estimate showing that up to 36.52% of the total cost of a systems development project may be consumed by efforts related to processing and debugging invalid defect reports. Capers Jones estimates that "The collected costs of processing invalid bug reports, duplicates, and abeyant bug reports can exceed 25% of the total cost of fixing real bugs" [6]. Using that figure would show 17.5% of the total cost of a software development project being consumed by invalid defect reports.

Since 17.5-36.52% of the total cost of a systems development project is consumed by invalid defect reports, it is far too large a number to ignore. Invalid defect reports must be planned for and strategies for mitigating these costs must be part of the systems development plan.

## 3 Current Mitigation Strategies

Resources do not always exist to deal with every defect report, and even mature projects can ship with known and unknown bugs [9]. Many large software projects, such as the Eclipse development project use triage systems to cope with defect reports [1]. In other development organizations there is a debug team dedicated to root cause analysis of newly opened defect records in order to prioritize them and route them to appropriate developers [10]. However with the manual approach, even though different users or system testers may report the same defect, but it may not be obvious from the users' description of the failure [11].

## 4 Predictive Analytics as a Mitigation Strategy

An alternate strategy for dealing with defect analysis and the invalid defect rate is using the predictive analytic techniques popularized by the success of IBMs Watson machine in Jeopardy. There has been research done analyzing defect reports using techniques such as

clustering in combination with text categorization techniques [13, 2]. It has been noted, that the computational expense of modeling is compounded as the volume of defect reports grow [5].

Hooimeijer and Weimer experiments have shown that even "relatively unsophisticated linear model[s] yield better-than-chance predictive power" excluding text analytics, easily achieving 62% precision when applied to the Mozilla bug repository [1]. Similar work in residual defect prediction has been done by Fenton using Bayesian Belief Networks showing that "we can gain significant improvements in predictive performance by utilizing these techniques" [4].

Experiments on test defect reports, as part of the Marist/IBM joint study have shown that application of these predictive models for classification of invalid defects can easily achieve better-than-naïve results in identifying invalid defects, as soon as they are opened and before the repair costs associated with them are incurred.

In the sample 69% of defects were invalid, while 31% were valid severity 1 through severity 4 defects. As table 4 shows, every single predictive classification algorithm was able to surpass 77% accuracy in identifying invalid defects, while the most accurate overall classifier achieved 81.43% accuracy in identifying invalid defects with minimal data cleansing and no text analytics.

**Table 4.**

| | False Positive | False Negative | Overall Error |
|---|---|---|---|
| Naïve Approach | 41% | 0% | 41% |
| Naïve Bayes (No Cleaning) | 33.76% | 12.41% | 21.03% |
| Naïve Bayes (Post Cleaning) | 25.67% | 18.15% | 21.32% |
| k-Nearest Neighbors (k=4) | 21.60% | 16.34% | 18.57% |
| Classification Tree (Best Pruned) | 24.69% | 17.25% | 20.40% |
| ANN (Minimum Overall Error) | 23.05% | 18.00% | 20.14% |
| ANN (Minimum False Positive) | 3.70% | 37.22% | 23.02% |
| ANN (Minimum False Negative) | 75% | 0.45% | 32.35% |

## 5 Conclusion

This exploration has analyzed the primary cost of invalid defects in systems development in terms of hard costs, such as defect repair hours and defect report management. The costs of invalid defects are significant, consuming 25-58% of the total defect

repair costs, or an estimated 17-36% of total system development cost. These costs are too large to ignore in information systems development quality plans.

There are many costs in addition to those explored in this paper. Some process impacts are time lost in meetings, time lost tracking and logging invalid defects and cycle time impacts of testers waiting for invalid defect root cause analysis. Furthermore in a test environment, developers observing testers who open invalid defects may cause the developers to stop taking testers seriously or otherwise cause rifts in the relationship between development and test. When users find invalid defects; even ones that are already fixed, the psychological impact is likely to result in negative user perceptions of reliability in the system.

Manual mitigation strategies have been adopted for this purpose but recent research shows analytics has the potential to automate or augment this triage process with surprising accuracy, even using relatively unsophisticated models. IBM's Watson machine was able to achieve substantial improvements partly by using many different predictive algorithms simultaneously. An approach like this would increase the predictive accuracy greatly.

**Bibliography:**

[1] Anvik, J., Hiew, L., and Murphy, G. C. Coping with an open bug repository. In Eclipse '05: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange, pages 35–39, New York, NY, USA, 2005. ACM Press.

[2] Bekkerman, R., El-Yaniv, R., Tishby, N., and Winter, Y. Distributional word clusters vs. words for text categorization. J. Mach. Learn. Res., 3:1183–1208, 2003.

[3] Boehm, B., and Basili, V. Software defect reduction. IEEE Computer Innovative Technology for Computer Professions, 34(1):135–137, January 2001.

[4] Fenton, N., Neil, M. A Critique of Software Defect Prediction Models. IEEE Transactions in Software Engineering, 25(5), pp. 675-689, 1999.

[5] Hooimeijer, P., Weimer, W. Modeling Bug Report Quality. ASE'07, November 5–9, 2007, Atlanta, Georgia, USA.

[6] Jones, Capers; Applied Software Measurement; McGraw Hill, 3rd edition, 2008; ISBN 978-0-07-150244-3.

[7] Jones, Capers; Estimating Software Costs; McGraw Hill, New York; 2007; ISBN 13-978-0-07-148300-1.

[8] Jones, Capers; A Short History of the Cost Per Defect Metric (2009); Retrieved from http://www.itmpi.org/assets/base/images/itmpi/privaterooms/capersjones/Capers_COSTPERDEFECT2009.pdf

[9] Liblit, B., Aiken, A., Zheng, X., and Jordan, M. I. Bug isolation via remote program sampling. In PLDI'03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, pages 141–154, New York, NY, USA, 2003. ACM Press.

[10] Loveland, S., Miller, G., Prewitt, R. and Shannon, M. Software Testing Techniques: Finding the Defects that Matter. Charles River Media, 2004.

[11] Podgursku, A., Leon, D., Francis, P., Masri, W., Minch, M., Sun, J., and Wang, B. Automated Support for Classifying Software Failure Reports. In Proceedings of the 25th International Conference on Software Engineering (ICSE-03), pages 465–477. IEEE Computer Society, 2003.

[12] Tanenbaum, A., and Woodhull, A. Operating Systems Design and Implementation. Prentice-Hall, Inc., 1999.

[13] Weib, C., Premraj, R., Zimmermann, T., and Zeller. A., How long will it take to fix this bug? In Proceedings of the Fourth International Workshop on Mining Software Repositories, May 2007.