

Defect patterns in two case histories: evidence in favour of continued testing after delivery

Les Hatton
CIS, University of Kingston*

September 30, 2004

Abstract

Defect patterns in two case histories with a similar client / server architecture show several interesting patterns confirmed by formal statistical analysis. First of all, users are much more sensitive to defects in graphical user interface (GUI) clients *even though the underlying server generates the important data*. This will be referred to here as *GUI blindness*. Second, about half of the defects in both products found after delivery were found by internal developers carrying on testing during normal adaptive development of the products. The end result is that in each case history, the product appears to the user to have about half of the defects actually shipped in the first release even though the products perform very different functions, (one is a language parsing static analysis tool and the other models underwater releases of compressed air used as an acoustic source in seismic surveys). This will be referred to here as *Testing after Delivery*.

\$Date: 2004/09/27 14:08:03 \$

1 Overview

This paper is empirical. It describes an unusual but systematic pattern which was observed in the author's defect database for two widely-used client/server products from very different areas: language parsing [1] and geophysical modelling, [2]. In both cases, the data is subjected to formal statistical analysis and the results are highly significant and very similar given their disparate applications area. The two products do however have the same client / server architecture, use the same programming languages and are of similar size although they have very different market places and numbers of end users. Both products are distributed by internet download only, making updates relatively simple for users. This is a key factor in exploiting the defect reduction achieved by the policy of post-release testing described below.

The analysis technique used here is to study each product independently and assess the likelihood that the percentage of defects found by the users compared with all defects found after delivery in the GUI client and in the server were the same.

*L.Hatton@kingston.ac.uk, lesh@oakcomp.co.uk

1.1 A language parsing product

Product description This particular product consists of a GUI client written in Tcl/Tk communicating with a server written in C. The product, distributed as "The Safer C Toolset", is designed to be portable so that both the client and server run on Windows, Linux and Solaris platforms. The server is around 121,000 lines of source code and the GUI client is around 25,500 lines of source code. The product was first released in July 1999 and has since gone through 13 releases up to the present day. It has several hundred users.

The defect data All defects have been tracked since product release. The product (which is a static analysis tool for the C language) goes through an unusually intensive testing regime which includes a requirement to parse the ISO C validation suite FIPS 160 successfully. As a result, the current cumulative defect density (defined here to be the total number of faults defects found since release divided by the current size in source lines of code) in both the client and the server is very low and is shown in Table 1 separated into various categories. A defect is defined here to be a fault that has failed.

Component	defect origin	Cum. defect density (per 1000 SLOC)
GUI client	Internal	0.117
GUI client	External	0.156
GUI client	Total	0.273
Server	Internal	0.115
Server	External	0.065
Server	Total	0.180

Table 1: Defect densities for case history 1

As can be seen in Table 1, one feature of the defect database is that defects which have occurred since release have always been partitioned into defects discovered inhouse (testing is ongoing and continues after product release as part of ongoing product development) and defects discovered by external users. A routine inspection of these data revealed a strong pattern whereby users appear to be much more sensitive to defects in the GUI client than in the server even though the server is effectively computing the important information. This data is subjected to a formal statistical test and its significance established in the next section.

2 Data analysis

The data will be analysed using the z-test for proportions, [3]. Let p_s be the proportion of all defects found by users in the server and let p_c be the proportion of all defects found by users in the client. Then,

$$p_s = \frac{0.065}{0.180} = 0.36; q_s = 1 - 0.36 = 0.64 \quad (1)$$

and

$$p_c = \frac{0.156}{0.273} = 0.57; q_c = 1 - 0.57 = 0.43 \quad (2)$$

It will be assumed as a null hypothesis that the two binomial populations for the defects found in the client and the defects found in the server have the same proportion of defects found by the users.

With this assumption, the following holds:-

$$z = \frac{p_s - p_c - 0}{\sqrt{\hat{p}\hat{q}\{\frac{1}{n_1} + \frac{1}{n_2}\}}} \sim N(0, 1) \quad (3)$$

where $n_1 = 35$ and $n_2 = 22$ are the number of defects found in the server and client respectively. An estimate for $\hat{p} = 0.5 * (0.18 + 0.273) = .27$ and for $\hat{q} = 0.5 * (0.64 + 0.43) = 0.504$. This gives

$$z = \frac{0.57 - 0.36}{0.1} = 2.1 \quad (4)$$

which is significant at the 2% level. We therefore reject the null hypothesis that they are the same and infer that users are more sensitive to the GUI client defects than those in the underlying server.

2.1 A geophysical modelling product

Product description Although an entirely different application area, this product has a very similar architecture and again consists of a GUI client written in Tcl/Tk communicating with a server written in C. The product, distributed as "Gundalf", is also designed to be portable across Windows, Linux and Solaris platforms. In this case, the server is around 83,900 lines of source code and the GUI client is around 24,800 lines of source code. The product was first released in April 2002 and has since gone through 23 releases up to the present day. It has twenty users at the time of writing.

The defect data All defects have been tracked since product release. The product goes through a series of regression tests comparing its output against known measurements recorded under controlled conditions usually in deep lakes or fjords. Again the current cumulative defect density in both the client and the server is very low and is shown in Table 2 separated into various categories.

Component	defect origin	Cum. defect density (per 1000 SLOC)
GUI client	Internal	0.323
GUI client	External	0.322
GUI client	Total	0.645
Server	Internal	0.108
Server	External	0.023
Server	Total	0.131

Table 2: Defect densities for case history 2

3 Data analysis

The data will be analysed in the same way as the first product using the z-test for proportions. Again, let p_s be the proportion of all defects found by users in the server and let p_c be the proportion of all defects found by users in the client. Then,

$$p_s = \frac{0.023}{0.131} = 0.176; q_s = 1 - 0.176 = 0.824 \quad (5)$$

and

$$p_c = \frac{0.323}{0.645} = 0.501; q_c = 1 - 0.501 = 0.499 \quad (6)$$

Again it is assumed as a null hypothesis that the two binomial populations for the defects found in the client and the defects found in the server have the same proportion of defects found by the users.

Repeating the analysis as used in the first product gives,

$$z = \frac{p_s - p_c - 0}{\sqrt{\hat{p}\hat{q}\left\{\frac{1}{n_1} + \frac{1}{n_2}\right\}}} \sim N(0, 1) \quad (7)$$

where $n_1 = 15$ and $n_2 = 26$ are the number of defects found in the server and client respectively. An estimate for $\hat{p} = 0.5 * (0.131 + 0.645) = .388$ and for $\hat{q} = 0.5 * (0.824 + 0.499) = 0.662$. This gives

$$z = \frac{0.501 - 0.176}{0.164} = 1.98 \quad (8)$$

which is significant at the 3% level, almost the same as the first case history, so again we reject the null hypothesis at around the same level of significance.

4 Conclusion

In two products with albeit the same architecture but very different application areas, it is shown at the 1% level, that users are more sensitive to defects in the GUI client than they are to those in the underlying server. This might seem blindingly obvious but given that in both cases, the underlying server is responsible for the essential accuracy of the information, this seems evidence of a worrying triumph of form of substance as graphical user interfaces become ever more eye-catching. This was referred to here as *GUI blindness*. This may be because the user in some sense relaxes when they find defects in the GUI and fails to be as fastidious with the actual generated information itself although this analysis has no way of knowing. *The essential conclusion is that users should pay more attention to what is important rather than what is obvious in a software product.*

A second conclusion is that from the developer's as well as the user's point of view, *it makes good sense to continue testing after delivery*. In both these case histories, about half the defects released in the product were never seen by the user because they were found first by continued testing during normal adaptive product development and corrected before the users ever had an opportunity to be inconvenienced by them. This however relies on the ability to update a product quickly and easily as is possible with internet download.

References

- [1] The Safer C toolset(1999) <http://www.oakcomp.co.uk/>
- [2] Gundalf: a modelling system for arrays of airguns in marine seismic surveying (2002) <http://www.gundalf.com/>
- [3] Spiegel M.R. and Stephens L.J. (1999) *Statistics, 3rd edition, (Schaum series)*, McGraw-Hill, New York.