

Fault Causing Characteristics in Software Development

Santos Faundez Sekirkin

Dept of Statistics
National University of Ireland - Dublin
Belfield - Dublin 4
Ireland
s00bf2e2@student.ucd.ie

Philip J. Boland

Dept of Statistics
National University of Ireland - Dublin
Belfield - Dublin 4
Ireland
philip.j.boland@ucd.ie

Abstract

The development of a piece of software can be a lengthy and complex process. The ultimate goal is to produce high quality software, and in so doing the company must establish a careful and detailed strategy for the detection and removal of irregularities which may be introduced into the software at any stage of development. In order to achieve this goal, most companies use a quality control system which involves the compilation of an extensive database of software problem reports. An SPR (software problem report) results from the detection of a fault by testers - for example by the running of test cases (scripts) covering the software specifications. A typical SPR contains information on the properties of the defects underlying the reported problem. These may include : the present state of problem, time effort necessary to treat the problem, severity, priority, functional area, individuals involved in the development and quality assurance process, and other indicators of complexity of that part of the code where the problem originated. This research involves a detailed analysis of the evolution of the development process corresponding to a collection of similar software projects in a local software company. One goal is to discover (by the use of multivariate statistical techniques such as factor analysis and/or classification analysis) common patterns in the problem reports in order to group them into clusters described by similar causes. Other objectives include analyzing times to both find and fix faults, which in turn might suggest improvements in the process of software development.

1 Introduction

Software Engineering is a collection of techniques and tools used to produce **reliable** software under conditions of **restricted resources**. Resource constraints generally fall into the areas of the time and monetary means assigned to the development of a project. Any developer of commercial software is concerned about the most accurate possible estimation of schedules for development and release of it's product, but also for it's reliability and overall quality. The idea for this research arose from a study of the necessary tools which exist for the assessment of the reliability of mission critical systems software. According to it's nature, tools and techniques used in the development of such critically important software in order to achieve high reliability are far too expensive to implement in the case of most commercial software. For instance, such techniques often entail very careful registration of the times between failures as well as careful records on the time to rectify any faults found, which according to our experience are rarely found in the developers of most commercial software and professional testing companies.

Our research is based on a case study of projects provided by local commercial software development and testing companies, but we believe our results will be readily applicable to many other software development projects with similar characteristics. The object of the research is to use the data collected by company quality engineering groups involved in project development to model the arrival of software problem reports via the use of stochastic modelling of marked point processes. One aim is to predict the number of reports on the basis of previously developed similar software projects. Furthermore, we expect to make use of the time to resolution of each of the software reports to model and predict the time to resolution of future reports.

A distinctive aspect of the project is that it takes into account the severity of each fault which is recorded in a software problem report, and it's relationship to both the time to resolution of the problem as well as the functional area of the software. The use of exploratory techniques on the data, such as correspondence analysis and principal components analysis, are done with the objective of singling out the functional areas of the software responsible for generation of reports of certain severity categories. Ideally one would like to extract features of the components underlying each functional area in order determine their relationship to the various severity categories, however this is often not possible due to lack of direct access to the code involved. These exploratory techniques may allow one to identify groupings of homogeneous functional areas which may be tracked by counting processes and lead to predictions for future numbers of problem reports.

2 Data used for the research

The data on which we base our modelling comes from locally developed software of a commercial nature, the details of which must be kept confidential because of their current relevance in ongoing software projects which are extensively used. The following table gives a list of some of the more important variables which are commonly entered into the SPR database of a software quality engineering group tracking development of a software project.

Table 1: Variables in a Software Problem Report Database

Data	Description
Functional Area	<i>Particular part of the software responsible for execution of certain tasks. Each functional area is considered as a block of software that can be developed independently.</i>
Severity of SPR	<i>Characterizes the level of impact of the problem on the normal performance of the software as a whole, as well as it's impact on the quality of the product and image of the developer.</i>
Date Created	<i>"Date Created" is the date when the software problem has been first recorded as a result of some irregularity observed in the performance of the software by internal or external users, that start using the software after a certain stage of integration.</i>
Date Open	<i>This is usually the date when the problem has been validated (as a real problem) and passed onto the development team for action. Being able to reproduce the fault is a key aspect in validating the fault.</i>
Pending Date	<i>The date when the problem, having been submitted to the developer, is fixed by her/him or is classified as a problem beyond her/his ability to fix. In the latter case the problem may be deferred or opened again.</i>
Date Verified	<i>After the fix of the bug has been made the software is returned to quality engineering group for verification of a correct fix for the problem.</i>
Date of Regression Pass	<i>After verification of a fix, a global check is carried out by the quality group through subjecting the software to a regression test, that is running test cases to verify that the original problem encountered has been correctly addressed.</i>
Date Closed	<i>Date when the problem is considered resolved, that is the bug is fixed, or the original report is confirmed to be a false alarm</i>

- Alt, M. (1990). *Exploring Hyperspace: a non-mathematical explanation of multivariate analysis*, McGraw - Hill Book Company.
- Miller, E. and Howden, W.E. (1981). *Software Testing & Validation Techniques*, Computer Society Press.
- Shumway, R. H. (1988). *Applied Statistical Time Series Analysis*, Prentice Hall.
- Box, G. E. P. and Jenkins, G. M. (1976). *Time Series Analysis: Forecasting and Control*, Holden-Day.
- Dillon, W. R. and Goldstein, M. (1984). *Multivariate Analysis: Methods and Applications*, John Wiley & Sons, Inc.
- Griliches, Z. and Intriligator, M. D. (1984). *Handbook of Econometrics*, Vol. 2, North-Holland.
- Musa, J. (1988). *Software Reliability Engineering*, McGraw-Hill.
- Harvey, A. C. (1981). *The Econometric Analysis of Time Series*, Philip Allan Publishers, Ltd.
- Mills, T. C. (1990). *Time Series Techniques for Economists*, Cambridge University Press.