Software Defect Analysis of a Multi-release Telecommunications System

Marek Leszak

Lucent Technologies Bell Labs
Lucent Technologies Network Systems GmbH, Optical Networking Group,
Thurn-und-Taxis-Str. 10, 90411 Nuernberg, Germany
Phone: +49 (0) 911 5263382
mleszak@lucent.com

Abstract. This paper provides a study of several process metrics of an industrial large-scale embedded software system, the Lucent product LambdaUnite™ MSS. This product is an evolutionary hardware/software system for the metropolitan and wide-area transmission and switching market. An analysis of defect data is performed, including and comparing all major (i.e. feature) releases till end of 2004. Several defect metrics on file-level are defined and analyzed, as basis for a defect prediction model. Main analysis results include the following. Faults and code size per file show only a weak correlation. Portion of faulty files per release tend to decrease across releases. Size and error-proneness in previous release alone is not a good predictor of a file's faults per release. Customer-found defects are strongly correlated with pre-delivery defects found per subsystem. These results are being compared to a recent similar study of fault distributions; the differences are significant.

Keywords: Case study, software process metrics, defect prediction, error-proneness, defect density.

1 Introduction

Measurement and evaluation of product and process characteristics is a critical activity throughout the entire software development, evolution, and maintenance lifecycle. It is fundamental to determine whether the software products we develop have the desired functional and non-functional properties; it is fundamental to determine whether we have achieved the desired quality, cost, and schedule attributes in our development projects.

The subject of our empirical study is LambdaUnite™ MSS [1] -a multi-release, large embedded hardware/software system which is still in evolution. For further details of this product see the related previous study [2]. Lifespan of this product reaches from its first commercial release in 2001, and subsequent feature release approx. every 6 month. Up to now (end of 2004) 11 releases have been or are being

F. Bomarius and S. Komi-Sirviö (Eds.): PROFES 2005, LNCS 3547, pp. 98–114, 2005.

delivered to various telecom service providers worldwide. As basis for our case study, we take the change evolution over these 11 releases.

Our development approach can be shortly characterized as follows. R&D processes applied for the LambdaUnite product are defined, applied in all releases, and quality con trolled. The product lifecycle is divided into several phases, reaching from system requirements & architecture, (software and hardware) design & development, software (and hardware) integration, system test, general availability (GA), to maintenance. The software part consists of software source and build files, and of architecture and design documentation - the latter being out-of-scope for this study. Roughly 85% of the sources are written in C++ and C; a smaller portion using shell script, tcl/tk, and perl. Source files mostly represent C++ classes and associated header files. Some principles from which we could profit for our study, to obtain consistent and complete raw data:

- all changes to software files are guarded by identified and managed Modification Requests (MRs) in the central change management database, adapted from ClearDDTS™ (from IBM-Rational)
- all software files are version controlled in ClearCase™ (from IBM-Rational)
- links between MRs and touched files are stored automatically during check-out and check-in
- various descriptive attributes per file and per MR are kept and integrity-controlled by a dedicated change management team, the change control board
- all this data is consistently available, in one environment, for all product releases

Goals of This Study

The underlying objective for the investigation in this paper is to obtain insight into software changes, esp. into error-proneness, to get quantitative decision input for better focusing quality improvement activities. Decision making input is provided mainly for

- prioritization of software and system testing, and assigning resources to corrective maintenance
- hints for re-design and refactoring of error-prone software parts

To support these objectives, our investigation is driven by following goals:

G1. Predict number of defect MRs (observed defects), overall and distribution per software subsystem

G2. Predict error-proneness, by studying defect distributions both on file level and on subsystem level

In order to achieve stronger focus on the defect model and analysis of concrete measurements, several correlations between defects and other metrics are studied.

G3. Predict number of post-release defects per product release, by studying correlation with e.g. pre-release defects