Software Faults in Evolving a Large, Real-Time System: a Case Study

Dewayne E. Perry and Carol S. Stieg

Abstract. We report the results of a survey about the software faults encountered during the testing phases in evolving a large real-time system. The survey was done in two parts: the first part surveyed all the faults that were reported and characterized them in terms of general categories; the second part resurveyed in depth the faults found in the design and coding phases. For the first part, we describe the questionnaire, report the general faults found, and characterize the requirements, design and coding faults by the testing phases in which they were found and by the time they were found during the testing interval. For the second part, we describe the questionnaire used to survey the design and coding faults, report the faults that occurred, how difficult they were to find and fix, what their underlying causes were (that is, what their corresponding errors were), and what means might have prevented them from occurring. We then characterize the results in terms of interface and implementation faults.

1 Introduction

It is surprising that so few software fault studies have appeared in the software engineering literature, especially since monitoring our mistakes is one of the fundamental means by which we improve our process and product. This is particularly true for the development of large systems. In preceding work [13, 14], Perry and Evangelist reported the prevalence of interface faults as a major factor in the development and evolution of a large real-time system (68% of the faults). One of the main purposes of that software fault study was to indicate the importance of tools (such as the Inscape Environment [16]) that manage interfaces and the dependencies on those interfaces.

Prior to this work, Endres [7], Schneidewind and Hoffman [19], and Glass [8] reported on various fault analyses of software development, but did not delineate interface faults as a specific category. Thayer, Lipow and Nelson [21] and Bowen [5] provide extensive categorization of faults, but with a relatively narrow view of interface faults. Basili and Perricone [2] offer the most comprehensive study of problems encountered in the development phase of a medium-scale system, reporting data on the fault, the number of components affected, the type of the fault, and the effort required to correct the fault. Interface faults were the largest class of faults (39% of the faults).
We make two important contributions in this case study. First, we present software fault data on evolutionary development [16], not on initial development as previous studies have done. Second, we use a novel approach in which we emphasize the cost of the finding (i.e., reproducing) and fixing faults and the means of preventing them.

In section 2, we provide the background for the study, describing the system in general terms, and the methodology employed in evolving the system. In section 3, we describe our experimental strategy and the approach we used in conducting the survey. In section 4, we report the overall Modification Request (MR) survey, providing first a summary of the questionnaire, then a summary of the results, and finally some conclusions. In section 5, we present the design and coding fault survey, providing first a summary of the questionnaire, then a discussion of the analysis, and finally a summary relating the results to interface faults. In section 6, we present conclusions and recommendations.

2 Background

The system discussed in this paper is a very large 3 scale, distributed, real-time system written in the C programming language in a Unix-based, multiple machine, multiple location environment.

The organizational structure is typical with respect to AT&T projects for systems of this size and for the number of people in each organization. Not surprisingly, different organizations are responsible for various parts of the system development: requirements specification; architecture, design, coding and capability testing; system and system stability testing; and alpha testing.

The process of development is also typical with respect to AT&T projects of this size. Systems Engineers prepare informal and structured documents defining the requirements for the changes to be made to the system. Designers prepare informal design documents that are subjected to formal reviews by three to fifteen peers depending on the size of the unit under consideration. The design is then broken into design units for low level design and coding. The products of this last phase are subjected both to formal code reviews by three to five reviewers and to low level unit testing. As components become available, integration and system testing is performed until the system is completely integrated.

The release considered here is a "non-initial" release — one that can be viewed as an arbitrary point in the evolution of this class of systems. Because of the size of the system, the system evolution process consists of multiple, concurrent releases — that is, while the release dates are sequential, a number of releases proceed concurrently in differing phases. This concurrency accentuates the inter-release dependencies and their associated problems. The magnitude of the changes (approximately 15-20% new code for each release) and the general make-up of the changes (bug-fixes, improvements, and new functionality, etc.) are generally uniform across releases. It is because of these two facts that we

3 By "very large", we mean a system of 1,000,000 NCSL or more [4]. AT&T has a wide variety of such very large systems.