Software Failures: Fishing in an Ocean not a Pond

Nachshon Margaliot

Dov Téeni
Bar-Ilan University

Follow this and additional works at: http://aisel.aisnet.org/icis1996

Recommended Citation
http://aisel.aisnet.org/icis1996/35

This material is brought to you by the International Conference on Information Systems (ICIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in ICIS 1996 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
SOFTWARE FAILURES: FISHING IN AN OCEAN NOT A POND

Nachshon Margaliot  
Dov Te’eni  
Bar-Ilan University

There is an urgent need for research in the economics of software development and maintenance. Decisions such as the timing of software release and the allocation of resources to testing are critical economically and strategically, yet very few organizations use analytical models of software development to support such decisions. Our objective is to develop a practical model of software failures suitable for today’s open systems environment.

A failure is an unexpected result of a program execution. Failures are the external phenomena that the user experiences. An error (or fault) is an incorrect code that, under certain conditions, will produce a failure. The more failures experienced, the more errors assumed to exist in the software.

Finding errors is like fishing. The proposed model describes fishing in an ocean rather than the pond of previous models. In a pond, the rate of catching fish depends on how many are left in it. When the pond is opened to an ocean, waves bring in new fish. Fishermen can usually find the whales but have no hope of finding all the small fish. New fishermen may have a better chance of finding new fish because they bring new perspectives but they too are at a loss when confronted with a practically infinite number of fish from the ocean.

Open computer systems are analogues to a pond linked to the ocean. The potential number of failures due to communication software, printers, operating systems and I/O devices is practically infinite. In the new context of open systems, we distinguish between failures due to errors in the specified system and failures due to incompatibilities. The former are analogues to fish in a pond. The latter, analogues to fish from the ocean, are caused by code that is incompatible with new conditions, e.g., working in parallel with other packages. These are usually small fish, not whales.

In contrast to most traditional models of software errors, we

1. use different dependent variables before and after software release;
2. distinguish between the detection of failures due to errors of the specified environment versus new environments;
3. assume infinite incompatibilities in infinite time due to ever-changing environments;
4. assume a delay between detections and corrections, which are distributed in patches or new versions.

We began with a pilot study of post-release behavior in a large insurance company. We then started systematic work with PCSOft International in April 1995. The authors and two graduate students used semi-structured interviews to characterize the error detection processes. Figure 1 plots both types of failures (due to errors in the specified system and incompatibilities) for the pre-release stage.

Acknowledgment: We thank Laura Darcy for introducing us to the software errors and John Wilson for his insightful commentary.
Figure 1: Failures during testing at PCSoft Int.

The data is used to estimate the rate of failures detected, which is then used to optimize the timing of patch release and new versions. When the rate of failures is constant, we can use an inventory model to time the release of patches or versions according to the ratio between costs of administration and costs of error. In Figure 2, N is the rate of error detection by users estimated from the data using our model.

Figure 2. An Inventory Model of Errors for Timing Releases