

What Is Software Failure ?

Ram Chillarege, Member IEEE

IBM Watson Research, Hawthorne

As software dominates most discussions in the information technology business, one needs to examine carefully where we are headed in software reliability. It is reasonable to ask about the nature of software faults and the remedy for them, either from a fault-tolerance perspective or, more generally, on a software dependability front. There are conferences on this topic, and over 300 technical papers that discuss some of its aspects. However, as many industry specialists agree, software is one area in the information technology industry which continues to baffle the scientist from a dependability perspective. While hardware & technology have seen four orders of magnitude improvement in the past decade, software has probably marginally improved or, some will argue, gotten worse. One then wonders if research in this area is headed in the right directions?

Thus it is vital to examine some of the fundamentals, eg, "What is a software failure?" This question is usually assumed to be well understood. However, when one examines it closely, it is startling how little is truly understood regarding those faults that matter. This problem is appreciably better understood in hardware than in software. Failures in hardware components, subsystems, or systems, have been better tracked over the years and there is a larger body of experience in failure modes & effects analysis. However, the counterpart in software is far less understood. It is further complicated by a lack of clarity as to what is a software failure.

The concept of a failure is defined [1] as "deviation of the delivered service from compliance with the specification". This definition applies well to hardware where there tends to be a reasonable level of specifications either on the product or the system. However, in the world of software, it causes major confusion since there aren't well-defined specifications for most products. Having an unclear definition for failure makes the task of designing fault-tolerance even harder. Therefore, it is necessary to examine carefully what might be an apt definition of software-failure.

A good place to start is the customer service center. Apart from obvious failures when a product ceases to work, there are a myriad of problem conditions where the distinction between failure and correct operation is not clear. If there were well-defined specifications, it could be argued whether the product worked as designed or not, distinguishing failure from correct operation.

However, as the software business explodes and reaches new customer groups, the tradition of a specification has changed. Often products do not have detailed specifications, nor would the customer expect them. Determining a failure by strict definition of the word loses relevance. What is important to the customer is that the product does the work and that they are satisfied. Customer expectation largely determines whether a failure has occurred or not.

It is far more useful, in the modern software business, to define a failure as when:

"The customer's expectation has not been met and/or the customer is unable to do useful work with the product."

This change in the definition of what is considered a failure has major repercussions throughout the software industry. It changes the focus of how products are planned and designed. For the software reliability community, this would be a major change that needs to be woven into the fabric of research, tools, and the development process.

Let us examine some of the immediate implications. Firstly, there is no fixed fault model. Since we drive the definition of a failure primarily by customers' expectation and their ability to work, it will be a moving target. As technology evolves, product concepts evolve and knowledge in the customer base grows; and the types of faults to be tolerated will change. To plan the fault-tolerance one has to be aware of the window of opportunity and aggressively conduct the studies to gain the necessary insights. The new definition also puts a variety of disparate problems on the same platter. Since the focus is on customer expectations, *failure* does not distinguish sources of faults as much as the effect. Some would argue that this has always been the case. However, with this new definition of failure there is no concept of a specification to hide under.

The data captured in the customer call centers has traditionally been split into defect oriented problems and non-defect oriented problems. The word defect refers to the fact that there is a change in software, which is necessitated due to a programming fault. A non-defect has been associated with issues that do not require a programming change; it includes difficulties with install, use, configurations, inter-operability, etc.

The data tell us that there is a 90% - 10% split between the non-defect oriented problems and defect oriented problems. The traditional view was that 90% of the problems are not associated with what would be classified as a software failure. However, since we have redefined software failures as events where customer

COMMENTARY COMMENTARY COMMENTARY COMMENTARY COMMENTARY COMMENTARY COMMENTARY

expectation is not met and/or that a customer is unable to do its work, the set of problems that need to be also considered for fault-tolerance is changed. Beneath the set of failures is a set of faults. When we have a new set of failures to address, fault-tolerance has to re-examine the methods, techniques, and issues that we focus on. This is a major shift in the perception of what software dependability needs to provide and what directions research needs to take.

These new perspectives on what a failure is, and therefore, the measures related to them such as reliability, need to be re-examined. This will be the beginning of setting the agenda towards the problems that really need to be tackled. Else software reliability may wander into the next decade no different from the current one.

REFERENCE

- [1] J.C. Laprie, *et al*, *Dependability: Basic Concepts and Terminology, Dependable Computing and Fault-Tolerant Systems*, vol 5, 1992; Springer-Verlag.

AUTHOR

Dr. Ram Chillarege; IBM T. J. Watson Research Center; Hawthorne, New York 10532 USA.

Internet (e-mail): ranchill@watson.ibm.com

Ram Chillarege is Senior Manager in the Center for Software Engineering at the IBM T. J. Watson Research Center. He is also an Associate Editor of this *Transactions*.

Publisher Item Identifier S 0018-9529(96)-08675-7

◀TR▶

OPINION OPINION OPINION OPINION OPINION OPINION OPINION OPINION OPINION OPINION OPINION OPINION

Knowledge vs Understanding

Paul Gottfried, Senior Member IEEE
Consultant, Silver Spring

In my consulting practice, I have found that one of the worst mistakes that I have made repeatedly is to accept the client's definition of the problem. If I let the client define the questions, I may give him technically correct answers — but at least as often as not, they are the right answers to the wrong questions. I suspect that the client typically is so close to the problem that he has (perhaps unconsciously) predetermined some aspects of the "solution", and his questions implicitly assume that these unstated pieces of the "solutions" are valid. It is vitally important for the consultant to question the questions — to make sure that he understands the real problem.

Textbooks, this *Transactions* and other journals and, of course, commercial software packages contain — among other things — correct answers to many questions. The set of questions and corresponding answers

constitutes knowledge. To choose the right questions requires understanding; the literature, and especially software packages, must not be expected to provide this. The practitioner must develop the necessary understanding: if he fails to do so, he is at best practicing what we used to call "cookbook engineering".

AUTHOR

Paul Gottfried; 9251 Three Oaks Drive; Silver Spring, Maryland 20901 USA.

Paul Gottfried (M'51, SM'58) received the BSEE (1949) from Rose-Hulman Institute of Technology. He has been an independent consultant in reliability and system safety since 1971. Mr. Gottfried is a Fellow of AAAS and a member of ASA and INFORMS, and has been active in the IEEE Reliability Society and its precursors for 40 years. He is a registered professional engineer.

Correspondence received 1996 July 20.

Publisher Item Identifier S 0018-9529(96)07558-6

◀TR▶