A Classification of Software Faults

Michael Grottke* and Kishor S. Trivedi
Department of Electrical & Computer Engineering
Duke University, Durham, NC 27708-0291, USA
{grottke, kst}@ee.duke.edu

1. Introduction

In recent years, researchers and practitioners have reported that software systems running continuously for a long time tend to show a degraded performance, an increased occurrence rate of failures (i.e., deviations of the delivered service from the correct service) or both. This phenomenon is referred to as "software aging". There have been several attempts to relate software faults (or bugs) responsible for aging to other classes of faults, like Bohrbugs and Heisenbugs. However, the meaning attached to the latter term is not consistent throughout literature, and often the terms are used without any explicit definitions at all. In this Fast Abstract, we propose definitions for a number of fault classes and clarify their relationships.

2. Proposed fault classification

2.1. Bohrbugs

The term "Bohrbug", first used in print by Gray [2], relates to solid or hard [1] software faults, i.e., faults that are easily detected and fixed and for which the failure occurrences are easily reproduced. This meaning is undisputed in the literature, see for example [3]. In our opinion, the terms "Bohrbug" and "Mandelbug" (see below) are complementary antonyms; i.e., each software fault belongs to exactly one of the two classes. We therefore propose the following definition:

Bohrbug := A fault that is easily isolated and that manifests consistently under a well-defined set of conditions, because its activation and error propagation lack "complexity" as set out in the definition of Mandelbug. Complementary antonym of Mandelbug.

2.2. Mandelbugs

The usual definition of a Mandelbug is "a bug whose underlying causes are so complex and obscure as to make its behavior appear chaotic and even non-deterministic" [3]. If the "behavior" of the bug is supposed to refer to the question whether it causes a failure or not, then "chaotic and even non-deterministic behavior" means that under seemingly identical conditions sometimes a failure occurs, while on other occasions no failure is experienced. This is another way of expressing the fact that a failure is not systematically reproducible, i.e., that it is caused by a soft or elusive [1] software fault. According to our interpretation, each software fault is either a Bohrbug or a Mandelbug. However, if the classification of a specific fault as Mandelbug is to be based on the judgement that the circumstances under which it leads to a failure are "too complex" to be easily reproducible, then the parting line between the two classes is essentially subjective.

In order to make the classification more objective, there is the need for an explanation of what constitutes the complexity that makes a software fault a Mandelbug. We identify two possible cases that are not mutually exclusive:

Firstly, a software fault in a specific application is a Mandelbug if the fact whether it causes a failure is influenced by other elements of the software system apart from the application itself, e.g., the operating system or the hardware. We refer to the set of these elements as the "system-internal environment" of the application. The influence of the system-internal environment can occur at any stage of the chain of causation between the fault and the eventual failure occurrence; i.e., a fault is a Mandelbug if its activation and/or its error propagation depend on interactions with the system-internal environment of the application. (The idea for this classification criterion is due to Shetti [4].) Examples are faults causing failures due to side-effects of other applications and faults for which the scheduling done by the operating system is crucial for the occurrence of a failure.

Secondly, we classify a fault in an application as a Mandelbug if complexity of the error propagation results in a delay between the fault activation and the final failure occurrence. For example, an erroneous calculation due to a fault in the software code of an application may be kept in the memory without immediately causing the service delivered by the software system to deviate from correct service; only later, when the result of the calculation is accessed and used in a way that influences the system behavior perceivable by the user, a failure will be experienced. In our opinion, it is not possible to specify exactly how long this delay has to be in order to make the failure occurrence appear non-deterministic. Therefore, this second classification criterion necessarily remains somewhat ambiguous.

Our proposed definition is as follows:

*Corresponding author, on leave of absence from the Chair of Statistics and Econometrics, University of Erlangen-Nuremberg, Germany. This work was supported by a fellowship within the Postdoc Program of the German Academic Exchange Service (DAAD).

In *Supplemental Proc. Sixteenth International IEEE Symposium on Software Reliability Engineering*, pages 4.19–4.20, 2005.

Mandelbug := A fault whose activation and/or error propagation are complex, where "complexity" can take two forms:

1. The activation and/or error propagation depend on interactions between conditions occurring inside the application and conditions that accrue within the system-internal environment of the application.
2. There is a time lag between fault activation and failure occurrence, e.g., because several different error states have to be traversed in the error propagation.

Typically, a Mandelbug is difficult to isolate, and/or the failures caused by a it are not systematically reproducible. Complementary antonym of Bohrbug.

2.3. Heisenbugs

Gray, who wrote the earliest paper [2] mentioning "Heisenbugs", uses the term as a synonym for elusive faults. Drawing upon that paper, most researchers in the field of software rejuvenation have adopted this interpretation.

However, the term had been invented by Lindsay while working with Gray at the University of Berkeley in the 1960s [6]. According to Lindsay, he came up with the word for referring to a software fault that "went away, because the measurement or the observation affected the phenomena you were trying to see" [6]. Outside the field of software rejuvenation, most references mentioning the word "Heisenbug" use it in this sense, sometimes including those software faults whose failure behavior alters (although the failure does not completely disappear) when it is researched; see [3]. This is the interpretation that we will follow.

An interesting collection of field reports about Heisenbugs [5] reveals two important categories of how trying to observe a failure can make it disappear:

1. Some debuggers initialize unused memory to default values. Failures related to improper initialization may therefore go away as soon as the debugger is turned on.
2. Trying to investigate a failure can influence process scheduling in such a way that the failure does not occur again. For example, scheduling-related failures in multi-threaded programs may disappear when a debugger is used to single-step through a process.

In both cases, the act of observing influences the failure behavior via factors belonging to the system-internal environment of the application in which the Heisenbug is located. Therefore, all Heisenbugs are Mandelbugs. Whether a Mandelbug will stop manifesting (or manifest differently) depends on the method or tool employed for probing or isolating it. As a consequence, a fault can only be classified as a Heisenbug with respect to a *specific* observation method/tool. We suggest the following definition:

Heisenbug := A fault that stops causing a failure or that manifests differently when one attempts to probe or isolate it. Sub-type of Mandelbug.

2.4. Aging-related bugs

The software aging phenomenon can be explained by the fact that the responsible faults (referred to as "aging-related bugs") cause errors to accumulate over time. These error conditions may accrue either within the running application (e.g., round-off errors in program variables) or in the system-internal environment (e.g., unreleased physical memory due to memory leaks in the application). In either case, the error conditions do not lead to failures right away - otherwise, there could be no aging -, but the failures occur with a delay. This fulfills the second criterion for classifying the underlying faults as Mandelbugs; therefore, all aging-related bugs are Mandelbugs. The class of aging-related bugs of an application may or may not overlap with the class of those software faults that are Heisenbugs with respect to a specific observation tool or method.

Our proposed definition is as follows:

Aging-related bug := A fault that leads to the accumulation of errors either inside the running application or in its system-internal environment, resulting in an increased failure rate and/or degraded performance. Sub-type of Mandelbug.

A Venn diagram showing the relationships between the four fault categories is depicted in Figure 1.

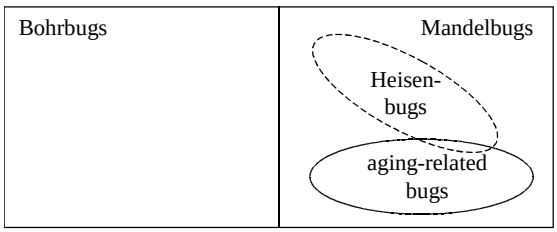

Figure 1. *Venn diagram of software fault types*

References

[1] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[2] J. Gray. Why do computers stop and what can be done about it? Technical Report 85.7, PN87614, Tandem Computers, Cupertino, 1985.

[3] E. S. Raymond. *The New Hacker's Dictionary*. The MIT Press, Cambridge, 1991.

[4] N. Shetti. Heisenbugs and Bohrbugs: Why are they different? Technical Report DCS-TR-580, Department of Computer Science, Rutgers University, Piscataway, 2005.

[5] Wiki Wiki Web. Heisen bug examples. Last modified Jan. 21, 2004, URL = http://c2.com/cgi/wiki?HeisenBugExamples (Link verified on May 26, 2005).

[6] M. Winslett. Bruce Lindsay speaks out. *ACM SIGMOD Record*, 34(2):71–79, 2005.

In *Supplemental Proc. Sixteenth International IEEE Symposium on Software Reliability Engineering*, pages 4.19–4.20, 2005.