Use and Abuse of Exceptions -
12 Guidelines for Proper Exception Handling

Jürgen Schwille

University of Stuttgart, Dept. of Computer Science, FRG
schwille@informatik.uni-stuttgart.de

Abstract. This paper presents a list of twelve guidelines for proper use of exception handling in Ada. After comparing Ada’s exception handling to other mechanisms, each guideline is discussed in depth, illustrated by several examples. Analyzing the Booch Components provides additional substantiation for the presented guidelines. After reading this paper you should be able to distinguish clearly when to use and when not to use exception handling - an important question that every Ada developer should have in mind.

1 Introduction

Ada combines well-known techniques, like modularization or strong typing, with novel features, like generics or exception handling, in a coherent fashion. Many of these, and in particular exceptions, have raised some controversy.

Exceptions are an essential Ada feature but we need guidelines to use them in a proper way. Hoare (1981) is correct in stating that exceptions are a "dangerous" feature since they can easily be misused. Rule out this danger by adhering to the following guidelines:

Establishing Exceptions
1. Every exception must represent an error.
2. Set up a user-defined exception for every predefined exception a program unit possibly propagates.

Describing Exceptions
3. Describe exactly the pre- and postconditions of every user-defined exception.

Raising Exceptions
4. Clean up before you leave.
5. Raise an exception as early as possible.
6. Provide a check function if possible.
7. Never let an exception propagate beyond its scope.
8. Do not raise predefined exceptions explicitly.

Handling Exceptions
9. Handle an exception either by removing the cause or by propagating it.
10. Record the origin and the cause of an exception.
11. Use caution when programming handlers for others.
12. Handle exceptions locally.
2 Error Handling Mechanisms

Error handling is the daily grind of every programmer. It is inevitable to handle erroneous user input, for example. Meyer (1988) states that "special cases are the scourge of programmers." How can we master them?

The following comparison points out the main differences of Ada's exception handling to other mechanisms usable in Ada: gotos, status codes, checks, and error handling routines.

**Gotos** give the developer (too) much freedom in designing the control-flow of a program. In contrast, Ada is more restrictive: normal program execution is abandoned when an exception occurs, exceptions can only be propagated along the invocation chain, etc. Ada forces the developer to produce understandable, instead of spaghetti code.

**Status codes** are used to indicate erroneous program execution. Every subprogram returns a status code that has to be checked after each subprogram call. For every possible status code an appropriate handling has to be provided. Ada's exception handling mechanism, as compared to status codes, is more reliable and leads to programs that are easier to understand: First, a forgotten status code check can have disastrous consequences as the program continues to execute, while a forgotten exception handling leads inevitably to an abandoned program. This is probably better than continuing after an unhandled error occurred. Second, when using status codes, the error handling code is mixed with code for normal processing. In Ada these two types of code can be separated clearly.

**Checks** facilitate error prediction. A check function tests whether the following subprogram call will produce an error, by examining the appropriate error condition. Such a check function *cannot guarantee* an error-free execution, it only indicates that the execution will *probably* be error-free. Also, it is sometimes either impossible or very costly to provide such a check function. In addition, the above-mentioned problem of mixing error handling code with code for normal processing applies here as well.

**Error handling routines** were suggested by Parnas (1972), among others. The specification of a subprogram describes the triggering conditions for an error, as well as the name of the appropriate error handling routine. Parnas assumes that the error handling routine is written by the user of the subprogram.

Suppose you want to use error handling routines in Ada. To facilitate more than one error handling routine, and for ease of clarity, the error handling routine should be an explicit parameter of the called subprogram. In C++ you can use function pointers for this purpose (Stroustrup, 1986). In Ada, we need to use generics instead. Thus, we have to deal with many generic subprograms - this is not easy, especially when you think of using more than one error handling routine in the same caller. (In Ada 9X, though, subprograms will be allowed as parameters.)

Finally, error handling routines do not solve the problem of mixing error handling code with code for normal processing.

---

1 The term error handling routine, actually an error handling subprogram, is used only to avoid confusion with the subprogram that detects an error.