Applying Evolutionary Testing to Search for Critical Defects

André Baresel, Harmen Sthamer, and Joachim Wegener

Software Methods and Tools, Research and Technolgy, Daimlerchrysler AG,
Alt-Moabit 96a , 10559 Berlin, Germany
{andre.baresel,harmen.sthamer,
joachim.wegener}@daimlerchrysler.com

Abstract. Software systems are used regularly in safety-relevant applications. Therefore, the occurrence of critical defects may not only cause costly recalls but may also endanger human lives. Accordingly, the development of software systems in industrial practice must comply with the highest quality requirements and standards. In practice, the most important analytical quality assurance method is dynamic testing and the most important activity to ensure this quality is test case determination. The effectiveness and efficiency of the test process can be clearly improved by Evolutionary Testing. Evolutionary Testing is a metaheuristic search technique for the generation of test cases.

1 Introduction

Testing is the most important quality assurance method for embedded systems. To increase the effectiveness and efficiency of the test and thus reduce the overall development costs, DaimlerChrysler Research works in the area of *Evolutionary Testing* [2]. Critical defects are determined using the classical Evolutionary Structural testing approach [1]. The search for defects has to generate inputs executing the statement under investigation as well as an additional condition which will lead to a defect situation. For this reason, we designed a new set of instrumentation functions that monitors during test executions actual values, array index values, pointer addresses, etc... We defined test goals for all possible locations of defects, e.g. arithmetic operations and conditions which could lead to erroneous program behavior. These test goals were investigated separately by evolutionary optimizations. To check the feasibility of this approach the software of a navigation system is used.

2 Critical Defects

Critical defects emerge from statements or conditions which are executed and result in a defect situation and thereby violate the system's integrity. Critical defects can cause data corruption, hang-ups, system crash and can thus even endanger human life. Often, the problem is difficult to be identified because only specific input scenarios lead to one of the defects known for software systems. In this project the following five defects were under investigation:

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1427–1428, 2004.
© Springer-Verlag Berlin Heidelberg 2004

Erroneous Memory Accesses can be caused by erroneous array indices (out of bounds errors) or incorrect pointer usage (null pointer usage). Three defects are defined here in the category **Arithmetric Calculation Errors.** These are the well-known problems of overflow, underflow and division by zero. Depending on the processor, the programming language and the runtime system, underflow may be ignored and zero substituted for the unrepresentable value, though this might lead to a later division by zero error which cannot be ignored. The third critical defect is the **Violation of Assertions**. Assertions are software codes which check and monitor whether certain conditions within the software code are fulfilled. A major problem in software development are incorrect program constructs which lead to so-called **Endless Loops**. Our approach searches for find input scenarios which execute loops with the highest possible number of iterations. Producing erroneous values in case of **Casting to Smaller Data Types** may cause lost of data, leading to silent bugs. An example for this is the transformation of a number encoded with 32 bits into a number with 16 bits. The test goal detecting incorrect casting operations targets statements with explicit casting operations with the condition of maximizing/minimizing the operand of the cast function.

3 Conclusion

This paper introduces the idea of using evolutionary algorithms to generate test data for detecting critical defects in software functions. New instrumentation rules and the construction of a third component for the fitness function of have been developed for the classical Evolutionary Structural Testing. The authors argue that the application of search techniques to find test data, detecting critical defects, improves the test quality. Even in the case where no solution was found leading to an error, the evolutionary search creates test data that are close to the boundaries of error situations and therefore improves the confidence in the correctness of the software. Automatic test case design increases the effectiveness and efficiency of the test and thus reduce the overall development costs. Test data generation targeting critical defects can be used as an additional criterion to improve the quality of software products.

Acknowledgements. The work described has been performed within the SysTest project. The SysTest project is funded by the European Community under the 5th Framework Programme (GROWTH), project reference G1RD-CT-2002-00683.

References

[1] Wegener, J. ; Baresel, A.; Sthamer, H.: *Evolutionary test environment for automatic structural testing*. Information and Software Technology Special Issue on Software Engineering using Metaheuristic Innovative Algorithms. Vol. 43(14) (2001) 841 – 854

[2] Wegener, J.; Grochtmann, M.: *Verifying time constraints of real-time systems by means of evolutionary testing*. Real-Time Systems 15 (3). (1998) 275 – 298