Test-driven Fault Navigation for Debugging Reproducible Failures

Michael Perscheid and Robert Hirschfeld

Hasso Plattner Institute, University of Potsdam, Germany
firstname.lastname@hpi.uni-potsdam.de

Abstract. We present test-driven fault navigation as an interconnected debugging guide that integrates spectrum-based anomalies and failure causes. By analyzing failure-reproducing test cases, we reveal suspicious system parts, developers most qualified for addressing localized faults, and erroneous behavior in the execution history. The Paths tool suite realizes our approach: PathMap supports a breadth first search for narrowing down failure causes and recommends developers for help; PathFinder is a lightweight back-in-time debugger that classifies failing test behavior for easily following infection chains back to defects.

Debugging failing test cases, particularly the search for failure causes, is often a laborious and time-consuming activity. With the help of spectrum-based fault localization [2] developers are able to reduce the potentially large search space by detecting anomalies in tested program entities. However, such anomalies do not necessarily indicate defects and so developers still have to analyze numerous candidates one by one until they find the failure cause. This procedure is inefficient since it does not take into account how suspicious entities relate to each other, whether another developer is better qualified for debugging this failure, or how erroneous behavior comes to be.

We introduce a systematic top-down debugging process with corresponding new tools that integrate anomalies and failure causes. Developers are able to navigate from failures to causes by reproducing observable faults with the help of test cases. Afterwards, they can isolate anomalies within parts of the system, identify other developers for help, and understand erroneous behavior. Figure 1 summarizes our test-driven fault navigation process and its primary activities:

Reproducing failure As a precondition for all following activities, developers have to reproduce the observable failure in the form of at least one test case. Besides the beneficial verification of resolved failures, we require tests above all as entry points for analyzing erroneous behavior [6]. We have chosen unit test frameworks because of their importance in current development projects. Our approach is neither limited to unit testing nor does it require minimal test cases as proposed by some guidelines [1].

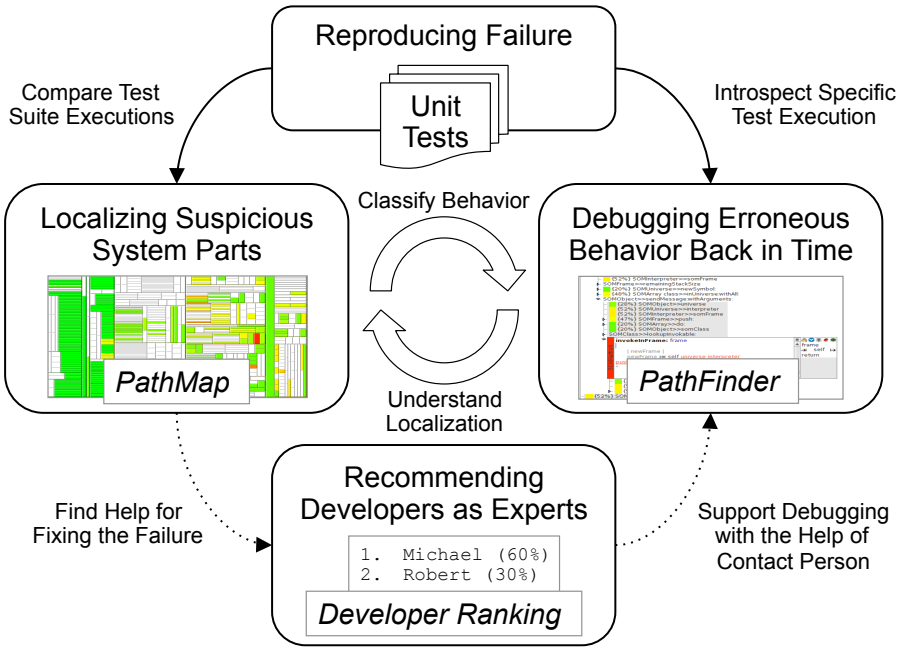

Fig. 1. Our debugging process guides with interconnected advice to reproducible failure causes in structure, behavior, and to corresponding experts.

Localizing suspicious system parts (Structural navigation) Having at least one failing test, developers can compare its execution with other test cases and identify structural problem areas. By analyzing failed and passed test behavior, possible failure causes are automatically localized within a few suspicious methods so that the necessary search space is significantly reduced. For supporting spectrum-based fault localization within the system's structure, we have developed an extended test runner called *PathMap* that provides a scalable tree map visualization and a low overhead analysis framework that computes anomalies at methods and refines results at statements on demand [3].

Recommending developers as experts (Team navigation) Some failures require expert knowledge of others so that developers can understand and debug faults more easily. By combining localized problem areas with source code management information, we provide a novel *developer ranking metric* that identifies the most qualified experts for fixing a failure [4]. Developers having changed the most suspicious methods are more likely to be experts than authors of non-infected system parts. We have integrated our metric within PathMap providing navigation to suitable team members.

Debugging erroneous behavior back in time (Behavioral navigation) For refining their understanding of erroneous behavior, developers explore the execu-

tion and state history of a specific test. To follow the infection chain back to the failure cause, they can start our lightweight back in time debugger, called *PathFinder* [5], either at the failing test directly or at arbitrary methods as recommended by PathMap. If anomalies are available, suspicious methods classify the executed trace and so ease the behavioral navigation to defects.

Besides our systematic process for debugging reproducible failures, the combination of unit testing and spectrum-based fault localization also provides the foundation for interconnected navigation with a high degree of automation. All activities and their anomalous results are affiliated with each other and so allow developers to explore failure causes from combined perspectives. Our tools support these points of view in a practical and scalable manner.

References

1. K. Beck. *Test-driven Development: By Example*. Addison-Wesley Professional, 2003.
2. James A. Jones, Mary Jean Harrold, and John Stasko. Visualization of Test Information to Assist Fault Localization. In *ICSE*, pages 467–477, 2002.
3. M. Perscheid, D. Cassou, and R. Hirschfeld. Test Quality Feedback: Improving Effectivity and Efficiency of Unit Testing. In *C5*, 2012.
4. M. Perscheid, M. Haupt, R. Hirschfeld, and H. Masuhara. Test-driven Fault Navigation for Debugging Reproducible Failures. In *JSSST*, 2011.
5. M. Perscheid, B. Steinert, R. Hirschfeld, F. Geller, and M. Haupt. Immediacy through Interactivity: Online Analysis of Run-time Behavior. In *WCRE*, 2010.
6. B. Steinert, M. Perscheid, M. Beck, J. Lincke, and R. Hirschfeld. Debugging into Examples: Leveraging Tests for Program Comprehension. In *TestCom/FATES*, 2009.