# Wo ist der Fehler und wie wird er behoben? Ein Experiment mit Softwareentwicklern.

Marcel Böhme[1], Ezekiel O. Soremekun[2], Sudipta Chattopadhyay[3], Emamurho Ugherughe[4], Andreas Zeller[5]

## 1 Zusammenfassung

Dieser Beitrag ist eine gekürzte, deutsche Version unseres Artikels "Where is the Bug and How is it Fixed? An Experiment with Practitioners" publiziert in dem Berichtsband des elften gemeinsamen Treffens der European Software Engineering Conference und des ACM SIGSOFT Symposium on the Foundations of Software Engineering [B7].

Seit mehr als zwanzig Jahren hat die Forschung eine Reihe von Ansätzen zur automatischen Lokalisierung, Erklärung und Behebung von Programmfehlern entwickelt. In letzter Zeit haben Forscher auch einige Benchmarks bereitgestellt, die zur empirischen Auswertung solcher Ansätze genutzt werden können. Mit Hilfe solcher Benchmarks können Wissenschaftler empirisch fundierte Behauptungen über die Effektivität solcher Werkzeuge und Techniken aufstellen. Zum Beispiel wird oft ein Fehlerlokalisierungswerkzeug als effektiver bewertet, welches einer Anweisung, die während der Fehlerbehebung geändert wurde, einen höheren Rang zuweist. Dabei wird angenommen, daß ein Entwickler dieselbe Anweisung als Fehlerursache identifizieren würde. Eine automatische Fehlerbehebung wird als effektiver bewertet, wenn daraufhin alle Regressionstests erfolgreich durchlaufen. Hier wird angenommen, daß ein Entwickler eine solche fehlerbehebende Programmänderung akzeptieren oder gar selbst durchführen würde. Leider ist der Debuggingprozess in der Realität nicht ganz so einfach, besonders nicht für den Menschen. Wir stellen einen anderen Benchmark zur Verfügung; einen Benchmark der eine Realitätskontrolle erlaubt.

In Anbetracht der Komplexität des Debuggingprozesses sollte man doch annehmen, daß neue Werkzeuge standardmäßig mithilfe von Nutzerstudien ausgewertet werden: Passt das Werkzeug in den Prozess? Ist es wertschaffend? Inwiefern? Dennoch ist nicht wirklich gut erforscht, wie Menschen eigentlich Fehler lokalisieren, erklären und beheben. Zwischen 1981 und 2010 haben Parnin und Orso gerade mal eine Handvoll von Artikeln identifiziert, die Ergebnisse von Nutzerstudien präsentiert haben. Keiner dieser Artikel involvierte sowohl richtige Entwickler als auch echte Programmfehler. Seit Ende 2010 haben auch wir gerade einmal drei Artikel gefunden, die beides involvierten.

[1] Monash University, Faculty of Information Technology, Melbourne, Australia marcel.boehme@acm.org
[2] Saarland University & CISPA, Saarbrücken, Germany soremekun@st.cs.uni-saarland.de
[3] Singapore University of Technology and Design, Singapore, Singapore sudipta_chattopadhyay@sutd.edu.sg
[4] SAP, Berlin, Germany emamurho@gmail.com
[5] Saarland University & CISPA, Saarbrücken, Germany zeller@st.cs.uni-saarland.de

In diesem Artikel versuchen wir nicht ein bestimmtes Werkszeug zu evaluieren. Stattdessen beleuchten wir den gesamten Debuggingprozess, wie er von Softwareentwicklern real durchgeführt wird. Wir untersuchen im Einzelnen wie lange es typischerweise dauert, wie schwierig die Fehlersuche und -behebung wahrgenommen wird und welche Strategien bei der Fehlersuche und -behebung angewendet werden. Für unseren Benchmark entnehmen wir die fehlerverursachenden Anweisungen, Erklärungen des Fehlerhergangs und die eigentlichen Fehlerbehebungen (d.h. Patches) welche *Entwickler* hervorbrachten. Wir nutzten 27 reale Fehler von CoREBench [BR14] die systematisch aus den 10.000 jüngsten Commits and den dazugehörigen Fehlerberichten extrahiert wurden. Wir beauftragten 12 Softwareingenieure aus 6 Ländern für jeden dieser Programmfehler i) die fehlerverursachenden Anweisungen zu lokalisieren, ii) den Fehlerhergang zu beschreiben und iii) den Fehler zu beheben.

*Erkenntnisse*. Nach unserem besten Wissen haben wir den ersten Beleg dafür gefunden, daß der Debuggingprozess überhaupt automatisiert werden kann. In unserem Experiment haben unterschiedliche Entwickler grundsätzlich dieselben Fehlerursachen lokalisiert und die gleiche Erklärung zu dem Fehlerhergang abgegeben. Wenn sich sogar unsere menschlichen Teilnehmer widersprächen, wie könnte eine Maschine dann die "richtige" Fehlerursache und -diagnose etablieren? Wir fanden auch heraus, daß viele der eingereichten Fehlerbehebungen (d.h. Patches) eigentlich inkorrekt waren: Obwohl 97% aller Patches plausibel waren, das heißt alle Regressionstests liefen erfolgreich durch, waren lediglich 63% wirklich korrekt, das heißt sie bestanden auch unseren Codereview. Das motiviert natürlich weitführende Forschung an der automatischen Fehlerbehebung. Drei von vier inkorrekten Patches sind inkorrekt, weil sie neue Fehler einführen oder den eigentlichen Fehler nicht vollständig beheben. Das motiviert Untersuchungen der automatischen Regressionstestgenerierung.

*Benchmark*. Da unsere Teilnehmer grundsätzlich bei den essentiellen Fehlermerkmalen übereinstimmen, kann man deren Artefakte als Grundwahrheit behandeln. Aus diesem Grund stellen wir die gewonnen Studiendaten als Benchmark mit dem Namen DbgBench bereit. DbgBench kann in kosteneffektiven Nutzerstudien genutzt werden um zu untersuchen wie sich die Zeit, Schwierigkeit und Patchkorrektheit mit dem neuen Werkzeug verbessert. DbgBench kann auch genutzt werden um ohne einer Nutzerstudie auszuwerten wie gut sich das neuartige Werkzeug gegen unsere menschlichen Teilnehmer misst wenn es um Aufgaben wie das Lokalisieren, Erklären und Beheben von Programmfehlern geht. Mehr Informationen zu DbgBench finden Sie unter folgender Adresse: `https://dbgbench.github.io`.

## Literaturverzeichnis

[B7] Böhme, Marcel; Soremekun, Ezekiel O.; Chattopadhyay, Sudipta; Ugherughe, Emamurho; Zeller, Andreas: Where is the Bug and How is it Fixed? An Experiment with Practitioners. In: Proceedings of the 11th Joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. ESEC/FSE, S. 117–128, 2017.

[BR14] Böhme, Marcel; Roychoudhury, Abhik: CoREBench: Studying Complexity of Regression Errors. In: Proceedings of the 23rd ACM/SIGSOFT International Symposium on Software Testing and Analysis. ISSTA, S. 105–115, 2014.