Der Mikroprozess von Programmierfehlern

Sebastian Jekutsch

Arbeitsgruppe Software Engineering
Institut für Informatik
Freie Universität Berlin
Takustr. 9, 14195 Berlin, Germany
jekutsch@inf.fu-berlin.de

Die Softwaretechnik beschäftigt sich auffallend wenig mit den Programmier*fehlern*, die Softwaredefekten zugrunde liegen, umso mehr jedoch damit, wie man einmal eingebaute Defekte nachher wieder findet und ausbaut. Es existieren zwar einige Praktiken, um entstandene Defekte frühzeitig zu erkennen, das Verhindern von Fehlern, auch von Programmierfehlern, ist aber wenig untersucht.

Das Fehlermachen ist ein vermutlich individueller Prozess. Auf dieser Annahme basierend hat der Persönliche Softwareprozess (PSP) [1] eine Methode entwickelt, die eine systematische Analyse der eigenen Fehler bietet. Diese findet mittels konsequent geführter Aufzeichnungen statt, neben der eigentlichen Programmierarbeit. Manuelle Aufzeichnungen sind jedoch häufig fehlerhaft und dermaßen aufwändig, dass PSP in der Praxis kaum eingesetzt wird, auch wenn deren positive Wirkung nachgewiesen wurde [4].

Eine Reflexion über die eigenen Programmierfehler ist jedoch auch nachträglich möglich. Dazu bedarf es aber einer Nachbetrachtung der eigenen Arbeit und einer werkzeugunterstützten Analyse des eigenen *Mikroprozesses*, also der einzelnen, tatsächlichen Arbeitsschritte bei der Programmierung [3]. Mit Hilfe eines automatisch aufgezeichneten Mikroprozesses, des Wiederabspielens der persönlichen Arbeitsschritte bestimmter "schlecht" gelaufener Programmiersitzungen und einer möglichst automatischen Analyse typischer und verdächtiger Fehlermuster kann eine Fehleranalyse auch ohne umständliche Selbstaufzeichnung gelingen.

Ich möchte in meinem **Promotionsvorhaben**, für das noch **zwei Jahre** Zeit bleiben, die Chancen dieser Idee evaluieren. Meine **Forschungsfrage** ist: Wie kann man einem Programmierer werkzeuggestützt helfen, Fehler zu vermeiden? Die vier **Thesen** sind, dass (1.) eine Reflexion über die eigene Arbeit beim Lernen helfen kann, indem (2.) nachträglich eine geeignete, automatisch erstellte Mikroprozessaufzeichnung präsentiert wird. Dieses Prinzip wird (3.) zumindest für Belastungsfehler funktionieren, d.h.

Fehler aufgrund widriger Umstände (Stress, Umgebung) oder Fehler aufgrund individueller Überlastung (zu schwieriges Problem), denn hier paart sich (4.) das Fehlermachen mit typischen Verhaltensmustern, Auffälligkeiten und Symptomen, die im präsentierten Mikroprozess erkennbar sind. Diese sind sozusagen Anti-Praktiken des Programmierens.

Das **Vorgehen** wird sich an dem klassischen Zyklus **qualitativer Forschung** „Beobachten - Analysieren - Hypothesenbilden - Validieren“ halten. Es bedarf eines im Wesentlichen neuen Instrumentariums und Vokabulars, da es wenige Vorarbeiten im Bereich des Mikroprozesses des Programmierens und der Programmierfehler gibt.

Derzeit werden zur Analyse einige Programmiersitzungen aufgezeichnet, sowohl mit Bildschirmvideo und Webcam, als auch durch Abspeichern einzelner Arbeitsschritte bei einer Entwicklungsumgebung [5]. Ein besonderer Schwerpunkt liegt darauf, ob die Art der Erstellung von Codefragmenten sich in problematischen Situationen von der in erfolgreichen Situationen unterscheidet [2].

Die Forschung zur Entdeckung und Beschreibung typischer Fehlerverhaltensmuster ist also klar **empirisch** in ihrer Natur, beinhaltet aber auch die Konstruktion eines Werkzeuges. Softwaretechnische Forschung muss meiner Meinung nach immer einen empirischen Anteil haben. Entweder ist die Effektivität einer vorgeschlagenen Prozessverbesserung zu belegen oder – wie in dieser Arbeit – das Problemfeld zunächst ethnographisch zu beleuchten. Softwaretechnische Fragestellungen drehen sich letztendlich um das Verhalten, die Fähigkeiten und die Kooperation von Menschen. Dies ist erst Recht im Bereich der Softwareentwicklung nicht ausreichend verstanden. Softwaretechnikarbeiten sind in Folgerung dessen meist auch **interdisziplinär**. Dies macht die Forschung spannend, in vielerlei Hinsicht.

Literatur

[1] Watts S. Humphrey. *A Discipline for Software-Engineering.* Addison-Wesley, 1995.

[2] Sebastian Jekutsch. An annotation scheme to support analysis of programming activities. In *Papers presented at Workshop on Ethnographies of Code.* Infolab21, Lancaster University, 2006.

[3] Lutz Prechelt, Sebastian Jekutsch, and Philip Johnson. Actual process: A research program. Technical Report B-06-02, Institut f. Informatik, Freie Universität Berlin, March 2006.

[4] Lutz Prechelt and Barbara Unger. An experiment measuring the effects of personal software process (PSP) training. *IEEE Trans. Softw. Eng.*, 27(5):465–472, 2001.

[5] Frank Schlesinger and Sebastian Jekutsch. ElectroCodeoGram: An environment for studying programming. In *Papers presented at Workshop on Ethnographies of Code.* Infolab21, Lancaster University, UK, 2006.