Das Verhalten von Entwicklern bei der Durchführung von Programmiertätigkeiten: Erfassen, Visualisieren und Verstehen

Martin Schröer Rainer Koschke

Universität Bremen

Zusammenfassung

Wir beschäftigen uns mit dem Thema des Programmverstehens, um konkrete Zusammenhänge zwischen spezifischen Entwicklungstätigkeiten und dazu erforderlichen Informationen sowie von Entwicklern dabei eingesetzten Lösungsansätzen zu analysieren mit dem Ziel, diese Erkenntnisse für eine kontextabhängige, bedarfsgerechte und individuelle Unterstützung von Entwicklern bei der Bewältigung von Programmieraufgaben einzusetzen - sowohl in Form von Handlungsempfehlungen als auch in Form eines Softwaretools. Ein von uns zu diesem Zweck geschaffenes Framework zur Erfassung, Visualisierung und Auswertung von Interaktionen von Entwicklern mit der IDE zeigte in einer ersten Studie u.a. unterschiedliche, charakteristische Phasen bei der Bewältigung von Programmieraufgaben, die Potential aufzeigen, um für eine solche Unterstützung genutzt zu werden.

Schlüsselbegriffe: Programmverstehen, Entwicklerverhalten, IDE Interaktionen, Informationsbedarf, Visualisierung, Aktive Blindheit, Recommending Tools

Motivation und Zielsetzung

Im Lebenszyklus einer Software nimmt die Softwarewartung oft mehr als die Hälfte des aufzuwendenden Aufwands ein [1], dies gilt um so mehr für modernere Softwarelösungen und -architekturen, die immer mehr auf Vernetzung und (Wieder-)Verwendung bestehender Elemente als auf die Neuentwicklung einzelner Komponenten und Funktionen setzen. Entwickler in der Softwarebranche müssen sich demzufolge heutzutage häufig mit mit bereits existierendem Quellcode auseinandersetzen, etwa um bestehende Funktionalitäten an neue Anforderungen anzupassen, Fehler zu beheben oder bestehende Bausteine (z.B. Softwarebibliotheken) in ein Softwaresystem zu integrieren. In den Erwerb von Wissen über solche Programme oder Programmteile investieren Entwickler oft sehr viel Aufwand. Der in diesem Zusammenhang notwendige Prozess, Wissen und ein Verständnis über ein Programm zu erlangen, wird allgemein als Programmverstehen [13] bezeichnet. Innerhalb dieses Themengebietes existieren unterschiedliche Erklärungsansätze, beginnend etwa mit Modellen, wie Wissen über ein Programm vermutlich gewonnen und in der Vorstellung von Entwicklern organisiert wird [2, 10, 15, 17, 18, 19]. Weitere Forschungsansätze widmeten sich der Beschreibung der konkreten Strategien, welche durch Entwickler bei der Bewältigung von Programmiertätigkeiten angewandt werden, sowohl im Allgemeinen [16] als auch in konkreten Entwicklungsszenarien [3, 4]. Jüngere Arbeiten konzentierten sich unter anderem auf spezifische Aspekte des Programmverstehens, etwa den konkreten Wissensbedarf von Entwicklern bei der Durchführung unterschiedlicher Aufgaben [5, 9, 14], die Eigenarten bestimmter Programmieraufgaben [4, 12], sowie auf den Einfluss von individueller Programmiererfahrung auf die unterschiedlichen Verständnisstrategien [6]. Durch die Einführung der interdisziplinären *Information Foraging Theory* [11] in das Verständnis, wie eine Suche nach Wissen und Informationen zum Programmverstehen konkret durchgeführt wird [7], eröffneten sich neue Ansätze, um die daran beteiligten Arbeitsschritte zu erfassen und zu interpretieren [8].

Auf Basis dieser neueren Erkenntnisse beabsichtigen wir, charakteristische Zusammenhänge zwischen bestimmten Programmiertätigkeiten und -aufgaben, individuellen Merkmalen von Entwicklern wie etwa deren Programmiererfahrung und den zur Bewältigung von Entwicklungstätigkeiten erforderlichen Informationen zu erfassen und zu analysieren, um auf dieser Grundlage Ansätze zur Vereinfachung und Optimierung des Programmverstehens entwicklen zu können. Dies soll sowohl in Form von situationsgerechten Handlungsempfehlungen als auch in Form eines Softwaretools erfolgen.

Forschungsfragen

Innerhalb unserer Arbeit beschäftigen wir uns insbesondere mit den folgenden konkreten Forschungsfragen.

1. Wie viel Programmverstehen ist für einen bestimmten Aufgabenkontext erforderlich? Führt ein geringes Programmverständnis zu mehr Fehlern?
2. Welchen Einfluss hat der Rahmenkontext, etwa die konkret zu bewältigende Aufgabe oder die individuelle Programmiererfahrung auf den Informationsbedarf und die eingesetzten Strategien zum Programmverstehen?
3. Welches Wissen wird für Programmverstehen in bestimmten Kontexten benötigt, in welchen Quellen ist es vorhanden und wie wird es durch Entwickler auch als verfügbar wahrgenommen?
4. Welche Faktoren beeinflussen die Wahrnehmung bei der Suche nach benötigten Informationen und tragen zum Auftreten äktiver Blindheit", dem bewussten Unterdrücken der Wahrnehmung bestimmter Informationen bei?
5. Kann durch ein systematisches kontextabhängiges Vorgehen, etwa einem Handlungsplan für bestimmte Aufgabenstellungen das Programmverstehen erleichtert werden?

6. Welche konkreten Arbeitsschritte werden von Entwicklern in welchen Phasen des Programmverstehens durchgeführt?
7. Wie werden die konkreten Arbeitsschritte beim Programmverstehen durch Entwickler priorisiert und organisiert?
8. Warum werden Hilfsmittel zum Programmverstehen in der Praxis nicht eingesetzt?
9. Welche Aspekte unterschiedlicher Entwicklungsaufgaben und Entwicklungssituationen sind für die jeweiligen Kontexte charakteristisch und wie lassen sie sich feststellen? Wie können (Recommendation-) Systeme diese Aspekte nutzen, um bedarfsgerechte Informationen anbieten zu können?
10. Welchen Einfluss hat die Organisation und Darstellung von Informationen auf die Wahrnehmung durch Entwickler?

Um diese Forschungsfragen zu beantworten, verfolgen wir einen mehrstufigen Ansatz. Zunächst erfassen und analysieren wir die einzelnen Arbeitsschritte von Entwicklern bei der Durchführung unterschiedlicher Entwickungsaufgaben. Anhand der so gewonnenen Daten leiten wir Zusammenhänge und Abhängigkeiten ab, etwa über die jeweils eingesetzten Strategien zum Programmverstehen oder den spezifischen Informationsbedarf bestimmter Entwicklungsaufgaben wie beispielsweise Fehlerbehebung oder Featureintegration. Die daraus gewonnenen Erkenntnisse werden dann in entsprechend konzipierten Werkzeugen umgesetzt, etwa in Form eines Recommender Tools, welches im Gegensatz zu bisherigen Ansätzen u.a. insbesondere den aktuellen Entwicklungsschritt des agierenden Entwicklers berücksichtigt, um aus der Vielzahl oft bereits verfügbarer Informationen kontextabhängig bevorzugt solche anzubieten (adaptives Filter), von denen angenommen werden kann, dass sie in der gegebenen Situation für den individuellen Entwickler am hilfreichsten sind: Es gilt, die richtige Information zur richtigen Zeit anzubieten.

Bisherige Ergebnisse

Im bisherigen Verlauf haben wir ein Framework zur Erfassung und Visualisierung von Interaktionen von Entwicklern mit der Eclipse IDE implementiert und eine erste Validierungsstudie durchgeführt.

Aufzeichnung von Entwicklerverhalten In Form eines Plug-Ins für die Eclipse IDE ermöglicht unser Framework die Erfassung nahezu jeglicher Interaktionen, welche ein Entwickler mit der IDE vornimmt. Diese können in Form einer Aufzeichnung gespeichert werden, um daraus Zusammenhänge zwischen den Handlungen eines Entwicklers zu unterschiedlichen Zeitpunkten einer Entwicklungstätigkeit zu analysieren.

Visualisierung von aufgezeichnetem Verhalten Um eine Auswertung in Form einer Sichtung schnell und effizient durchzuführen zu können, haben wir eine Visualisierung implementiert, welche die aufgezeichneten Interaktionen in Bezug auf ihren zeitlichen Verlauf und ihre Verortung im Programm darstellen kann. Diese Form der Repräsentation dient insbesondere dem Ziel, eine vereinfachte Vergleichbarkeit zwischen Aufzeichnungen zu ermöglichen, um etwa Gemeinsamkeiten oder Unterschiede schnell und effizient sichten zu können.

Drei Phasen der Entwicklungstätigkeit Bei der Auswertung der aufgezeichneten Daten aus der Studie fiel uns auf, dass obwohl die teilnehmenden Entwickler sehr unterschiedliche Verhalten in Bezug auf ihre konkreten Lösungsansätze und die zur Lösung der Aufgabe investierte Zeit zeigten, sie jedoch alle auffallend charakteristische Interaktionsphasen in ihrer Durchführung aufwiesen. Diese drei Phasen, welche wir als Ïnvestigation", "Bearbeitungünd "Verfikation"bezeichnen, unterschieden sich auffallend durch das zu beobachtende Navigationsverhalten und darin, welche konkreten Interaktionen in der jeweiligen Phase durchgeführt wurden: Etwa trat in der Investigationsphase deutlich mehr Navigationsverhalten als in den beiden anderen Phasen auf, wobei die durchgeführten Navigationen im Quellcode auch eine höhere Latenz, d.h. mehr Zeit zwischen Bewegungen im Code zeigten als in den anderen beiden Phasen. Dies lässt vermuten, dass sich die Entwickler in dieser Phase zunächst einen Überblick über den zu verstehenden Programmcode verschaffen wollten, während sie in den anderen Phasen augenscheinlich zielgerichtet und in schnellerer Abfolge einem zuvor gemachten Plan zur Ausführung einzelner Änderungen zu folgen schienen. Auch unterschied sich die Art der in den jeweiligen Phasen gezeigten Interaktionen: Während in der Investigationsphase nahezu keine Hervorhebung oder Änderungen an dem des Programmcode vorgenommen wurden, zeichnete sich die Bearbeitungsphase besonders durch solche Interaktionen aus. In der Verifikationsphase nahm die Anzahl der Navigationsverhalten im Vergleich dazu wieder zu, auch wurde das Programm hier häufiger ausgeführt, vermutlich, um die vorgenommenen Änderungen auf die gewünschte Wirkung zu testen. In dieser Beobachtung sehen wir großes Potential, denn eine Bestätigung und weitere Charakterisierung dieser Phasen etwa durch die Beobachtung spezifischer Interaktionsmuster könnte es ermöglichen, dass diese während einer Entwicklungstätigkeit durch ein Programm detektiert und etwa in Form einer Bereitstellung bedarfsgerechter Informationen für den Entwickler genutzt werden könnten.

Aktive Blindheit Im Rahmen unserer Studie konnten wir darüber hinaus feststellen, dass Entwickler anscheinend dazu neigten, Informationen, die zur Bewältigung einer Programmieraufgabe oder im Zusammenhang dazu zur Verfügung gestellt werden, zu ignorieren, wenn diese nicht im richtigen Kontext oder zum falschen Zeitpunkt zur Verfügung gestellt werden. Wird etwa eine Information zu früh zur Verfügung gestellt und von dem Entwickler zu diesem Zeitpunkt als irrelevant beurteilt, scheint es so, als würde ihre bewusste Wahrnehmung auch zu einem späteren Zeitpunkt, zu dem sie für die Erfüllung der Aufgabe relevant wäre, aktiv unterdrückt.

Nächste Schritte Unsere bisherigen Beobachtungen sollen in weiteren, breiter angelegten Studien mit umfangreicheren und komplexeren Programmen weiter evaluiert werden. Insbesondere beschäftigt uns dabei die Frage, ob die drei zuvor beschriebenen Phasen auch bei komplexeren Programmen, die etwa aus mehreren modularen Teilen bestehen, in gleicher Weise auftreten, oder ob möglicherweise auch mehrstufige oder rekursive Verstehensprozesse und entsprechende Phasen zu beobachten sind.

Anforderungen an den Einsatzkontext

Um unsere bisherigen Erkenntnisse bestätigen und weiter vertiefen zu können, ist es erforderlich, Entwickler mit unterschiedlichem Erfahrungsstand bei der Durchführung unterschiedlicher Programmieraufgaben zu beobachten, um so weitere Daten zu sammeln. Zu diesem Zweck kann unser Mimesis Framework in Eclipse IDEs integriert werden, wo es dann transparent für den Nutzer im Hintergrund läuft, Interaktionen erfasst und aufzeichnet. Um die so erfassten Daten auswerten zu können, brauchen wir auch Zugriff auf die vom jeweiligen Entwickler bearbeiteten Quellcodes, um die aufgezeichneten Interaktionen im Code verorten und interpretieren zu können. Es ist denkbar, dass solche Aufzeichnungen auch direkt am Arbeitsplatz eines Entwicklers durchgeführt werden, indem das Plug-In in seine gewohnte Entwicklungsumgebung installiert wird, der Austausch der Aufzeichnungen und Änderungen am Quellcode kann über ein entsprechendes Versionskontrollsystem vorgenommen werden.

Literatur

[1] Christoph Bommer, Markus Spindler, and Volkert Barr. *Softwarewartung: Grundlagen, Management und Wartungstechniken*. dpunkt. verlag, 2016.

[2] Ruven Brooks. Towards a theory of the comprehension of computer programs. *International journal of man-machine studies*, 18(6):543–554, 1983.

[3] Valentina Grigoreanu, Margaret Burnett, Susan Wiedenbeck, Jill Cao, Kyle Rector, and Irwin Kwan. End-user debugging strategies: A sensemaking perspective. *ACM Trans. Comput.-Hum. Interact.*, 19(1):5:1–5:28, May 2012. ISSN 1073-0516. doi: 10.1145/2147783.2147788. URL `http://doi.acm.org/10.1145/2147783.2147788`.

[4] Andrew J Ko, Brad A Myers, Michael J Coblenz, and Htet Htet Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on software engineering*, 32(12), 2006.

[5] Andrew J Ko, Robert DeLine, and Gina Venolia. Information needs in collocated software development teams. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 344–353. IEEE, 2007.

[6] Andrew Jensen Ko and Bob Uttl. Individual differences in program comprehension strategies in unfamiliar programming systems. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 175–184. IEEE, 2003.

[7] Joseph Lawrance, Rachel Bellamy, Margaret Burnett, and Kyle Rector. Using information scent to model the dynamic foraging behavior of programmers in maintenance tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1323–1332. ACM, 2008.

[8] Joseph Lawrance, Christopher Bogart, Margaret Burnett, Rachel Bellamy, Kyle Rector, and Scott D Fleming. How programmers debug, revisited: An information foraging theory perspective. *IEEE Transactions on Software Engineering*, 39(2):197–215, 2013.

[9] Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. On the comprehension of program comprehension. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(4):31, 2014.

[10] Nancy Pennington. Comprehension strategies in programming. In *Empirical studies of programmers: second workshop*, pages 100–113. Ablex Publishing Corp., 1987.

[11] Peter Pirolli and Stuart Card. Information foraging. *Psychological review*, 106(4):643, 1999.

[12] Tobias Röhm, Rebecca Tiarks, Walid Maalej, and Rainer Koschke. How do professional developers comprehend software. In *International Conference on Software Engineering.*, pages 81–90. ACM, 2012. **ACM SIGSOFT Distinguished Paper Award**.

[13] Spencer Rugaber. Program comprehension, July 1995. College of Computing, Georgia Institute of Technology.

[14] Caitlin Sadowski, Kathryn T Stolee, and Sebastian Elbaum. How developers search for code: a case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 191–201. ACM, 2015.

[15] Ben Shneiderman and Richard Mayer. Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International Journal of Parallel Programming*, 8(3):219–238, 1979.

[16] Jonathan Sillito, Gail Murphy, and Kris De Volder. Asking and answering questions during a programming change task. *IEEE Transactions on Software Engineering*, 34(4):434–451, 2008.

[17] Elliot Soloway, Beth Adelson, and Kate Ehrlich. Knowledge and processes in the comprehension of computer programs. *The nature of expertise*, pages 129–152, 1988.

[18] Anneliese von Mayrhauser and A Marie Vans. Industrial experience with an integrated code comprehension model. *Software Engineering Journal*, 10(5): 171–182, 1995.

[19] Anneliese von Mayrhauser and A Marie Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, 1995.