Jahresbericht 2017 des Lehrstuhls für Informatik 2 (Programmiersysteme)

Anschrift: Martensstr. 3, 91058 Erlangen
Tel.: +49-9131-85-27621
Fax: +49-9131-85-28809
E-Mail: info@i2.informatik.uni-erlangen.de

Ordinarius:
Prof. Dr. Michael Philippsen
Honorarprofessor:
Hon.-Prof. Dr.-Ing. Bernd Hindel
Hon.-Prof. Dr.-Ing. Detlef Kips
Emeritus:
Prof. em. Dr. Hans Jürgen Schneider
Sekretariat:
Margit Zenk
Wiss. Mitarbeiter:
Michael Baer, M. Sc.
Dipl.-Inf. Thorsten Blaß
Tobias Feigl, M. Sc. (ab 01.10.2017)
Florian Jung, M. Sc. (ab 01.12.2017)
Marius Kamp, M. Sc.
Dipl.-Math. Jakob Krainz (bis 30.06.2017)
Patrick Kreutzer, M. Sc.
Florian Mayer, M. Sc. (ab 01.08.2017)
Dr.-Ing. Christopher Mutschler
Dipl.-Inf. Daniela Novac
Dr.-Ing. Norbert Oster, Akad. ORat
IT-Betreuer:
Dipl.-Ing. (FH) Helmut Allendorf
Manfred Uebler
Gast:
Dr.-Ing. Josef Adersberger
Florian Lautenschlager, M. Sc.
Externes Lehrpersonal:
Dr.-Ing. Klaudia Dussa-Zieger
Dr.-Ing. Martin Jung

Seit Prof. Dr. Michael Philippsen 2002 die Leitung des 1972 gegründeten Lehrstuhls übernommen hat, stehen **Programmiersysteme für heterogene Multicore-Rechner** im Zentrum der Forschungsarbeiten des Lehrstuhls.

Bis 2014/2015 standen dabei vor allem systemnahe Forschungsthemen im Vordergrund, die näher an der Hardware, den Laufzeitsystemen, den unterliegenden Betriebssystemen und der Frage der Programmierung dieser Rechensysteme ausgerichtet waren. Es ging dem Lehrstuhl bisher insbesondere darum, das Parallelisierungspotential einer Anwendung verlustarm an die in der Hardware vorhandene Parallelität anzupassen und die von der Hardware bereitgestellten Möglichkeiten effizient nutzbar zu machen. Weil diese systemnahen Fragen noch immer von aktueller Bedeutung sind, werden sie auch in Zukunft in einigen Projekten des Lehrstuhls bearbeitet.

Jetzt konzentriert sich der Lehrstuhl auf Forschungsthemen, die ingenieurwissenschaftliche Antworten für den Software-Ingenieur liefern, der im Rahmen industrieller Software-Entwicklung für Multicore-Rechner, für daraus bestehende verteilte Systeme, für paralleles Cloud-Computing sowie für vernetzte eingebettete Systeme parallele Software entwickelt.

1 Forschungsschwerpunkte

Die Eckpunkte der vom Lehrstuhl bearbeiteten **Programmiersystemforschung** lassen sich wie folgt abstecken:

(a) Aufgrund des weiterhin dringenden Bedarfs entwickeln und evaluieren wir auch zukünftig **Programmiermodelle**, die heterogene parallele Komponenten mit einer einheitlichen Schnittstelle versehen und portablen Code ermöglichen, der auf unterschiedlichen Konfigurationen (aus Multicores, GPUS, Acceleratoren, FPGAs etc.) lauffähig ist. Nur so können sich Investitionen in parallele Software längerfristig rechnen.

(b) Es wird weiter an wirtschaftlicher Bedeutung gewinnen, vorhandene Software für Multicore-Rechner zu parallelisieren. Wir erforschen neuartige Werkzeuge, die auf **Code-Repositories** arbeiten, diese analysieren und den Entwickler bei **Migration, Refaktorisierung und Parallelisierung** unterstützen. In speziellen Situationen und Kontexten ist eine automatische Parallelisierung möglich und effektiv.

(c) Wegen der durch die Parallelität und den Synchronisierungsbedarf gestiegenen Komplexität muss der Entwickler stärker als bisher bei der Neu- und Weiterentwicklung von parallelem Code unterstützt werden, um Fehler im Vorfeld zu vermeiden und frühzeitig zu erkennen und zu beheben, auch weil diese Tätigkeiten nicht länger dem systemnah arbeitenden Spezialisten vorbehalten sind. Wir entwickeln dazu **schnellere, interaktive, inkrementelle und ggf. selbst parallel ausgeführte Code-Analysen**, durch die

nicht nur Wettlaufsituationen, konkurrierende Ressourcenzugriffe etc. entdeckt werden, sondern die dem Programmierer auch punktgenau und interaktiv in der Entwicklungsumgebung Verbesserungsvorschläge für den gerade in der Entwicklung befindlichen Code liefern.

(d) Im Lebenszyklus von paralleler Software stellen sich durch den Indeterminismus der Nebenläufigkeit auch beim **Test paralleler Programme**, bei ihrer Qualitätssicherung, bei der Sicherstellung der **Code-Authentizität** sowie bei ihrem Betrieb und ihrer **Diagnose** neuartige und an Bedeutung gewinnende Fragen. Wir untersuchen, wann paralleler Code ausreichend gut getestet ist, wie man parallele Programme gegen Angriffe schützen kann, wie Testdaten für parallele Programme erzeugt werden können, wie Ursachen von erratischem Verhalten des parallelen Codes systematisch gefunden werden können etc. Diese Fragen erhalten durch das Vorhandensein der Parallelität im Code eine neue Dimension, die etablierte Techniken aus dem Software-Engineering nur unzureichend beantworten können.

Der Lehrstuhl arbeitet stets **Programm-Code-basiert** und erstellt funktionsfähige **Prototypen** der erforschten Werkzeuge. Uns ist dabei eine **qualitative und quantitative Evaluation** sehr wichtig.

2 Forschungsprojekte

2.1 Inkrementelle Code-Analyse

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
Jakob Krainz
Laufzeit: 01.04.2012 – 30.06.2017

Um sicherzustellen, dass Fehler im Programmdesign schon früh im Entwicklungsprozess gefunden werden, ist es nützlich, Fehler möglichst schon während des Editierens des Programms zu finden. Dazu sollte die verwendete Analyse so schnell sein, dass ein interaktiver Einsatz möglich ist. Eine Möglichkeit, dies umzusetzen, ist der Einsatz von inkrementeller Analyse, bei der die Analyseergebnisse von Teilen eines Programms zu Gesamtergebnissen kombiniert werden. Vorteil von inkrementeller Programmanalyse ist, dass bei kleineren Änderungen ein großer Teil der Analyseergebnisse wieder verwendet werden kann, wie auch z.B. Analyseergebnisse von u.U. verwendeten Programmbibliotheken. Hierdurch kann der Analyseaufwand drastisch reduziert werden, wodurch die Analyse interaktiv nutzbar wird.

Unsere Analyse basiert darauf, für (Teile von) einzelnen Funktionen zu bestimmen, welche Auswirkungen die Ausführung auf den Zustand des Programms zur Laufzeit haben kann. Hierzu wird der Laufzeitzustand eines Programms abstrakt durch einen Graphen dargestellt, der die im Speicher befindlichen Variablen und Objekte und ihre Verzeigerung beschreibt. Die Funktion wird symbolisch ausgeführt und dabei wird bestimmt, welche Änderungen an dem Laufzeitzustand bzw. an dem diesen darstellenden Graphen verursacht werden können. Um die Effekte von Hintereinanderausführung von Programmteilen, Funktionsaufrufen, Schleifen, etc. zu bestimmen, können die Änderungsbeschreibungen der Programmteile dann zu größeren Änderungsbeschreibungen kombiniert werden. Die Analyse geht dabei Bottom-Up vor, analysiert also eine aufgerufene Funktion vor der aufrufenden Funktion (wobei Rekursion analysierbar ist).

In 2017 lag der Schwerpunkt unserer Forschung darauf, die eingesetzten Algorithmen und Datenstrukturen weiter zu entwickeln. Zusätzlich zu einer besseren Skalierbarkeit der Analyse bei großen zu analysierenden Programmen und einer Weiterentwicklung der inkrementellen Analyse (bei der die Analyseergebnisse von unveränderten Programmteilen weiterverwendet werden), lag der Fokus darauf, die Analyse anschaulich zu dokumentieren, ihre Korrektheit nachzuvollziehen und eine theoretische Grundlage zu konstruieren.

2.2 Entwicklung adaptiver Algorithmen in funkbasierten Lokalisierungssystemen

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
Dr.-Ing. Christopher Mutschler
Laufzeit: 15.05.2016 – 31.03.2017
Förderer:
Fraunhofer Institut für Integrierte Schaltungen

Ziel dieses Projektes ist die Entwicklung adaptiver Algorithmen für den Einsatz in Funklokalisierungssystemen. Im Rahmen dieses Projekts werden drei wesentliche Themen bearbeitet:

Automatisierte Konfiguration der Ereignis-Detektoren. In vorangegangenen Forschungsprojekten wurden die Grundlagen zur Analyse verrauschter Sensordatenströme gelegt. Allerdings bestand hierbei noch das Problem, dass Ereignis-Detektoren aufwändig und genau parametrisiert werden müssen, zufriedenstellende Ergebnisse zu produzieren. Dieses Arbeitspaket betrachtet Möglichkeiten einer automatisierten Konfiguration der Ereignis-Detektoren auf Basis vorhandener Ereignis- und Sensordatenströme.

In 2016 wurden erste Konzepte untersucht, um aus einer Vielzahl vorhandener Spieldaten die optimale Konfiguration der Ereignis-Detektoren zu bestimmen. Dabei wurden in einer Fußballanwendungen Spiele und Spielszenen durch Sportwissenschaftler manuell annotiert (z.B. Spieler A tritt Ball mit linkem Fuß zum Zeitpunkt t). Diese manuell annotierten Spielszenen sollen später zur Optimierung der Parameter in der Hierarchie von Ereignisdetektoren herangezogen werden.

Evaluierung von Methoden und Techniken des maschinellen Lernens für Anwendungen zur Lokalisierung. In vorhergehenden Forschungsprojekten wurden bereits erste Algorithmen des maschinellen Lernens im Kontext funkbasierter Lokalisierungssysteme entwickelt (z.B. evolutionäre Algorithmen zur Bestimmung von Antennenpositionen und -ausrichtungen). Im Rahmen dieses Arbeitspakets werden weitere Ansätze untersucht, um Lokalisierungssysteme durch derartige Methoden zu unterstützen.

Im Jahr 2016 wurden erste Ansätze evaluiert, um Teile der Positionsrechnung laufzeitbasierter Funklokalisierungssysteme durch Methoden des maschinellen Lernens zu ersetzen. Bislang werden die Rohdaten solcher Systeme durch eine Signalverarbeitungskette (Analog-Digital-Wandlung, Ankunftszeitbestimmung, Kalman-Filterung, Bewegungsanalyse) zu einer Position verrechnet. Dies erfordert einen vergleichsweise hohen Installations- und Konfigurationsaufwand für die Inbetriebnahme eines Lokalisierungssystems in der Zielumgebung und für die Zielanwendung.

Evaluierung bildgebender Verfahren zur Unterstützung funkbasierter Lokalisierungssysteme. Funkbasierte Lokalisierungssysteme können ihre Stärken gegenüber Kamera-basierten Lokalisierungssystemen immer dann ausspielen, wenn es zu Verdeckungen von Objekten kommen kann. Im Gegenzug haben Funkbasierte Systeme Probleme mit metallischen Aufbauten/Oberflächen, da die Funkwellen an metallischen Oberflächen reflektiert werden und damit über mehrere Pfade an den Empfangsantennen empfangen werden. In diesem Arbeitspaket sollen Algorithmen für eine Bild-basierte Ortungskomponente entwickelt werden, um Funk-Lokalisierungssysteme bei der Positionsrechnung zu unterstützen.

In 2016 wurde damit begonnen zwei unterschiedliche Systeme zu entwickeln: CNN-Lok, ein System zur kamerabasierten Eigenlokalisierung von Objekten (sog. inside-out tracking), sowie InfraLok ein System zur Lokalisierung mir Kameras in der Infrastruktur (sog. outside-in tracking) auf Infrarot-Basis. CNNLok nutzt ein Convolutional Neural Network, trainiert auf einer Vielzahl von in der Zielumgebung erstellter Kamerabilder. Das Netzwerk liefert anschließend zu einem aktuellen Kamerabild die Position der Kamera (bzw. des Objektes) im Raum. InfraLok detektiert Infrarot-LEDs über ein Multi-Kamerasystem und ermittelt deren Position im Raum.

2.3 Automatische Erkennung von Wettlaufsituationen

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
Michael Baer
Beginn: 01.01.2016

Große Softwareprojekte, an denen hunderte Entwickler arbeiten, sind schwer zu überblicken und enthalten viele Fehler. Zum Testen solcher Software haben sich automatisierte Tests bewährt, die Teilbereiche der Software (Unit-Tests) oder die komplette Software (Systemtest) testen und Fehler reproduzierbar erkennen können. Dieses Vorgehen funktioniert gut bei sequentiellen Programmen, die ein deterministisches Verhalten aufweisen. Moderne Software enthält jedoch vermehrt Parallelität. Durch diese Nebenläufigkeit treten etliche neue, teils schwer zu lokalisierende, Fehler auf, die nicht mehr durch herkömmliche Testverfahren sicher detektiert werden können. Dazu gehören vor allem Verklemmungen und gleichzeitige Zugriffe auf die selbe Speicherstelle. Ob ein solcher Fehler auftritt, hängt von dem konkreten Ausführungsplan der Aktivitätsfäden ab. Dieser unterscheidet sich jedoch bei jeder Ausführung und ist auch stark von dem darunterliegenden System abhängig. Somit treten entsprechende Fehler normalerweise nicht bei jedem Testlauf auf, je nach Testsystem sogar niemals. Aus diesem Grund sind die herkömmlichen Testverfahren nicht mehr ausreichend für moderne, nebenläufige Software.

In dem Projekt AuDeRace werden Verfahren entwickelt, die Nebenläufigkeitsfehler reproduzierbar, effizient und mit möglichst geringen Aufwand für Entwickler erkennen können. Unter anderem wird eine Technik entwickelt, die es ermöglicht, Tests um einen Ablaufplan zu erweitern, durch den die Ausführung weiterhin deterministisch bleibt. Ein weiteres generelles Problem an Tests bleibt dabei aber bestehen: Der Entwickler muss sich zunächst Testfälle überlegen und diese anschließend implementieren. Insbesondere im Kontext von Parallelität ist es aber extrem schwer, sich das Verhalten von Programmen vorzustellen und problematische Stellen zu identifizieren. Deshalb sollen kritische Stellen automatisiert erkannt werden, bevor überhaupt entsprechende Tests geschrieben wurde. Es existieren bereits Ansätze, potentiell kritische Stellen zu detektieren, allerdings werden dabei entweder sehr viele fälschliche Warnungen generiert oder die Analyse ist immens zeitintensiv. Ziel dieses Projektes ist es, durch eine geeignete Kombination aus statischer und dynamischer Analyse, die Anzahl an falschen Erkennungen zu reduzieren, bzw. langsame Analysen zu beschleunigen, sodass die Verfahren nicht nur in kleinen Testbeispielen funktionieren, sondern auch komplexe Softwareprojekte effizient analysieren können.

Im Jahr 2016 wurden bestehende Ansätze und Programme auf ihre Tauglichkeit untersucht. Dabei wurde ein vielversprechendes Verfahren ausgemacht, das mithilfe von Mo-

del Checking und vorgegebenen Bedingungen Ablaufpläne konstruiert, die ungewolltes Verhalten erzeugen. Allerdings zeigten die Ergebnisse ein deutliches Problem hinsichtlich eines produktiven Einsatzes, da in sinnvoller Zeit nur sehr kleine Programme analysiert werden konnten. Daher beschäftigte sich das Projekt damit, die Programme um möglichst viele Anweisungen zu kürzen, die nichts mit den gesuchten Wettlaufsituationen zu tun haben. Dadurch sollen spätere Analysen beschleunigt werden.

Im Jahr 2017 wurden die Untersuchungen zur automatischen Reduktion von Programmen weitergeführt. Außerdem wurde erforscht, ob sich die existierende Technik des Mutationstestens auch auf nebenläufige Programme erweitern lässt. Die Ergebnisse zeigen, dass es durchaus möglich ist, Tests für eine parallele Software qualitativ zu bewerten. Allerdings muss teilweise auf Heuristiken zurückgegriffen werden, um auch größere Projekte in vertretbarer Zeit bewerten zu können.

Das Projekt stellt einen Beitrag des Lehrstuhls Informatik 2 zum IZ ESI (Embedded Systems Initiative, http://www.esi.fau.de/) dar.

2.4 Analyse von Code-Repositories

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
Patrick Kreutzer
Georg Dotzler
Marius Kamp
Beginn: 01.01.2010

Bei der Weiterentwicklung von Software führen die Entwickler oftmals sich wiederholende, ähnliche Änderungen durch. Dazu gehört beispielsweise die Anpassung von Programmen an eine veränderte Bibliotheksschnittstelle, die Behebung von Fehlern in funktional ähnlichen Komponenten sowie die Parallelisierung von sequentiellen Programmteilen. Wenn jeder Entwickler die nötigen Änderungen selbst erarbeiten muss, führt dies leicht zu fehlerhaften Programmen, beispielsweise weil weitere zu ändernde Stellen übersehen werden. Wünschenswert wäre stattdessen ein automatisiertes Verfahren, das ähnliche Änderungen erkennt und mit dieser Wissensbasis Software-Entwickler bei weiteren Änderungen unterstützt.

Änderungsextraktion

In 2017 entwickelten wir ein neues Vorschlagssystem mit Namen ARES (Accurate RECommendation System). Verglichen mit bisherigen Ansätzen erzeugt es genauere Vorschläge, da seine Algorithmen Code-Verschiebungen während der Muster- und Vor-

schlagserzeugung berücksichtigen. Der Ansatz basiert darauf, dass zwei Versionen eines Programms miteinander verglichen werden. Das Werkzeug extrahiert dabei automatisch, welche Änderungen sich zwischen den beiden Versionen ergeben haben, und leitet daraus generalisierte Muster aus zu ersetzenden Code-Sequenzen ab. Diese Muster können anschließend von ARES dazu verwendet werden, analoge Änderungen für den Quellcode anderer Programme automatisch vorzuschlagen.

Zur Extraktion der Änderungen verwenden wir ein baumbasiertes Verfahren. Im Jahr 2016 wurde ein neuer Algorithmus (MTDIFF) für solche baumbasierten Verfahren entwickelt und gut sichtbar publiziert, der die Genauigkeit der Änderungsbestimmung verbessert.

Symbolische Ausführung von Code-Fragmenten

Im Jahr 2014 wurde ein neues Verfahren zur symbolischen Code-Ausführung namens SYFEX entwickelt, welches die Ähnlichkeit des Verhaltens zweier Code-Teilstücke bestimmt. Mit diesem Verfahren soll eine Steigerung der Qualität der Verbesserungsvorschläge erreicht werden. Abhängig von der Anzahl und Generalität der Muster in der Datenbank kann SIFE ohne das neue Verfahren unpassende Vorschläge liefern. Um dem Entwickler nur die passenden Vorschläge anzuzeigen, wird das semantische Verhalten des Vorschlags mit dem semantischen Verhalten des Musters aus der Datenbank verglichen. Weichen beide zu sehr voneinander ab, wird der Vorschlag aus der Ergebnismenge entfernt. Die Besonderheit von SYFEX besteht darin, dass es auf herausgelöste Code-Teilstücke anwendbar ist und keine menschliche Vorkonfiguration benötigt.

SYFEX wurde im Jahr 2015 verfeinert und auf Code-Teilstücke aus Archiven von verschiedenen Software-Projekten angewendet. Der Schwerpunkt im Jahr 2016 lag auf einer Untersuchung, inwieweit SYFEX zum semantischen Vergleich von Abgaben eines Programmierwettbewerbs geeignet ist. Im Jahr 2017 wurde SYFEX optimiert. Des Weiteren wurde mit der Erstellung eines Datensatzes semantisch ähnlicher Methoden aus quelloffenen Software-Archiven begonnen.

Detektion von semantisch ähnlichen Code-Fragmenten

SYFEX erlaubt es, die semantische Ähnlichkeit zweier Code-Fragmente zu bestimmen. So ist es damit prinzipiell möglich, Paare oder Gruppen von semantisch ähnlichen Code-Fragmenten (semantische Klone) zu identifizieren. Auf Grund des hohen Laufzeitaufwands verbietet sich der Einsatz von SYFEX – wie auch von anderen Werkzeugen dieser Art – allerdings, um in größeren Code-Projekten nach semantisch ähnlichen Code-Fragmenten zu suchen. Im Jahr 2016 wurde deshalb mit der Entwicklung eines Verfahrens begonnen, mit dessen Hilfe die Detektion semantisch ähnlicher Code-Fragmente beschleunigt werden kann. Grundlage dieses Verfahrens ist eine Reihe von sog. Basiskomparatoren, die zwei Code-Fragmente jeweils hinsichtlich eines Kriteriums (beispielsweise die Anzahl bestimmter Kontrollstrukturen oder die Beschaffenheit der

Kontrollflussgraphen) miteinander vergleichen und dabei möglichst geringen Laufzeitaufwand haben. Diese Basiskomparatoren können anschließend zu einer Hierarchie von Verfahren verknüpft werden. Um damit die semantische Ähnlichkeit zweier Fragmente möglichst genau bestimmen zu können, wird mit Hilfe der Genetischen Programmierung nach Hierarchien gesucht, die die von SYFEX für eine Reihe von Code-Paaren berechneten Ähnlichkeitswerte möglichst gut approximieren. Im Rahmen einer ersten Untersuchung hat sich gezeigt, dass sich das implementierte Verfahren tatsächlich für die Bestimmung von semantisch ähnlichen Code-Paaren eignet.

Die Implementierung dieses Verfahrens wurde im Jahr 2017 weiter verbessert. Zudem spielte die tiefergehende Evaluation des Verfahrens auf Basis von Methodenpaaren aus Software-Archiven sowie von Abgaben für Programmieraufgaben eine wichtige Rolle.

Semantische Code-Suche

Häufig steht die bei der Software-Entwicklung zu implementierende Funktionalität bereits in ähnlicher Form als Teil von Programmbibliotheken zur Verfügung. In vielen Fällen ist es ratsam, diese bereits vorhandene Realisierung zu verwenden statt die Funktionalität erneut zu implementieren, beispielsweise um den Aufwand für das Entwickeln und Testen des Codes zu reduzieren.

Voraussetzung für die Wiederverwendung einer für den Anwendugszweck geeigneten Implementierung ist, dass Entwickler diese überhaupt finden können. Zu diesem Zweck werden bereits heute regelmäßig Code-Suchmaschinen verwendet. Etablierte Verfahren stützen sich dabei insbesondere auf syntaktische Merkmale, d.h. der Nutzer gibt beispielsweise eine Reihe von Schlüsselwörtern oder Variablen- und Methodennamen an, nach denen die Suchmaschine suchen soll. Bei diesen Verfahren bleibt die Semantik des zu suchenden Codes unberücksichtigt. Dies führt in der Regel dazu, dass relevante, aber syntaktisch verschiedene Implementierungen nicht gefunden werden (”false negatives”) oder dass syntaktisch ähnliche, aber semantisch irrelevante Ergebnisse präsentiert werden (”false positives”). Die Suche nach Code-Fragmenten auf Basis ihrer Semantik ist Gegenstand aktueller Forschung.

Im Jahr 2017 wurde am Lehrstuhl mit der Entwicklung eines neuen Verfahrens zur semantischen Code-Suche begonnen. Der Nutzer spezifiziert dabei die gesuchte Funktionalität in Form von Eingabe-Ausgabe-Beispielen. Mit Hilfe eines aus der Literatur stammenden Verfahrens zur Funktionssynthese wird eine Methode erzeugt, die das durch die Beispiele beschriebene Verhalten möglichst genau realisiert. Diese synthetisierte Methode wird dann mit Hilfe des im Rahmen dieses Forschungsprojekts entwickelten Verfahrens zur Detektion von semantisch ähnlichen Code-Fragmenten mit den Methodenimplementierungen vorgegebener Programmbibliotheken verglichen, um ähnliche Implementierungen zu finden, die dem Nutzer als Ergebnis der Suche präsentiert werden. Eine erste Evaluation der prototypischen Implementierung zeigt die Umsetzbarkeit und Verwendbarkeit des Verfahrens.

Cluster-Bildung von ähnlichen Code-Änderungen

Voraussetzung für die Erzeugung generalisierter Änderungsmuster ist es, die Menge aller aus einem Quelltext-Archiv extrahierten Code-Änderungen in Teilmengen zueinander ähnlicher Änderungen aufzuteilen. Im Jahr 2015 wurde diese Erkennung ähnlicher Änderungen im Rahmen eines neuen Werkzeugs C3 verbessert. In einem ersten Schritt wurden verschiedene Metriken für den paarweisen Ähnlichkeitsvergleich der extrahierten Code-Änderungen implementiert und evaluiert. Darauf aufbauend wurden aus der Literatur bekannte Clustering-Algorithmen evaluiert und neue Heuristiken zur automatisierten Bestimmung der jeweiligen Parameter implementiert, um das bisherige naive Verfahren zur Identifizierung ähnlicher Änderungen zu ersetzen. Mit den im Rahmen von C3 implementierten Verfahren konnte im Vergleich zum bisherigen Ansatz eine deutliche Verbesserung erzielt werden. So können mit den neuen Verfahren mehr Gruppen ähnlicher Änderungen identifiziert werden, die sich für die Weiterverarbeitung im Rahmen von SIFE zur Generierung von Vorschlägen eignen.

Die zweite Verbesserung zielt darauf ab, die erhaltenen Gruppen ähnlicher Änderungen zusätzlich automatisiert zu verfeinern. Zu diesem Zweck wurden verschiedene Verfahren aus dem Umfeld des maschinellen Lernens zur Ausreißererkennung untersucht, um Änderungen, die fälschlicherweise einer Gruppe zugeordnet wurden, wieder zu entfernen.

Im Jahr 2016 wurde C3 um eine weitere Metrik zum Vergleich zweier Code-Änderungen erweitert, die im Wesentlichen den textuellen Unterschied zwischen den Änderungen (wie er beispielsweise von dem Unix-Werkzeug 'diff' erzeugt wird) bewertet. Des Weiteren wurde das in C3 implementierte Verfahren im Rahmen eines Konferenzbeitrags veröffentlicht. In diesem Zusammenhang wurde auch der zur Evaluation des Verfahrens erzeugte Datensatz von Gruppen ähnlicher Änderungen unter einer Open-Source-Lizenz veröffentlicht, siehe https://github.com/FAU-Inf2/cthree . Dieser kann zukünftigen Arbeiten als Referenz oder Eingabe dienen. Außerdem wurden prototypisch Verfahren implementiert, mit denen die Ähnlichkeitsberechnung und das Clustering in C3 inkrementell erfolgen können. Diese erlauben es, dass bei neuen Änderungen, die zu einem Software-Archiv hinzugefügt werden, die zuvor bereits berechneten Ergebnisse weiterverwendet werden können und nur ein Teil der Arbeit wiederholt werden muss.

2.5 Design for Diagnosability

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
Florian Lautenschlager
Dr.-Ing. Josef Adersberger

Laufzeit: 15.05.2013 – 31.07.2016
Laufzeitverlängerung bis: 30.09.2018
Förderer:
Bayerisches Staatsministerium für Wirtschaft und Medien, Energie und Technologie (StMWIVT) (ab 10/2013)

Viele Software-Systeme verhalten sich während der Testphase oder sogar im Regelbetrieb im negativen Sinne auffällig. Die Diagnose und die Therapie solcher Laufzeitanomalien ist oft langwierig und aufwändig bis hin zu unmöglich. Mögliche Folgen bei der Verwendung des Software-Systems sind lange Antwortzeiten, nicht erklärbares Verhalten oder auch Abstürze. Je länger die Folgen unbehandelt bleiben, desto höher ist der entstehende wirtschaftliche Schaden.

"Design for Diagnosability"beschreibt eine Werkzeugkette mit Modellierungssprachen, Bausteinen und Werkzeugen, mit denen die Diagnosefähigkeit von Software-Systemen gesteigert wird. Mit dieser Werkzeugkette werden Laufzeitanomalien schneller erkannt und behoben – idealerweise noch während der Entwicklung des Software-Systems. Unser Kooperationspartner QAware GmbH bringt ein Software EKG ein, mit dem die Exploration von Laufzeit-Metriken aus Software-Systemen, visualisiert als Zeitreihen, möglich ist.

Das Forschungsprojekt Design for Diagnosability erweitert das Umfeld dieses bestehenden Software-EKG. Die Software-Blackbox misst minimal-invasiv technische und fachliche Laufzeitdaten des Systems. Die Speicherung der erfassten Daten erfolgt in Form von Zeitreihen in einer neu entwickelten Zeitreihendatenbank Chronix. Chronix ist darauf ausgelegt, eine Vielzahl an Zeitreihen äußerst effizient hinsichtlich Speicherplatzbedarf und Zugriffszeiten zu speichern. Chronix ist ein Open Source Projekt (www.chronix.io) und kann frei benutzt werden. Die Zeitreihen werden mit der Time-Series-API analysiert, z.B. mittels einer automatisierten Strategie zur Erkennung von Ausreißern. Die Time-Series-API bietet Grundbausteine, um weitere Strategien zur Identifikation von Laufzeitanomalien in Zeitreihen umzusetzen.

Die aufgeführten Werkzeuge werden in Kombination mit dem bestehenden Software-EKG zum Dynamic Analysis Workbench ausgebaut, um eine zeitnahe Diagnose und Behebung von Laufzeitanomalien zu ermöglichen. Hierzu sind Diagnosepläne vorgesehen, die einen Software-Entwickler unterstützen, eine Laufzeitanomalie schneller und zuverlässiger einzugrenzen und zu erkennen. Das Ziel der Werkzeugkette ist die Qualität von Software-Systemen zu erhöhen, insbesondere hinsichtlich der Kennzahlen Mean-Time-To-Repair sowie Mean-Time-Between-Defects.

Vor dem erfolgreichen Projektabschluss im Juli 2016 konnten noch eine Reihe wesentlicher Beiträge geleistet werden:

- Wir haben Chronix und ein Framework zur verteilten Berechnung gekoppelt. Dadurch skaliert die Anomalieanalyse jetzt auf riesige Mengen an Zeitreihendaten.

- Wir haben Chronix fortentwickelt und weitere Komponenten ergänzt, wie z. B. ein noch effizienteres Speichermodell, mehrere Adapter für diverse Zeitreihendatenbanken, weitere server-seitige Analysefunktionen und neue Zeitreihentypen.
- Wir haben unseren Benchmark für Zeitreihendatenbanken veröffentlicht.
- Wir haben zur Analyse von Anomalien einen Ansatz entwickelt, der Aufrufe ausgehend von der Anwendung bis hinein auf die Betriebssystemebene nachvollzieht.

Obwohl die Förderung im Jahr 2016 auslief, haben wir im Jahr 2017 noch weitere Beiträge geleistet:

- Wir haben Chronix auf der USENIX Conference on File and Storage Technologies (FAST) im Februar 2017 in Santa Clara, CA, vorgestellt.
- Wir haben Chronix mit Schnittstellen ausgestattet, um in der Industrie verwendete Zeitreihendatenbanken anzubinden.
- Wir haben einen Ansatz entwickelt, der für eine gegebene Analyse (Funktion und zu analysierende Zeitreihen) die ideale Cluster-Konfiguration (bzgl. Verarbeitungszeit und Kosten) bestimmt.
- Wir haben Spark, ein Rahmenwerk zur verteilten Verarbeitung von Massendaten so erweitert, dass die GPU bei der verteilten Analyse von Zeitreihen genutzt werden kann. Ergebnisse haben wir auf der Apache Big Data Konferenz im Mai 2017 in Miami, Florida, vorgestellt.

2.6 International Collegiate Programming Contest an der FAU

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
Daniela Novac
Michael Baer
Beginn: 01.11.2002

Die Association for Computing Machinery (ACM) richtet seit Jahrzehnten den International Collegiate Programming Contest (ICPC) aus. Dabei sollen Teams aus je drei Studenten in fünf Stunden neun bis elf Programmieraufgaben lösen. Als Erschwernis kommt hinzu, dass nur ein Computer pro Gruppe zur Verfügung steht.

Die Aufgaben erfordern solide Kenntnisse von Algorithmen aus allen Gebieten der Informatik und Mathematik, wie z.B. Graphen, Kombinatorik, Zeichenketten, Algebra und Geometrie.

Der ICPC wird jedes Jahr in drei Stufen ausgetragen. Zuerst werden innerhalb der Universitäten in lokalen Ausscheidungen die maximal drei Teams bestimmt, die dann zu den regionalen Wettbewerben entsandt werden. Erlangen liegt seit dem Jahr 2009 im Einzugsbereich des Northwestern European Regional Contest (NWERC), an dem u.a. auch Teams aus Großbritannien, den Benelux-Staaten und Skandinavien teilnehmen. Die Sieger aller regionalen Wettbewerbe der Welt (und einige Zweitplatzierte) erreichen die World Finals, die im Frühjahr des jeweils darauffolgenden Jahres (2018 in Beijing, China) stattfinden

Im Jahr 2017 fanden zwei lokale Wettbewerbe an der FAU statt. Im Wintersemester wurde ein Mannschaftswettbewerb ausgetragen mit dem Ziel, neue Studierende für die Wettbewerbe zu begeistern - es meldeten sich 29 Erlanger Teams an. Jedes Team bestand aus maximal drei Studenten. Außerdem nahmen noch 35 Teams von anderen europäischen Universitäten teil. Im Sommersemester fand vor dem Wettbewerb zum wiederholten Mal das Hauptseminar "Hallo Welt! - Programmieren für Fortgeschritteneßtatt, um Studierende verschiedener Fachrichtungen in Algorithmen und Wettbewerbsaufgaben zu schulen. Der Wettbewerb im Sommersemester diente danach der Auswahl der studentischen Vertreter der FAU für den NWERC 2017 in Bath (GB). Insgesamt nahmen an dem deutschlandweit organisierten Ausscheidungskampf eine Rekordzahl von 36 Teams der FAU mit Studenten verschiedenster Fachrichtungen teil.

Aus den besten Teams und unter Berücksichtigung der Altersbegrenzung, wurden neun Studenten ausgewählt, die für den NWERC Dreierteams bildeten. Diese erreichten dann beim NWERC in Bath die Plätze 14, 29 und 57 von insgesamt 120 teilnehmenden Teams und somit eine solide Platzierung, knapp an einer Medaille vorbei.

2.7 Software-Wasserzeichen

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
Daniela Novac
Beginn: 01.01.2016

Unter Software-Wasserzeichnen versteht man das Verstecken von ausgewählten Merkmalen in Programme, um sie entweder zu identifizieren oder zu authentifizieren. Das ist nützlich im Rahmen der Bekämpfung von Software-Piraterie, aber auch um die richtige Nutzung von Open-Source Projekten (wie zum Beispiel unter der GNU

Lizenz stehende Projekte) zu überprüfen. Die bisherigen Ansätze gehen davon aus, dass das Wasserzeichen bei der Entwicklung des Codes hinzugefügt wird und benötigen somit das Verständnis und den Beitrag der Programmierer für den Einbettungsprozess. Ziel unseres Forschungsprojekts ist es, ein Wasserzeichen-Framework zu entwickeln, dessen Verfahren automatisiert beim übersetzen des Programms Wasserzeichen hinzufügen, auch für existierende Programme. Als ersten Ansatz untersuchen wir eine Wasserzeichenmethode, die auf einer symbolischen Ausführung und anschließender Funktionsynthese basiert.

Im Jahr 2017 wurden dazu verschiedene Funktionssynthese-Methoden untersucht, um zu ermitteln, welche sich für unsere Ansatz eignen würden.

2.8 Parallele Code-Analyse auf einer GPU

Projektleitung:
Prof. Dr. Michael Philippsen
Beteiligte:
Thorsten Blaß
Beginn: 01.07.2013

Im Übersetzerbau (und auch an anderen Stellen) gibt es Analyseverfahren, bei denen Informationen solange durch einen Graph propagiert und dabei verändert werden, bis sich das Analyseergebnis als Fixpunkt einstellt.

Ziel dieses Projektes ist ein Programmrahmen, in dem verschiedene derartige Verfahren parallel und dadurch schneller auf der Graphikkarte ablaufen können.

Graphen sind eine fundamentale Datenstruktur zur Repräsentation der Beziehungen zwischen Daten (z.B. soziale Netzwerke, Analyse der Verlinkung von Webseiten). Zu analysierende Graphen können Millionen/Milliarden von Knoten/Kanten enthalten. GPUs können Graphen mit mehreren 1000 Threads parallel verarbeiten. Graph-Analysen arbeiten nach dem Bulk Synchrones Parallel (BSP) Model. Eine Graph-Analyse zerfällt in drei, strikt getrennte, Phasen: Rechnen, Kommunikation und Synchronisation. Letztere beiden setzen Interaktion mit dem Host (CPU) voraus, welche sich negativ auf die Laufzeit auswirken. Unser GPU-basierter Übersetzer arbeitet ebenfalls nach dem BSP Model: Ein Entwickler modifiziert Code, dieser wird in einen (Kontrollfluss-) Graphen transformiert und auf der GPU analysiert. Jede Code-Veränderung löst diesen Zyklus aus. Der Graph muss also sehr schnell aufgebaut und auf die GPU transferiert werden.

Publikationen im Bereich der Graph-Analysen beschränken sich darauf, das Rechnen zu beschleunigen und auch nur dies zu vermessen. Die Ende-zu-Ende Zeit der Ausführung eines Programmes wird nicht berücksichtigt. Der Einfluss der Kommunikation und

Synchronisation wird außer Acht gelassen, hat aber einen entscheidenden Einfluss auf die Laufzeit.

Für unseren GPU gestützten Übersetzer, betrachten wir alle drei Phasen des BSP-Models. 2017 veröffentlichten wir ein Papier, das die Synchronisation erheblich beschleunigt. Ferner fokussierten wir uns auf die Kommunikation. In diesem Kontext bedeutet Kommunikation das Kopieren des Graphen auf die GPU (und zurück). Während diese Zeit stark von der Datenstruktur des Graphen abhängt, beeinflusst sie auch die Dauer der Rechenphase.

Weil bislang noch keine Untersuchung in der Literatur vorhanden ist, die die Datenstrukturen systematisch hinsichtlich ihrer Effizienz im Kontext von GPUs untersucht, implementierten wir einerseits mehrere Benchmarks, die unterschiedliche Eigenschaften von Graphen abfragen (Zugriff nur auf Vorgänger/Nachfolger, zufälliger Zugriff auf Knoten). Andererseits implementierten wir 8 Datenstrukturen zur Darstellung von Graphen auf der GPU. Die daraus resultierenden Kombinationen wurden mit, strukturell verschiedenen, Eingabegraphen vermessen. Die Ergebnisse sollen Entwickler bei der passenden Wahl der Datenstruktur für ihr GPU-Problem unterstützen.

3 Publikationen 2017

- Feigl, T., Mutschler, C., Philippsen, M., & Kõre, E. (2017). Acoustical manipulation for redirected walking. In Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology (VRST '17) (pp. 45:1-45:2). Gothenburg, SE: New York: ACM.

- Blaß, T., Philippsen, M., & Veldema, R. (2017). Efficient Inspected Critical Sections in data-parallel GPU codes. In Rauchwerger, Lawrence (Eds.), Proceedings of the 30th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2017) (pp. -). College Station, TX, US: Berlin: Springer-Verlag Berlin Heidelberg.

- Tausch, N. (2017). Eine domänenspezifische Sprache zur Analyse von Software-Verfolgbarkeitsinformationen (Dissertation).

- Polzer, T., & Adersberger, J. (2017, May). Leveraging the GPU on Spark. Paper presentation at Apache: Big Data North America 2017, Miami, FL, US.

- Roth, D., Kleinbeck, C., Feigl, T., Mutschler, C., & Latoschik, M.E. (2017). Social Augmentations in Multi-User Virtual Reality: A Virtual Museum Experience. In Proceedings of the 2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct) (pp. 42-43). Nantes, FR: IEEE.

– Hammer, J., Gaukler, M., Kanzler, P., Hörauf, P., & Novac, D. (2017). FAU FabLab: A Fabrication Laboratory for Scientists, Students, Entrepreneurs and the Curious.

– Oster, N., Kamp, M., & Philippsen, M. (2017). AuDoscore: Automatic Grading of Java or Scala Homework. In Sven Strickroth Oliver Müller Michael Striewe (Eds.), Proceedings of the Third Workshop "Automatische Bewertung von Programmieraufgaben" (ABP 2017). Potsdam, DE: Online Proceedings for Scientific Conferences and Workshops (CEUR-WS).

– Krainz, J., & Philippsen, M. (2017). Diff Graphs for a fast Incremental Pointer Analysis. In ACM (Eds.), Proceedings of the 12th Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems (ICOOOLPS'17). Barcelona, ES: ACM DL.

– Ellner, R. (2017). Modellierung und effiziente Ausführung von Softwareentwicklungsprozessen (Dissertation).

– Dotzler, G., Kamp, M., Kreutzer, P., & Philippsen, M. (2017). More Accurate Recommendations for Method-Level Changes. In Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE2017) (pp. 798-808). Paderborn, DE: New York, NY, USA: ACM DL.

– Lautenschlager, F., Philippsen, M., Kumlehn, A., & Adersberger, J. (2017). Chronix: Long Term Storage and Retrieval Technology for Anomaly Detection in Operational Data. In USENIX Association (Eds.), Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST 17) (pp. 229-242). Santa Clara, CA, US.

– Tausch, N. (2017). Eine domänenspezifische Sprache zur Analyse von Software-Verfolgbarkeitsinformationen (Dissertation).

– Roth, D., Kleinbeck, C., Feigl, T., Mutschler, C., & Latoschik, M.-E. (2017, October). Social Augmentations in Multi-User Virtual Reality: A Virtual Museum Experience. Poster presentation at 2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct), Nantes, FR.

4 Studien- und Abschlussarbeiten

– Master Thesis: Distributed GPU-accelerated time series analysis with Apache Spark and Chronix. Bearbeiter: Tobias Polzer (beendet am 16.01.2017); Betreuer: Dr.-Ing. Josef Adersberger; Prof. Dr. Michael Philippsen

– Master Thesis: Ermittlung des Skalierungsbedarfs bei der Verarbeitung und Analyse von Zeitreihendaten. Bearbeiter: Sebastian Haubner (beendet am 23.01.2017); Betreuer: Florian Lautenschlager, M. Sc.; Prof. Dr. Michael Philippsen

– Master Thesis: Werkzeug zur automatischen Ergänzung bestehender Testsuiten zur maschinellen Bewertung studentischer Abgaben. Bearbeiter: Guillermo Janner (beendet am 15.02.2017); Betreuer: Marius Kamp, M. Sc.; Dr.-Ing. Norbert Oster, Akad. ORat; Prof. Dr. Michael Philippsen

– Studienarbeit: Beschreibung von Varianten im Systemtest mit Hilfe von domain specific languages (DSLs). Bearbeiter: Max Draf (beendet am 10.03.2017); Betreuer: Dr.-Ing. Martin Jung; Hon.-Prof. Dr.-Ing. Detlef Kips

– Hausarbeit: Konzeption und Umsetzung eines Werkzeugs für den Mutationstest mit spezialisierten Operatoren für Nebenläufigkeitsfehler. Bearbeiter: Benjamin Bösl (beendet am 27.03.2017); Betreuer: Michael Baer, M. Sc.; Dr.-Ing. Norbert Oster, Akad. ORat; Prof. Dr. Michael Philippsen

– Master Thesis: Analyse und Bewertung von Software-Architekturen zwecks Unterstützung der präventiven Wartung durch Restrukturierung. Bearbeiter: Andreas Grünwald (beendet am 4.4.2017); Betreuer: Dr.-Ing. Norbert Oster, Akad. ORat; Patrick Kreutzer, M. Sc.; Marius Kamp, M. Sc.; Prof. Dr. Michael Philippsen

– Bachelor Thesis: Exploiting Reinforcement Learning for complex trajectory planning for mobile robots. Bearbeiter: Leonid Butyrev (beendet am 01.06.2017); Betreuer: Dr.-Ing. Thorsten Edelhäußer; Prof. Dr. Michael Philippsen

– Bachelor Thesis: Entwicklung eines Werkzeugs zur Bestimmung der Fehleraufdeckungsgüte von Testfällen beim Mutationstest mit Operatoren für nebenläufige Programme. Bearbeiter: Stefan Kraus (beendet am 12.06.2017); Betreuer: Dr.-Ing. Norbert Oster, Akad. ORat; Michael Baer, M. Sc.; Prof. Dr. Michael Philippsen

– Bachelor Thesis: Modellieren und Generieren von Aktionsplänen für autonome mobile Systeme. Bearbeiter: Robert Klinger (beendet am 02.10.2017); Betreuer: Hon.-Prof. Dr.-Ing. Detlef Kips; Dr.-Ing. Martin Jung

– Bachelor Thesis: Semantische Code-Suche mittels Funktionssynthese. Bearbeiter: Christian Spangler (beendet am 21.12.2017); Betreuer: Patrick Kreutzer, M. Sc.; Marius Kamp, M. Sc.; Prof. Dr. Michael Philippsen