Annual Report of the Chair of Computer Science 2 (Programming Systems)

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

Ordinarius:
Prof. Dr. Michael Philippsen
Honorary Professor:
Hon.-Prof. Dr.-Ing. Bernd Hindel
Hon.-Prof. Dr.-Ing. Detlef Kips
Professor Emeritus:
Prof. em. Dr. Hans Jürgen Schneider
Secretary:
Margit Zenk
Scientific Staff:
Michael Baer, M. Sc.
Dipl.-Inf. Thorsten Blaß
Tobias Feigl, M. Sc. (since October 01, 2017)
Florian Jung, M. Sc. (since December 01, 2017)
Marius Kamp, M. Sc.
Dipl.-Math. Jakob Krainz (until June 30, 2017)
Patrick Kreutzer, M. Sc.
Florian Mayer, M. Sc. (since August 01, 2017)
Dr.-Ing. Christopher Mutschler
Dipl.-Inf. Daniela Novac
Dr.-Ing. Norbert Oster, Akad. ORat
IT-support:
Dipl.-Ing. (FH) Helmut Allendorf
Manfred Uebler
Guest:
Dr.-Ing. Josef Adersberger
Florian Lautenschlager, M. Sc.
External Teaching Staff:
Dr.-Ing. Klaudia Dussa-Zieger
Dr.-Ing. Martin Jung

Since 2002 Prof. Michael Philippsen is heading the Chair of Computer Science 2 which was founded in 1972. The group's main focus is on **Programming Systems for heterogeneous Multicores**.

Up to 2014/2015, we mainly took a system-level perspective and worked on the interface to both the parallel hardware and the OS, on runtime system issues, and on how to best express parallelism in program code. We strived to best harvest the potential of parallelism that is slumbering within applications and to find the most efficient ways to map it to the parallelism provided by the hardware. As such systems-oriented topics continue to be relevant, some of our projects continue to address them.

Nowadays the Programming Systems Group mainly finds answers for professional software engineers who develop industry-sized parallel programs for multicores, for distributed networks of multicores, for parallel cloud computing, and for networks of embedded systems.

1 Focus of research

A few corner stones stake out our field of **programming system research**: (a) We develop and evaluate uniform **programming models** for heterogeneous systems. To make the investment worthwhile, modules of parallel software need homogeneous interfaces to gain portability across changing and heterogeneous configurations of multicores, GPUs, accelerators, FPGAs etc. (b) There is growing commercial relevance in migrating legacy software to run on multicores. We investigate novel tools that work on **source code repositories**, analyze them, and help developers to **migrate, refactor and parallelize** software. In special cases, there are effective ways to even perform this task automatically. (c) Since parallelism and synchronization issues make parallel software more complex, software engineers need more support in their development and maintenance tasks. Only with such tool support they can avoid, detect and fix mistakes early on. And only with such tool support these tasks do no longer need the specialized expertise of low-level systems programmers. We thus develop **fast, interactive, and incremental code analyses** (that themselves may execute in parallel) that not only detect race conditions, conflicting accesses to resources, etc., but that also provide support and suggestions to developers while they are working in their codes with their IDEs. (d) During the life cycle of parallel software, the nondeterministic behavior of concurrency poses new challenges for **testing of parallel code**, for the assurance of quality and **code-authenticity**, as well as for operation and **diagnosis**. We explore, when parallel code is tested thoroughly enough, how to guard parallel code against attacks, how to generate sets of test data for parallel programs, how to systematically track down the reasons of erratic behavior of parallel codes, etc. Established techniques from sequential software

engineering cannot yet deal with the parallelism in the code that adds a new dimension to these problems.

We follow a **program-code-centric approach**, we build operational **prototypes** for the tools that we conceive, and we carry forward our principle of a thorough **qualitative and quantitative evaluation** of our ideas.

2 Research projects

2.1 Analysis of Code Repositories

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Georg Dotzler
Marius Kamp, M. Sc.
Patrick Kreutzer, M. Sc.
Start: 1.1.2010

Software developers often modify their projects in a similar or repetitive way. The reasons for these changes include the adoption of a changed interface to a library, the correction of mistakes in functionally similar components, or the parallelization of sequential parts of a program. If developers have to perform the necessary changes on their own, the modifications can easily introduce errors, for example due to a missed change location. Therefore, an automatic technique is desireable that identifies similar changes and uses this knowledge to support developers with further modifications.

Extraction of Code-Changes

In 2017, we developed a new code recommendation tool called ARES (Accurate RE-commendation System). It creates more accurate recommendation compared to previous tools as its algorithms take care of code movements during pattern and recommendation creation. The foundation of ARES lies in the comparison of two versions of the same program. It extracts the changes between the two versions and creates patterns based on the changed methods. ARES uses these patterns to suggest similar changes for the source code of different programs automatically.

The extraction of code changes is based on trees. In 2016 we developed (and visibly published) a new tree-based algorithm (MTDIFF) that improves the accuracy of the change extraction.

Symbolic Execution of Code-Fragments

In 2014 we developed a new symbolic code execution engine called SYFEX to determine the behavioral similarity of two code fragments. In this way we aim to improve the quality of the recommendations. Depending on the number and the generality of the patterns in the database, it is possible that without the new engine SIFE generates some unfitting recommendations. To present only the fitting recommendations to the developers, we compare the summary of the semantics/behavior of the recommendation with summary of the semantics/behavior of the database pattern. If both differ too severely, our tool drops the recommendation from the results. The distinctive features of SYFEX are its applicability to isolated code fragments and its automatic configuration that does not require any human interaction.

In 2015 SYFEX was refined and applied to code fragments from the repositories of different software projects. In 2016 we investigated to which extend SYFEX can be used to gauge the semantic similarity of submissions for a programming contest. In 2017 we optimized the implementation of SYFEX. We also began collecting a data set of semantically similar methods from open source repositories.

Detection of Semantically Similar Code Fragments

SYFEX computes the semantic similarity of two code fragments. Therefore, it allows to identify pairs or groups of semantically similar code fragments (semantic clones). However, the high runtime of SYFEX (and similar tools) limit their applicability to larger software projects. In 2016, we started the development of a technique to accelerate the detection of semantically similar code fragments. The technique is based on so-called base comparators that compare two code fragments using a single criterion (e.g., the number of used control structures or the structure of the control flow graph) and that have a low runtime. These base comparators can be combined to form a hierarchy of comparators. To compute the semantic similarity of two code fragments as accurately as possible, we use genetic programming to search for hierarchies that approximate the similarity values as reported by SYFEX for a number of pairs of code fragments. A prototype implementation confirmed that the method is capable of detecting pairs of semantically similar code fragments.

We further improved the implementation of this approach in 2017. Additionally, we focused on evaluating the approach with pairs of methods from software repositories and from programming exercises.

Semantic Code Search

The functionality that has to be implemented during the development of a software product is often already available as part of program libraries. It is often advisable to reuse such an implementation instead of rewriting it, for example to reduce the effort for developing and testing the code.

To reuse an implementation that fits the purpose, developers have to find it first. To this end developers already use code search engines on a regular basis. State-of-the-art search engines work on a syntactic level, i.e., the user specifies some keywords or names of variables and methods that should be searched for. However, current approaches do not consider the semantics of the code that the user seeks. As a consequence, relevant but syntactically different implementations often remain undetected (”false negatives”) or the results include syntactically similar but semantically irrelevant implementations (”false positives”). The search for code fragments on a semantic level is the subject of current research.

In 2017 we began the development of a new method for semantic code search. The user specifies the desired functionality in terms of input/output examples. A function synthesis algorithm from the literature is then used to create a method that implements the specified functionality as accurately as possible. Using our approach to detect similar code fragments, this synthesized method is then compared to the methods of program libraries to find semantically similar implementations. These implementations are then presented as search results to the user. A first evaluation of our prototypical implementation shows the feasibility and practicability of the approach.

Clustering of Similar Code-Changes

To create generalized change patterns, it is necessary that the set of extracted code changes is split into subsets of changes that are similar to each other. In 2015 this detection of similar code changes was improved and resulted in a new tool, called C3. We developed and evaluated different metrics for a pairwise similarity comparison of the extracted code changes. Subsequently, we evaluated different clustering algorithms known from the literature and implemented new heuristics to automatically choose the respective parameters to replace the previous naive approach for the detection of similar code changes. This clearly improved the results compared to the previous approach, i.e., C3’s new techniques detect more groups of similar changes that can be processed by SIFE to generate recommendations.

The aim of the second improvement is to automatically refine the resulting groups of similar code changes. For this purpose we evaluated several machine learning algorithms for outlier detection to remove those code changes that have been spuriously assigned to a group.

In 2016 we implemented a new similarity metric for the comparison of two code changes that essentially considers the textual difference between the changes (as generated, for example, by the Unix tool ’diff’). We published both a paper on C3 and the dataset (consisting of groups of similar changes) that we generated for the evaluation of our tool under an open-source license, see https://github.com/FAU-Inf2/cthree . This dataset can be used as a reference or as input data for future research. In addition, we prototypically extended C3 by techniques for an incremental similarity computation and clustering.

This allows us to reuse results from previous runs and to only perform the absolutely necessary work whenever new code changes are added to a software archive.

2.2 Automatic Detection of Race-Conditions

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Michael Baer, M. Sc.
Start: 1.1.2016

Large software projects with hundreds of developers are difficult to review and often contain many bugs. Automatic tests are a well established technique to test sequential and deterministic software. They test the whole project (system test) or each module by itself (unit test). However, recent software contains more and more parallelism. This introduces several new bug patterns, like deadlocks and concurrent memory accesses, that are harder or even impossible to be detected reliably using conventional test methods. Whether the faulty behavior arises depends on the concrete scheduling of the threads which is indeterministic and varies between individual executions depending on the underlaying system. Due to this unpredictable behavior such bugs do not necessarily manifest in an arbitrary test run or may never arise in the testing environment at all. As a result, conventional tests are not well suited for modern, concurrent software.

With the project AuDeRace, we develop methods to efficiently and reliably detect concurrent bugs while keeping the additional effort for developers as low as possible. In an initial approach we define a testing framework that allows the specification of a scheduling plan to regain deterministic execution. However, a major problem still remains: The developer has to identify and implement well suited test cases that cover the potential fault in the program and execute them in a special deterministic way in order to trigger the failure. Especially in the context of concurrency, it is difficult to visualize the behavior of a program and identify the problematic parts. To overcome this, the critical parts shall automatically be narrowed down before even writing dedicated test cases. Existing approaches and tools for this purpose generate too many false positives or the analysis is very time consuming, making their application to real world code prohibitive. The goal of this project is to generate less false positives and increase the analysis speed by combining existing static and dynamic analysis. This allows for the efficient use in not only small example codes but also large and complex software projects.

In 2016 existing approaches were studied regarding their usability as a starting basis for our project. The most promising method uses model checking and predefined assertions

to construct thread schedules that trigger the faulty behavior. However, the approach is currently infeasible for larger projects because only very small codes could be analyzed in reasonable time. Therefore, we focused on automatically detecting and removing statements that are unrelated to the parallelism respectively to the potentially faulty code parts in order to decrease the execution time of the preliminary static analysis.

In 2017 the work on automatically reducing programs to speed up furher analysis was continued. Furthermore, we evaluated whether the concept of mutation testing can be applied to parallel software as well. The results indicate that this extension is indeed possible to rate tests qualitativly. However, to complete the analysis for larger programs within reasonable runtimes, a few heuristics need to be applied during the process.

This project is a contribution of the Chair of Computer Science 2 (Programming Systems) to the IZ ESI (Embedded Systems Initiative, http://www.esi.fau.de/)

2.3 Design for Diagnosability

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dr.-Ing. Josef Adersberger
Florian Lautenschlager, M. Sc.
Duration: 15.5.2013–30.9.2018
Sponsored by:
IuK Bayern

Many software systems behave obtrusively during the test phase or even in normal operation. The diagnosis and the therapy of such runtime anomalies is often time consuming and complex, up to being impossible. There are several possible consequences for using the software system: long response times, inexplicable behaviors, and crashes. The longer the consequences remain unresolved, the higher is the accumulated economic damage.

”Design for Diagnosability” is a tool chain targeted towards increasing the diagnosability of software systems. By using the tool chain that consists of modeling languages, components, and tools, runtime anomalies can easily be identified and solved, ideally already while developing the software system. Our cooperation partner QAware GmbH provides a tool called Software EKG that enables developers to explore runtime metrics of software systems by visualizing them as time series.

The research project Design for Diagnosability enhances the eco-system of the existing Software EKG. The Software-Blackbox measures technical and functional runtime values of a software system in a minimally intrusive way. We store the measured values

as time series in a newly developed time series database, called Chronix. Chronix is an extremely efficient storage of time series that optimizes disk space as well as response times. Chronix is an open source project (www.chronix.io) and is free to use for everyone.

The newly developed Time-Series-API analyzes these values, e.g., by means of an outlier detection mechanism. The Time-Series-API provides multiple additional building blocks to implement further strategies for identifying runtime anomalies.

The mentioned tools in combination with the existing Software EKG will become the so-called Dynamic Analysis Workbench. This tool enables developers to diagnose, explain, and fix any occurring runtime anomalies both quickly and reliably. It will provide diagnosis plans to localize and identify the root causes of runtime anomalies. The full tool chain aims at increasing the quality of software systems, particularly with respect to the metrics mean-time-to-repair and mean-time-between-defects.

Before we have successfully completed the project in July 2016, we have made the following contributions:

- We have linked Chronix and a framework for distributed data processing so that our anomaly analyses now scale to huge sets of time series data.
- We extended Chronix with additional components. Among them are, for example, a more efficient storage model, some adapters for more time series databases, additional server-side analysis functions, and some new time series types.
- We have published our benchmark for time series databases.
- We have investigated and implemented an approach to link application-level calls, e.g., a login of a user, down to the resulting calls on the OS level.

Although funding expired in 2016, we made further contributions in 2017:

- We presented Chronix at the FAST conference in Santa Clara, CA in February 2017.
- We have equipped Chronix with interfaces to attach time series databases that are used in the industry.
- We have developed an approach that determines the ideal cluster configuration (w.r.t. processing time and costs) for a given analysis (specific function and set of time series).

- We have expanded Spark, a framework for distributed processing of large-scale data, so that it now can make use of GPUs in distributed time series analyses. We presented the results at the Apache Big Data Conference in Miami, Florida, in May 2017.

2.4 Adaptive Algorithms for RF-based Locating Systems

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dr.-Ing. Christopher Mutschler
Duration: 15.5.2016–31.3.2017
Sponsored by:
Fraunhofer Institut für Integrierte Schaltungen

The goal of this project is the development of adaptive algorithms for radio-based realtime locating systems. In the scope of this project we cover three essential topics:

Automatic configuration of event detectors. In previous research projects we built the basics for the analysis of noisy sensor data streams. However, event detectors still need to be parameterized carefully to yield satisfying results. This work package explores the possibilities of an automatic configuration of the event detectors on existing sensor and event data streams. In 2016 we investigated concepts to extract optimal configurations from available sensor data streams. For soccer sport scientists manually annotated matches and scenes (e.g. player A kicks the ball with his/her left foot at time t). These manually annotated scenes may later by used to optimize the hierarchy of event detectors.

Evaluation of machine learning techniques for locating applications. In previous research projects we already developed machine learning algorithms for radio-based locating systems (e.g., evolutionary algorithms to estimate antenna positions and orientations). This work package investigates further approaches that use machine learning to enhance the performance of realtime locating systems. In 2016 we evaluated concepts to replace parts of the position estimation algorithms by machine learning algorithms. Up to now a signal processing chain (analog/digital conversion, time of arrival estimation, Kalman filtering, motion estimation) uses the raw sensor data to calculate a position. This often results in high installation and configuration costs for the setup of locating systems in the application environment.

Evaluation of vision-based techniques to support radio-based realtime locating. Radio-based locating systems have strengths if objects are occluded as microwaves may pass through the occluding objects. However, metallic surfaces in the environment pose

challenges as they reflect RF-signals. Hence, the RF-signal that a transmitter emits arrives at the antennas over multiple paths. It is then often difficult to extract the directly received parts of the signal at the antenna and hence it is a challenge to properly estimate the distance between the antenna and the emitter. In this work package we investigate vision-based locating techniques that may help RF-based systems in calculating positions. In 2016 we developed two systems: CNNLok may be used by objects carrying a camera (self-localization), i.e., inside-out tracking, whereas InfraLok uses cameras installed in the environment to track objects with infrared light. CNNLok uses a convolutional neural network (CNN) that is trained on several camera images taken in the environment (at known places). At runtime the CNN receives a camera image and calculates the position of the camera. InfraLok detects infrared LEDs using a multi-camera system and calculated the position of objects in space.

2.5 Incremental Code Analysis

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Math. Jakob Krainz
Duration: 1.4.2012–30.6.2017

To ensure that errors in a program design are caught early in the development process, it is useful to detect mistakes already during the editing of the code. For that the employed analysis has to be fast enough to enable interactive use. One method to achieve this is the use of incremental analysis, which combines analysis results of parts of the program to analyze the whole program. As an advantage, it is then possible to re-use large parts of the analysis results when a small change to the program occurs, namely for the unaffected parts of the program and for libraries. Thus the work required for analysis can be drastically reduced, which makes the analysis useful for interactive use.

Our analysis is based on determining, for (parts of) functions, which effects their execution can have on the state of a program at runtime. To achieve this, the runtime state of a program is modeled as a graph, which describes the variables and objects in the program's memory and their points-to relationship. The function is executed symbolically, to determine the changes made to the graph or, equivalently, to the runtime state described by it. To determine the effects of executing pieces of code in order, function calls, loops, etc., the change descriptions for smaller parts of the program can be combined in various ways, resulting in descriptions for the behavior of larger parts of the program. The analysis works in a bottom-up fashion, analyzing a called method before analyzing the callee (with recursion being analyzable as well).

In 2017 we continued improving the algorithms and data structures used for the analysis. In addition to further development of the analysis' scalability towards large code bases, and of incremental analysis (where we re-used the analysis results for unmodified program parts), we focused on an easy to grasp documentation of the analysis, in order to understand it, and to lay theoretical basics to verify the analysis' correctness.

2.6 International Collegiate Programming Contest at the FAU

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Daniela Novac
Michael Baer, M. Sc.
Dipl.-Math. Jakob Krainz
Dipl.-Inf. Tobias Werth
Start: 1.11.2002

The Association for Computing Machinery (ACM) has been hosting the International Collegiate Programming Contest (ICPC) for many years. Teams of three students try to solve nine to eleven programming problems within five hours. What makes this task even harder, is that there is only one computer available per team. The problems demand for solid knowledge of algorithms from all areas of computer science and mathematics, e.g., graphs, combinatorics, strings, algebra, and geometry.

The ICPC consists of three rounds. First, each participating university hosts a local contest to find the up to three teams that are afterwards competing in one of the various regional contests. Germany lies in the catchment area of the Northwestern European Regional Contest (NWERC) with competing teams from Great Britain, Benelux, Scandinavia, etc. The winners of all regionals in the world (and some second place holders) advance to the world finals in spring of the following year (2018 in Beijing, China).

In 2017 two local contests took place in Erlangen. In the winter semester we conducted a team contest with teams consisting of at most three students. The main goal of this contest was to interest new students in the contests. We had 29 FAU teams plus 35 more teams from universities all over Europe. Before the second contest, as in the previous years, in the summer term the seminar "Hello World - Programming for the Advanced" served to prepare students from different disciplines in algorithms and contest problems. In the German-wide contest of the summer term we selected the students that would represent the FAU at the NWERC 2017 in Bath (UK). A record number of 36 teams with students of computer science, computational engineering, mathematics as well as informations and communication technology took the challenge.

We formed the NWERC teams out of the best participants in the qualifications, given the age restrictions. At the NWERC in Bath, our teams reached places 14, 29 and 57 out of the 120 teams participating - a solid result, only narrowly missing on a medal.

2.7 Parallel code analysis on a GPU

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Thorsten Blaß
Start: 1.7.2013

In compiler construction there are analyses that propagate information along the edges of a graph and modify it, until they reach a fix point.

In this project we build a framework to accelerate such analyses by exploiting the massive parallelism of graphic cards.

Graphs are fundamental data structures to represent relations between data (e.g., social networks, web link analysis). Graphs can have millions/billions of vertices/edges. GPUs can process graphs with 1000th of threads in parallel very efficiently. Graph Analyses use the Bulk Synchronous Parallel Model (BSP) which divides the analysis into three strictly separated phases: computation, communication and synchronization. The two latter ones require communications with the host system (CPU) that slow down execution. Our GPU-based compiler works after the BSP model too. Internally the code is represented as (control flow-) graph. This graph is transferred to the GPU and gets analyzed. Every code modification triggers this cycle. The Graph has thus to be generated and transferred to the GPU very fast. Publications in the field of graph-analysis focus on optimizing the computation time. The end-to-end execution time (including communication and synchronization) is ignored but has a strong impact on the run-time. Our compiler considers every phase of the BSP. In 2017 we published a paper that significantly reduces the time for synchronization.

In addition, we focus on speeding up of the communication phase of the BSP model. Communication here means the transfer of the graph in both directions (GPU <-> Host). While the graph and data structure used has strong impact on the run-time behavior it also influences the computation phase. Since there is no publication in the literature that systematically investigates the impact of the data-structure on the end-to-end run-time of a GPU graph analysis, we implemented a number of benchmarks that use different attributes of graphs (e.g., access successor/predecessor, random node access) and eight different graph data structures to represent graphs on the GPU. For the

measurements we used a number of structurally different graphs. The results are likely to help developers in picking the right graph data structure for there GPU-problem.

2.8 Software Watermarking

Project manager:
Prof. Dr. Michael Philippsen
Project participants:
Dipl.-Inf. Daniela Novac
Start: 1.1.2016

Software watermarking means hiding selected features in code, in order to identify it or prove its authenticity. This is useful for fighting software piracy, but also for checking the correct distribution of open-source software (like for instance projects under the GNU license). The previously proposed methods assume that the watermark can be introduced at the time of software development, and require the understanding and input of the author for the embedding process. The goal of our research is the development of a watermarking framework that automates this process by introducing the watermark while compiling the code and can also be used for existing code. As a first approach we study a method that is based on symbolic execution and function synthesis.

In 2017 several function synthesis methods were analysed in order to determine the most appropriate one for our approach.

3 Publications 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.

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

– 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 Exam theses (german only)

– 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