Research Summary

Ralf Lämmel

December 2020

Abstract

- Section 1 summarizes **ongoing research**.
- Section 2 suggests topics for **planned research**.
- Section 3 briefly recounts **past research**.

1 Ongoing Research

My current research can be largely organized in three tracks. Here I do not cover recent industrial research carried out over the last 3.5 years at Facebook, but some of these topics are addressed in Section 2 on planned research. Each track is characterized according to the following 'metamodel':

Research context
What is this research about and why is it important?

Research challenges
What are the challenges that require significant research?

Research objectives
What properties or deliverables does the research aim at?

Research approach
What research methods or techniques does the research employ?

Related publications
What publications have been produced already in the track?

Status
What is the completion status for the different aspects in the track?

Funding
What is the funding situation for the research track?

1.1 Language-integrated semantic queries

Research context Semantic data is now part of much software architectures for the services and apps in the internet. For instance, Google and Microsoft maintain knowledge graphs that enhance internet search. Wikidata is an open-source knowledge graph that stores structured data for Wikipedia with an ontology defined by Schema.org which structures Wikidata. In the field of life sciences, Bio2RDF is a biological database which provides interlinked life science data with many billion triples. These examples demonstrate that semantic data models (e.g., RDF or OWL) are important for representing knowledge in complex use cases. Clearly, all such data needs to be queried in many different ways, to which end not just designated (stand alone) query languages are needed, but also programmatic access in diverse applications. The adequate support for such crucial, programmatic access is the theme of this ongoing research track.

Research challenges When compared to relational databases or object graphs in programming, semantic data involves several characteristics that call for a designated, original approach to programmatic data access. In particular, i) conceptualizations (schemas) are part of the semantic data; ii) reasoning is used to infer additional data; iii) semantic data is graph-or entity relationship-like; iv) the semantics of the data model is usually based on an open-world assumption; v) not just the datasets (in terms of numbers of triples), also the actual schemas (e.g., in terms of nominal and expressible concepts) are extra-huge. Overall, typed programming languages, as they exist, do not support programmatic access to semantic data in a manner that the description logic-based model of semantic data would be reasonably enforced in programming, which has consequences comparable to the lack of type safety in programming and impedance mismatches in data access that make data programmability error-prone and complicated.

Research objectives The concepts from semantic data must serve as types in programming. Hence, description logics need to be amalgamated with classic types. The notion of subtyping or subsumption needs to be generalized to also cover logic-based subsumption. Semantic data queries need to be expressible within programs, subject to interpolation of program variables, also subject to type checking that, in fact, involves reasoning. A formal language model and a practically applicable language integration are to be supplied. The former is to be formally analyzed. The latter is to be empirically validated.

Research approach The formal foundation of programmatic data access relies on the integration of description logics for typing and reasoning into a basic calculus (e.g., a lambda calculus) the type system and dynamic semantics is defined by appropriate reduction systems, subject to a proof of soundness, as standard in programming language theory. The practical realization of programmatic data access relies on actual language integration, subject to the instantiation of a suitable language extension architecture. The approach is validated by appropriate benchmark examples, an empirical analysis to provide evidence for the need for such an approach based on open-source traces of semantic data queries, and controlled experiments that ultimately show that users benefit from the integration.

Related publications

- Philipp Seifer, Martin Leinberger, Ralf Lämmel, and Steffen Staab. Semantic query integration with reason. *Art Sci. Eng. Program.*, 3(3):13, 2019
- Martin Leinberger, Philipp Seifer, Claudia Schon, Ralf Lämmel, and Steffen Staab. Type checking program code using SHACL. In Chiara Ghidini, Olaf Hartig, Maria Maleshkova, Vojtech Svátek, Isabel F. Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois, and Fabien Gandon, editors, *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part I*, volume 11778 of *Lecture Notes in Computer Science*, pages 399–417. Springer, 2019
- Martin Leinberger, Philipp Seifer, Tjitze Rienstra, Ralf Lämmel, and Steffen Staab. Deciding SHACL shape containment through description logics

reasoning. In Jeff Z. Pan, Valentina A. M. Tamma, Claudia d'Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal, editors, *The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, Athens, Greece, November 2-6, 2020, Proceedings, Part I*, volume 12506 of *Lecture Notes in Computer Science*, pages 366–383. Springer, 2020

- Martin Leinberger, Ralf Lämmel, and Steffen Staab. The essence of functional programming on semantic data. In Hongseok Yang, editor, *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10201 of *Lecture Notes in Computer Science*, pages 750–776. Springer, 2017

Status In current work, we need to incorporate update/deletion for semantic data, the use of metadata, the composition of queries for advanced query languages within programs, and the optimization of language-integrated queries. Further, we also aim at validating the corresponding language integrations or extensions by controlled experiments, where we have so far limited ourselves to 'indirect' means of validation by supporting the relevance of the work on the grounds of mining data from open source repositories.

Funding This track has been carried out within a project funded by DFG (German National Science Foundation). This project will end in summer 2021. This track was also supported by Microsoft Research funding. We are in the process of seeking additional funding; see Section 2.1.

1.2 Usage analysis of languages and technologies

Research context The need for understanding the usage of programming languages or other software languages as well as the usage of programming technologies (such as development tools or libraries or APIs) arises in many contexts of software engineering. For instance:

- Understanding the way in which APIs are used provides fundamental insight into prevalent programming styles, which needs to be taken into account in computer science education. For instance, API-usage analysis, as reported in some of the papers in the list below, found that the object-oriented technique of inheritance is hardly ever used, despite most APIs being carefully designed to enable it.
- Understanding the most important patterns of usage for a given domain-specific language enables a more native support of the patterns, thereby reducing error-prone boilerplate code; see, for example; the paper on privacy policies in the list below, which found that a large part of the language was effectively dead, thereby supporting, for example, the theory that the language did not well support privacy policies, as needed in practice.

- Being able to extract actual usage patterns for complex programming technologies from existing (open-source) software projects, provides us with a descriptive model of technology usage, which helps developers subsequently to actually understand and correctly use the technology; see, for example, the paper on EMF code generation in the list below.

A systematic approach to the analysis of language and technology usage is the theme of this ongoing research track.

Research challenges One challenge is that there is not a single uniform approach for all important instances of usage analysis. The actual approach depends much on the language or technology at hand and the underlying research questions for the analysis. For instance, consider the following two questions: i) Is a given type of code clones common in a given corpus? ii) Can a certain design pattern be detected in a given corpus? These two questions are far appart in terms of the needed mechanics of the analysis, even when considered for the same language. Another challenge concerns the methodology for the analysis. For instance, how to locate representative evidence and how to deal with mixed effects or hierarchical structure in the data? These and other methodological aspects are extremely important to mitigate threats regarding external and internal validity. Another challenge concerns the fact that the usage of languages and technologies is intertwined because the use of technologies almost certainly involves designated languages or designated use of languages; likewise, most programs intimately interact with technologies (e.g., through annotations, generated stubs or library calls). Yet another challenge concerns scalability of the relevant analyses, especially when a large corpus such as 'all' (relevant) open-source repositories is considered.

Research objectives An overarching objective is to contribute to the improvement of the state of the art in usage analysis for languages and technologies. To this end, we engage in important case studies with research questions that, in themselves, help solving important problems in software engineering; see some of the papers below. We have been addressing case studies that help i) deciding on the relevance of language constructs, ii) understanding the possibly limited diversity of language usage, thereby challenging the relevance or suitability of the language, and iii) extracting descriptive models of technologies that had previously only informal descriptions. Another objective in this research track is to address scalability-related hurdles, e.g., by means of appropriate incrementality in the analysis. Yet another objective is to create 'reusable' methodologies that handle the more complex situations of mixed-effect models in regression-based analyses.

Research approach This track deploys methodologies, as known from software reverse engineering, program comprehension, and mining software repositories. Typically, we use phases like data extraction, data cleaning/filtering (preprocessing), data analysis, and compilation of the results (e.g., by means of visualization). Those phases are set up according to the underlying research questions and the assumed theory. When we address scalability challenges, we use principles of benchmarking for validation. When we address mathematical aspects (e.g., mixed effect models), we explore the parameter space and we check on correlations and error rates of different types.

Related publications

- Johannes Härtel and Ralf Lämmel. Why to use mixed-effect models in msr? To be submitted., 2021
- Philipp Seifer, Johannes Härtel, Martin Leinberger, Ralf Lämmel, and Steffen Staab. Empirical study on the usage of graph query languages in open source java projects. In Oscar Nierstrasz, Jeff Gray, and Bruno C. d. S. Oliveira, editors, *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2019, Athens, Greece, October 20-22, 2019*, pages 152–166. ACM, 2019
- Johannes Härtel, Marcel Heinz, and Ralf Lämmel. EMF patterns of usage on github. In Alfonso Pierantonio and Salvador Trujillo, editors, *Modelling Foundations and Applications - 14th European Conference, ECMFA@STAF 2018, Toulouse, France, June 26-28, 2018, Proceedings*, volume 10890 of *Lecture Notes in Computer Science*, pages 216–234. Springer, 2018
- Johannes Härtel, Hakan Aksu, and Ralf Lämmel. Classification of apis by hierarchical clustering. In Foutse Khomh, Chanchal K. Roy, and Janet Siegmund, editors, *Proceedings of the 26th Conference on Program Comprehension, ICPC 2018, Gothenburg, Sweden, May 27-28, 2018*, pages 233–243. ACM, 2018
- Juri Di Rocco, Davide Di Ruscio, Johannes Härtel, Ludovico Iovino, Ralf Lämmel, and Alfonso Pierantonio. Understanding MDE projects: megamodels to the rescue for architecture recovery. *Softw. Syst. Model.*, 19(2):401–423, 2020
- Marcel Heinz, Johannes Härtel, and Ralf Lämmel. Reproducible construction of interconnected technology models for EMF code generation. *J. Object Technol.*, 19(2):8:1–25, 2020
- Ralf Lämmel and Ekaterina Pek. Understanding privacy policies - A study in empirical analysis of language usage. *Empir. Softw. Eng.*, 18(2):310–374, 2013
- Coen De Roover, Ralf Lämmel, and Ekaterina Pek. Multi-dimensional exploration of API usage. In *IEEE 21st International Conference on Program Comprehension, ICPC 2013, San Francisco, CA, USA, 20-21 May, 2013*, pages 152–161. IEEE Computer Society, 2013
- Ralf Lämmel, Rufus Linke, Ekaterina Pek, and Andrei Varanovich. A framework profile of .net. In Martin Pinzger, Denys Poshyvanyk, and Jim Buckley, editors, *18th Working Conference on Reverse Engineering,*

WCRE 2011, Limerick, Ireland, October 17-20, 2011, pages 141–150. IEEE Computer Society, 2011

- Ralf Lämmel, Ekaterina Pek, and Jürgen Starek. Large-scale, ast-based api-usage analysis of open-source java projects. In William C. Chu, W. Eric Wong, Mathew J. Palakal, and Chih-Cheng Hung, editors, *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC), TaiChung, Taiwan, March 21 - 24, 2011*, pages 1317–1324. ACM, 2011
- Ralf Lämmel and Ekaterina Pek. Vivisection of a non-executable, domain-specific language - understanding (the usage of) the P3P language. In *The 18th IEEE International Conference on Program Comprehension, ICPC 2010, Braga, Minho, Portugal, June 30-July 2, 2010*, pages 104–113. IEEE Computer Society, 2010
- Jean-Marie Favre, Dragan Gasevic, Ralf Lämmel, and Ekaterina Pek. Empirical language analysis in software linguistics. In Brian A. Malloy, Steffen Staab, and Mark van den Brand, editors, *Software Language Engineering - Third International Conference, SLE 2010, Eindhoven, The Netherlands, October 12-13, 2010, Revised Selected Papers*, volume 6563 of *Lecture Notes in Computer Science*, pages 316–326. Springer, 2010

Status We initiated this research track around 10 years ago, beginning with fairly straightforward analyses of API usage (from today's perspective). More recently, we succeeded conducting studies on language and technology usage analysis with methodologies that are complex in terms of the research questions and mechanics of analyses. For instance, we recently completed a study on graph query languages, which informs our other research track on language integration for semantic data queries; see Section 1.1. This work required an analysis for language embedding for an entire family of languages. In our ongoing research, we address questions related to the use of annotations in Wikidata, thereby preparing a future research track on provenance; see Section 2.1. We also carry out research on the reproducibility of results in the mining software repository domain; we expect to show that the use of historically more naive models implies that some of the published effects do not hold.

Funding A significant part of this work has been supported by projects funded by DFG (German National Science Foundation), Krupp Stiftung, and BM RLP (Ministry of Education, Rhineland Palatinate).

1.3 Models of linguistic software architecture

Research context A reduced, in fact, outdated view of 'programming' would center around the use of programming languages for algorithms and data structures. However, modern 'programming' (in fact, software development) largely relies on a platform- and system-specific portfolio of software technologies such as compilers, code generators, frameworks, and APIs. These technologies come with their own domain-specific notations, protocols, and yet other linguistic means of using the technologies and integrating such use with programming functionality. We refer to this structure of software systems as their *linguistic software architecture*. Modelling support for capturing such architecture is the theme of this ongoing research track. This is an important research area because the complexity of software technologies has reached a point where developers can hardly comprehend and apply them while appropriate models can powerfully abstract from idiosyncrasies and specifics of use and serve both prescriptive (guiding) and descriptive (explanatory) purposes. For example, the linguistic architecture of a code generator (or, in fact, of a system which uses the code generator) would clarify the different generated and non-generated artefacts, the languages that are used by those artefacts, the steps to realize the code generation, and also any sort of client components that would benefit from the generated code.

Research challenges We also refer to models of linguistic software architecture as technology models. (There is also the related term of megamodels which however often assumes a model-driven engineering focus.) The initial challenge with technology modeling is one of language design: we need a modeling language for the domain of linguistic architecture. Subsequent challenges are concerned with actual authoring or mining of models and validation or use of models in practice. For a large part, linguistic software architecture operates in the realm of ontology engineering if we assume an emerging ontology of technology usage.

Research objectives The objectives can be derived here directly from the challenges. We aim at a modeling language for technology usage. We need to be able to populate this language with models in a scalable manner such that we can eventually cover all representative usage patterns that are relevant in practice. To this end, we need to combine authoring (thereby requiring human work) and mining (thereby leveraging automation, but challenging validation in return). We also need to interconnect the models with the reality so that one can understand how a given technology model is an effective abstraction of a given, concrete system. We also need to support the prescriptive direction so that developers can implement new components that comply with given technology models by construction.

Research approach The methodological space of this research track is vast. An overview can be extracted from the objectives. These methods and techniques are involved: software/modeling language design and implementation (the latter specifically to serve validation or prescriptive purpose), mining software repositories (to extract usage, thereby building on Section 1.2), linked data (for the aforementioned interconnection), ontology engineering (for maturing the essential ontology for technologies over time).

Related publications

- Ralf Lämmel. Megamodels on the catwalk. Keynote at MODELSWARD 2021 `http://www.modelsward.org/`, 2021
- Marcel Heinz, Ralf Lämmel, and Mathieu Acher. Discovering indicators for classifying wikipedia articles in a domain - A case study on software languages. In Angelo Perkusich, editor, *The 31st International Conference on Software Engineering and Knowledge Engineering, SEKE 2019, Hotel Tivoli, Lisbon, Portugal, July 10-12, 2019*, pages 541–706. KSI Research Inc. and Knowledge Systems Institute Graduate School, 2019
- Jean-Marie Favre, Ralf Lämmel, and Andrei Varanovich. Modeling the linguistic architecture of software products. In Robert B. France, Jürgen Kazmeier, Ruth Breu, and Colin Atkinson, editors, *Model Driven Engineering Languages and Systems - 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30-October 5, 2012. Proceedings*, volume 7590 of *Lecture Notes in Computer Science*, pages 151–167. Springer, 2012
- Jean-Marie Favre, Ralf Lämmel, Martin Leinberger, Thomas Schmorleiz, and Andrei Varanovich. Linking documentation and source code in a software chrestomathy. In *19th Working Conference on Reverse Engineering, WCRE 2012, Kingston, ON, Canada, October 15-18, 2012*, pages 335–344. IEEE Computer Society, 2012
- Ralf Lämmel and Andrei Varanovich. Interpretation of linguistic architecture. In Jordi Cabot and Julia Rubin, editors, *Modelling Foundations and Applications - 10th European Conference, ECMFA@STAF 2014, York, UK, July 21-25, 2014. Proceedings*, volume 8569 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2014
- Johannes Härtel, Lukas Härtel, Ralf Lämmel, Andrei Varanovich, and Marcel Heinz. Interconnected linguistic architecture. *Art Sci. Eng. Program.*, 1(1):3, 2017
- Marcel Heinz, Ralf Lämmel, and Andrei Varanovich. Axioms of linguistic architecture. In Luís Ferreira Pires, Slimane Hammoudi, and Bran Selic, editors, *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2017, Porto, Portugal, February 19-21, 2017*, pages 478–486. SciTePress, 2017
- Marcel Heinz, Johannes Härtel, and Ralf Lämmel. Reproducible construction of interconnected technology models for EMF code generation. *J. Object Technol.*, 19(2):8:1–25, 2020

Status In our ongoing research, we are largely concerned with maturing the initial research results regarding axiomatization of modeling relations, the validation/interpretation semantics for the models, and the comprehensive application of the method to representative problems (technologies). Also, we need to dedicate research and community efforts to complete more ontology engineering passes. Only then, we will be able to achieve a long-term and strong impact with this research.

Funding This work has been supported by a project funded by BM RLP (Ministry of Education, Rhineland Palatinate), Krupp Stiftung, and basic funding of the department.

2 Planned Research

I am currently looking at three tracks for planned research. These tracks are at different stages of clarity and funding. Several of these tracks are motivated by industrial research I carried out until recently at Facebook in the broad context of software engineering and infrastructure. Each track is characterized according to the following 'metamodel':

Research context
What is this research about and why is it important?

Research challenges
What are the challenges that require significant research?

Research objectives
What properties or deliverables does the research aim at?

Research approach
What research methods or techniques does the research employ?

Resources and funding
What resources are needed and how could they be funded?

2.1 End-to-end provenance

Research context Data provenance has become a critical property of data intelligence systems. For instance, any system making decisions based on machine learning or otherwise, is now typically required to support proper provenance so that the origin of all data can be tracked and understood and even taken into account in reasoning. Legislation indeed requires data provenance for certain kinds of systems. Thus, data provenance is becoming part of the functional correctness notion for (data intelligence) systems. This planned research track is focused on addressing provenance, in its broadest sense, in the context of systems that involve semantic data queries (e.g., on knowledge graphs) — much in the sense of Section 1.1.

Research challenges We speak of end-to-end provenance here because we specifically focus on the challenges due to the fact that semantic data may be stored in a knowledge graph (a triple store), but queries are performed in a program, and query results may flow into other program components, which makes it clearly harder to support provenance. Thus, we certainly face challenges in terms of program and query analysis. An additional area of concern is that the underlying schema for semantic data is often not clear enough in terms of provenance-like properties and semantics. For instance, Wikidata attributes certain properties to certain sources, but it is actually hard to unambiguously resolve the scope of attribution and close inspection of some sample sources suggests that attribution may be imprecise, for example, in terms of neglecting indirection levels such as the source quoting another source contributing the actual attribution.

Research objectives We need to take a comprehensive inventory of provenance-like metadata in semantic data and the related issues of scope and semantics. We expect to suggest a stronger conceptualization technique so that provenance-like aspects can be more strongly modeled in terms of annotations. We plan to also carry on all such provenance/scope-like metadata into the programmatic access layer, i.e., into programs, thereby refining our earlier work on language integrated semantic queries. Semantic data with annotations must become first class in programming. Semantic queries have to be composable (i.e., query results can be queried again) while preserving provenance along querying, but also when query results transition into regular data structures in programs.

Research approach The intended stronger conceptualization relies on a refinement of the semantic data model, subject to input from the field of metamodeling (e.g., multilevel modeling) and scopes in programming languages. The tracking of provenance from triple store to query, along composed queries, into classic data structures requires a generalized form of information flow analysis. Overall, our previous work on language-integrated semantic queries (Section 1.1) will be revisited and advanced.

Resources and funding A funding application for a project has been submitted to DFG (German National Science Foundation) in October 2020. This planned research is also part of an emerging research consortium on trustworthiness in data intelligence. I collaborate with Prof. Steffen Staab from the University of Stuttgart on this topic and we expect to fund at least 1 PhD student on each side. In fact, there is currently one PhD student and one PostDoc in Koblenz who help with initiating this effort.

2.2 Developer workflow analysis

Research context Understanding the workflow of developers, as they approach tasks, resolve bugs, integrate changes, debug problems, etc. is of huge importance for any sort of progress in software development. Consider the following problem in a software company (e.g., any of the Big Tech companies). Business intelligence needs to determine whether developer productivity in terms of so-called key-performance indicators decay in the view of changes to infrastructure, processes, or other parameters that may possibly affect developer performance within a company at an aggregated level. More specifically, the management would like to monitor ‘time to complete a software change’ or ‘time to review a submitted software change’. For such aggregation of times spent to be possible, we need to understand developer workflow at a rather detailed level and at a high degree of precision. The corresponding analysis is the theme of this planned research track. See also this paper for further context:

> Ralf Lämmel, Alvin Kerber, and Liane Praza. Understanding what software engineers are working on: The work-item prediction chal-

lenge. In *ICPC '20: 28th International Conference on Program Comprehension, Seoul, Republic of Korea, July 13-15, 2020*, pages 416–424. ACM, 2020

Research challenges Understanding (analyzing) developer workflow is challenging in a general setting for several reasons: i) How to associate diverse developer actions with specific work items? (Only then we would understand what a developer is working on.) ii) How to abstract workflows at a useful level of abstraction? (Only then we could compare and aggregate behaviors.) iii) How to detect and remove noise from the traces of developer actions? (For instance, non-work-related intermezzos of a developer should be removed both for privacy reasons and not to affect any work-related interpretations.) Simpler forms of developer workflow analysis have been addressed in the literature, for example, the analysis of workflow related to the use of version control or the commit/review cycle. However, developer workflow analysis is much harder, when we indeed aim at covering developer actions comprehensively, i.e., when we include, for example, work on documentation or debugging sessions or local commits that do not make it into a public commit.

Research objectives We aim at a reference architecture for logging and integrating all logging of developer activity which is both realistic in terms of common practice of logging and comprehensive enough to enable developer workflow analysis. We further aim at a portfolio of heuristics and machine learning components that can resolve typical challenges of understanding developer workflow. Finally, the best way of showing that developer workflow can be analyzed is to demonstrate precision or correctness based on representative case studies; see also the aforementioned paper for some scenarios, for example, automated documentation of on-call or alert-response behavior or aggregation of key-performance indicators, thereby informing business intelligence.

Research approach This work is largely concerned with process mining, data mining, information retrieval, clustering, similarity analysis. Some of the work will need to rely on unsupervised machine learning since the costs of collecting labeled data, for example, for associating each and every developer action with a work item, is simply too high in practice. Another part of this research track will also publish minimal requirements for a developer logging and tooling infrastructure so that developer workflow analysis is reasonably possible (precise). This may require some reverse engineering efforts to understand the state of logging and tool integration in practice. This may also require some data integration and re-engineering efforts to describe possible paths towards better logger and tool integration.

Resources and funding I am working on a funding application for some of the more fundamental research-related aspects, but we will be starting shortly to work on the topic in my research team based on available basic funding – also subject to master thesis-level work. The general track definitely requires several PhD students to complete the vision sketched above. This track would also benefit from collaboration with industrial partners for the purpose of clarifying some of the requirements and for validation of some of the components in practice.

2.3 Reviewer engagement recommendation

Research context The code review process is an integral part of producing correct and maintainable software: a change, as committed by one developer for review and eventual integration with the 'master' branch in the code base or a release, is reviewed by peers (i.e., other developers) who may recommend or request changes, before the commit can ultimately 'land'. While code review is assumed to be necessary for quality assurance as well as knowledge sharing, there is the very real risk that productivity is negatively affected. In particular, submitters may be blocked from making progress, if reviewers are only slowly delivering feedback. Also, submitters may be slowed down, if reviewer feedback asks for changes that arguably do not improve quality. Likewise, reviewers may be overwhelmed with review requests and may feel obliged to provide feedback, but due to a lack of guidance, they may end up focusing on minor problems (so-called 'nitpicking'). This planned research track aims at improving reviewer engagement, thereby improving commit and review quality, while improving developer productivity at the same time. Reviewer engagement is subject to the recommendation of code segments of a commit that are particularly worthy of reviewer attention complete with input as to the nature of potential quality problems.

Research challenges There is an established research area of so-called fault prediction which uses, for example, machine learning models to predict with a probability that a given commit introduces a bug that would need to be fixed later or that a given commit could trigger crashes or imply security problems. There is also work on automated bug fixing where a machine learning model may advice on code fixes that are known from the past. Fault prediction and automated bug fixing concern code quality in terms of correctness. There is also an established research area of refactoring recommendation, which uses software metrics and possibly search-based software engineering to determine the applicability of refactorings with positive impact. Refactoring recommendation concerns code quality in terms of maintainability. None of these approaches address reviewer engagement, except for the most clear-cut cases such as 'apply this refactoring' or 'accept this code fix'. Reviewer engagement recommendation requires code-segment level advice and requires understanding the intent of a change and also requires an explanation of the potential quality issue. Also, it may not be enough to just learn from past changes and potential quality problems, as they showed up; we also need to track code churn over time to learn from trends. While recommendations should be actionable, they should not be limited to clear cases of recommending a particular compensatory code change.

Research objectives The overarching objective is to develop a recommendation system that can enrich a given candidate commit within the scope of a given repository and the version timeline with actionable advice that guides reviewers in terms of what code segments to look at closer and for what reason. In fact, such information would also be directly useful for commit authors, as they may want to preempt change requests. To this end, we need a component that understands code segments likely to receive justified and relevant reviewer comments — complete with the nature of the comments to be expected. Validation is required to determine whether deployment of such a recommendation system increases productivity for commit authors and reviewers and increase code and review quality.

Research approach We combine methodologies known from mining software repositories and program comprehension with specific machine learning approaches that help us to predict code changes at a code-segment level, the intents of a given commit, and the nature of the predicted code changes. In terms of the focus on research questions, we would primarily address code quality in terms of maintainability as well as knowledge sharing, but less so code quality in terms of correctness, as the latter has been addressed by a number of highly specialized approaches in the past. Validation would be ideally based on some form of A/B testing to determine whether the cohort with access to recommendations performs better than the control group. Such A/B testing may be hard to perform in a real setting. Additionally (or instead), we may perform a controlled experiment or leverage some form of simulation involving strategically withheld data.

Resources and funding I have addressed this area in preliminary, as yet, unpublished industrial research at Facebook until recently. In a next step, I plan to work with several students in the scope of their master-thesis projects, to better understand the problem in terms of related work and feasibility of using open-source software repositories for validation. I expect to submit a funding application later in 2021. This track would clearly benefit from collaboration with industrial partners especially in the context of validation.

3 Past Research

The following areas are mentioned *in passing* to better capture my research background and to hint at areas or methods that may be of value in the future or are, in fact, used 'under the hood' in current research. **This section is likely to be parseable only for an expert in software engineering and programming languages.**

3.1 Generic functional programming

In cooperation with Simon Peyton Jones and Joost Visser, I developed an influential and original approach to generic programming (in the sense of typed functions that can be applied uniformly to 'all' types). This approach is implemented in compilers and libraries and has use cases in, for example, software refactoring and program analysis. The 2004 paper from the following list received the 'Most influential paper award at the ACM International Conference on Functional Programming 2014'.

Related publications

- Ralf Lämmel and Joost Visser. Typed combinators for generic traversal. In Shriram Krishnamurthi and C. R. Ramakrishnan, editors, *Practical Aspects of Declarative Languages, 4th International Symposium, PADL 2002, Portland, OR, USA, January 19-20, 2002, Proceedings*, volume 2257 of *Lecture Notes in Computer Science*, pages 137–154. Springer, 2002
- Ralf Lämmel and Joost Visser. A strafunski application letter. In Verónica Dahl and Philip Wadler, editors, *Practical Aspects of Declarative Languages, 5th International Symposium, PADL 2003, New Orleans, LA, USA, January 13-14, 2003, Proceedings*, volume 2562 of *Lecture Notes in Computer Science*, pages 357–375. Springer, 2003
- Ralf Lämmel and Simon L. Peyton Jones. Scrap your boilerplate: a practical design pattern for generic programming. In Zhong Shao and Peter Lee, editors, *Proceedings of TLDI'03: 2003 ACM SIGPLAN International Workshop on Types in Languages Design and Implementation, New Orleans, Louisiana, USA, January 18, 2003*, pages 26–37. ACM, 2003
- Ralf Lämmel and Simon L. Peyton Jones. Scrap more boilerplate: reflection, zips, and generalised casts. In Chris Okasaki and Kathleen Fisher, editors, *Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming, ICFP 2004, Snow Bird, UT, USA, September 19-21, 2004*, pages 244–255. ACM, 2004
- Ralf Lämmel and Simon L. Peyton Jones. Scrap your boilerplate with class: extensible generic functions. In Olivier Danvy and Benjamin C. Pierce, editors, *Proceedings of the 10th ACM SIGPLAN International Conference on Functional Programming, ICFP 2005, Tallinn, Estonia, September 26-28, 2005*, pages 204–215. ACM, 2005
- Ralf Lämmel. Scrap your boilerplate with xpath-like combinators. In Martin Hofmann and Matthias Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 137–142. ACM, 2007
- Anya Helene Bagge and Ralf Lämmel. Walk your tree any way you want. In Keith Duddy and Gerti Kappel, editors, *Theory and Practice of Model Transformations - 6th International Conference, ICMT@STAF 2013, Budapest, Hungary, June 18-19, 2013. Proceedings*, volume 7909 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2013

- Ralf Lämmel, Simon J. Thompson, and Markus Kaiser. Programming errors in traversal programs over structured data. *Sci. Comput. Program.*, 78(10):1770–1808, 2013

3.2 Big-data programming models

In particular, I performed early research on Google's MapReduce programming model, which also resulted in a publication with several hundreds citations, as it serves as the de-facto in-depth definition of the programming model. I continued to research this area sporadically and, with collaborators, made contributions to performance-related aspects; this also includes a recent paper which is tailored towards longitudinal data in mining software repositories.

Related publications

- Ralf Lämmel. Google's mapreduce programming model - revisited. *Sci. Comput. Program.*, 70(1):1–30, 2008
- Ralf Lämmel and David Saile. MapReduce with Deltas. In *PDPTA'11 - The 2011 International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 26–32. CSREA Press, 2011
- Johannes Härtel and Ralf Lämmel. Incremental map-reduce on repository history. In Kostas Kontogiannis, Foutse Khomh, Alexander Chatzigeorgiou, Marios-Eleftherios Fokaefs, and Minghui Zhou, editors, *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*, pages 320–331. IEEE, 2020

3.3 Advanced modularity mechanisms

I have been interested in advanced forms of modularity, subject to appropriate language support for a long time. The aspect-oriented software development community also hosted the modularity notion during the time of its existence. I contributed the first published formal semantics of the essential core of aspect-oriented programming. With collaborators, I also discovered and advanced some aspect-oriented expressiveness in the legacy language Cobol and I integrated (conceptually) specialized forms of aspect-oriented programming. Outside the narrow aspect-oriented scope, I also worked on advanced retroactive interface implementation and extensible collections.

Related publications

- Ralf Lämmel. A semantical approach to method-call interception. In Harold Ossher and Gregor Kiczales, editors, *Proceedings of the 1st International Conference on Aspect-Oriented Software Development, AOSD 2002, University of Twente, Enschede, The Netherlands, April 22-26, 2002*, pages 41–55. ACM, 2002
- Ralf Lämmel, Eelco Visser, and Joost Visser. Strategic programming meets adaptive programming. In William G. Griswold and Mehmet Aksit, editors,

Proceedings of the 2nd International Conference on Aspect-Oriented Software Development, AOSD 2003, Boston, Massachusetts, USA, March 17-21, 2003, pages 168–177. ACM, 2003

- Ralf Lämmel and Kris De Schutter. What does aspect-oriented programming mean to cobol? In Mira Mezini and Peri L. Tarr, editors, *Proceedings of the 4th International Conference on Aspect-Oriented Software Development, AOSD 2005, Chicago, Illinois, USA, March 14-18, 2005*, pages 99–110. ACM, 2005
- Robert Hirschfeld and Ralf Lämmel. Reflective designs. *IEE Proc. Softw.*, 152(1):38–51, 2005
- Ralf Lämmel and Christian Stenzel. Semantics-directed implementation of method-call interception. *IEE Proc. Softw.*, 151(2):109–128, 2004
- Oleg Kiselyov, Ralf Lämmel, and Keean Schupke. Strongly typed heterogeneous collections. In Henrik Nilsson, editor, *Proceedings of the ACM SIGPLAN Workshop on Haskell, Haskell 2004, Snowbird, UT, USA, September 22-22, 2004*, pages 96–107. ACM, 2004
- Stefan Wehr, Ralf Lämmel, and Peter Thiemann. Javagi : Generalized interfaces for java. In Erik Ernst, editor, *ECOOP 2007 - Object-Oriented Programming, 21st European Conference, Berlin, Germany, July 30 - August 3, 2007, Proceedings*, volume 4609 of *Lecture Notes in Computer Science*, pages 347–372. Springer, 2007

3.4 Grammarware and software language engineering

Early in my postdoctoral research, I worked on engineering methods for, what has been coined, 'grammarware', for example, methods for the transformation, recovery, and testing of grammars. Later I contributed to the generalization of grammarware engineering to software language engineering.

The TOSEM paper from the following list has guided much subsequent work by others, which is also reflected in hundreds of citations. The TestCom paper from the following list also demonstrates how grammarware-based work eventually generalized to software engineering (here: testing of program artefacts with some underlying grammar-like structure).

Related publications

- Johannes Härtel, Lukas Härtel, and Ralf Lämmel. Test-data generation for xtext - tool paper. In Benoît Combemale, David J. Pearce, Olivier Barais, and Jurgen J. Vinju, editors, *Software Language Engineering - 7th International Conference, SLE 2014, Västerås, Sweden, September 15-16, 2014. Proceedings*, volume 8706 of *Lecture Notes in Computer Science*, pages 342–351. Springer, 2014
- Bernd Fischer, Ralf Lämmel, and Vadim Zaytsev. Comparison of context-free grammars based on parsing generated test data. In Anthony M. Sloane and Uwe Aßmann, editors, *Software Language Engineering - 4th International Conference, SLE 2011, Braga, Portugal, July 3-4, 2011, Revised Selected Papers*, volume 6940 of *Lecture Notes in Computer Science*, pages 324–343. Springer, 2011

- Ralf Lämmel and Vadim Zaytsev. Recovering grammar relationships for the java language specification. *Softw. Qual. J.*, 19(2):333–378, 2011
- Vadim Zaytsev and Ralf Lämmel. A unified format for language documents. In Brian A. Malloy, Steffen Staab, and Mark van den Brand, editors, *Software Language Engineering - Third International Conference, SLE 2010, Eindhoven, The Netherlands, October 12-13, 2010, Revised Selected Papers*, volume 6563 of *Lecture Notes in Computer Science*, pages 206–225. Springer, 2010
- Ralf Lämmel and Vadim Zaytsev. Recovering grammar relationships for the java language specification. In *Ninth IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2009, Edmonton, Alberta, Canada, September 20-21, 2009*, pages 178–186. IEEE Computer Society, 2009
- Ralf Lämmel and Wolfram Schulte. Controllable combinatorial coverage in grammar-based testing. In M. Ümit Uyar, Ali Y. Duale, and Mariusz A. Fecko, editors, *Testing of Communicating Systems, 18th IFIP TC6/WG6.1 International Conference, TestCom 2006, New York, NY, USA, May 16-18, 2006, Proceedings*, volume 3964 of *Lecture Notes in Computer Science*, pages 19–38. Springer, 2006
- Paul Klint, Ralf Lämmel, and Chris Verhoef. Toward an engineering discipline for grammarware. *ACM Trans. Softw. Eng. Methodol.*, 14(3):331–380, 2005
- Steven Klusener and Ralf Lämmel. Deriving tolerant grammars from a base-line grammar. In *19th International Conference on Software Maintenance (ICSM 2003), The Architecture of Existing Systems, 22-26 September 2003, Amsterdam, The Netherlands*, page 179. IEEE Computer Society, 2003
- Ralf Lämmel and Chris Verhoef. Cracking the 500-language problem. *IEEE Softw.*, 18(6):78–88, 2001
- Ralf Lämmel and Chris Verhoef. Semi-automatic grammar recovery. *Softw. Pract. Exp.*, 31(15):1395–1438, 2001
- Ralf Lämmel. Grammar testing. In Heinrich Hußmann, editor, *Fundamental Approaches to Software Engineering, 4th International Conference, FASE 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*, volume 2029 of *Lecture Notes in Computer Science*, pages 201–216. Springer, 2001
- Ralf Lämmel. Grammar adaptation. In José Nuno Oliveira and Pamela Zave, editors, *FME 2001: Formal Methods for Increasing Software Productivity, International Symposium of Formal Methods Europe, Berlin, Germany, March 12-16, 2001, Proceedings*, volume 2021 of *Lecture Notes in Computer Science*, pages 550–570. Springer, 2001

3.5 Bidirectional and coupled transformations

In the 'early days' of bidirectional programming, I contributed a widely cited 4 pages workshop paper which provided a more inclusive definition of BX, which I recently also formalized. I have been engaging with the community and the topic ever since, which is also reflected in my co-authorship in a widely cited survey of the field. Much of my insight into the field was based on my interest in object/relational/XML mappings which I also worked on in the scope of industrial research at Microsoft. Some of this work continues today in the form of language-integrated semantic data queries.

Related publications

- Ralf Lämmel. Coupled software transformations. In *Proc. SET 2004 (First International Workshop on Software Evolution Transformations)*, pages 31–35, 2004. Extended Abstract. Available online at `http://softlang.uni-koblenz.de/cxrevisited/`
- Krzysztof Czarnecki, J. Nathan Foster, Zhenjiang Hu, Ralf Lämmel, Andy Schürr, and James F. Terwilliger. Bidirectional transformations: A cross-discipline perspective. In Richard F. Paige, editor, *Theory and Practice of Model Transformations - 2nd International Conference, ICMT@TOOLS 2009, Zurich, Switzerland, June 29-30, 2009. Proceedings*, volume 5563 of *Lecture Notes in Computer Science*, pages 260–283. Springer, 2009
- Davide Di Ruscio, Ralf Lämmel, and Alfonso Pierantonio. Automated co-evolution of GMF editor models. In Brian A. Malloy, Steffen Staab, and Mark van den Brand, editors, *Software Language Engineering - Third International Conference, SLE 2010, Eindhoven, The Netherlands, October 12-13, 2010, Revised Selected Papers*, volume 6563 of *Lecture Notes in Computer Science*, pages 143–162. Springer, 2010
- Ralf Lämmel. Coupled software transformations revisited. In Tijs van der Storm, Emilie Balland, and Dániel Varró, editors, *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering, Amsterdam, The Netherlands, October 31 - November 1, 2016*, pages 239–252. ACM, 2016
- Ralf Lämmel. Style normalization for canonical x-to-o mappings. In G. Ramalingam and Eelco Visser, editors, *Proceedings of the 2007 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation, 2007, Nice, France, January 15-16, 2007*, pages 31–40. ACM, 2007
- Ralf Lämmel. LINQ to XSD. In *PLAN-X 2007, Programming Language Technologies for XML, An ACM SIGPLAN Workshop colocated with POPL 2007, Nice, France, January 20, 2007*, pages 95–96, 2007
- Ralf Lämmel and Erik Meijer. Mappings make data processing go 'round. In Ralf Lämmel, João Saraiva, and Joost Visser, editors, *Generative and Transformational Techniques in Software Engineering, International Summer School, GTTSE 2005, Braga, Portugal, July 4-8, 2005. Revised Papers*, volume 4143 of *Lecture Notes in Computer Science*, pages 169–218. Springer, 2005
- Ralf Lämmel and Erik Meijer. Revealing the X/O impedance mismatch - (changing lead into gold). In Roland Carl Backhouse, Jeremy Gibbons, Ralf Hinze, and Johan Jeuring, editors, *Datatype-Generic Programming - International Spring School, SSDGP 2006, Nottingham, UK, April 24-27, 2006, Revised Lectures*, volume 4719 of *Lecture Notes in Computer Science*, pages 285–367. Springer, 2006

3.6 Software chrestomathies

Collections of programs useful for learning have existed for a longer time. It is due to me and collaborators that we lifted chrestomathies to the level of software engineering and worked them out in several dimensions such as their integration with ontological knowledge. These efforts have enabled the effective use of chrestomathies in university teaching.

Related publications

- Jean-Marie Favre, Ralf Lämmel, Thomas Schmorleiz, and Andrei Varanovich. 101companies: A community project on software technologies and software languages. In Carlo A. Furia and Sebastian Nanz, editors, *Objects, Models, Components, Patterns - 50th International Conference, TOOLS 2012, Prague, Czech Republic, May 29-31, 2012. Proceedings*, volume 7304 of *Lecture Notes in Computer Science*, pages 58–74. Springer, 2012
- Ralf Lämmel, Thomas Schmorleiz, and Andrei Varanovich. The 101haskell chrestomathy: A whole bunch of learnable lambdas. In Rinus Plasmeijer, editor, *Proceedings of the 25th Symposium on Implementation and Application of Functional Languages, Nijmegen, The Netherlands, August 28-30, 2013*, page 25. ACM, 2013
- Ralf Lämmel. Software chrestomathies. *Sci. Comput. Program.*, 97:98–104, 2015
- Simon Schauss, Ralf Lämmel, Johannes Härtel, Marcel Heinz, Kevin Klein, Lukas Härtel, and Thorsten Berger. A chrestomathy of DSL implementations. In Benoît Combemale, Marjan Mernik, and Bernhard Rumpe, editors, *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2017, Vancouver, BC, Canada, October 23-24, 2017*, pages 103–114. ACM, 2017

3.7 Software transformation

I have been always interested in foundations and applications of software transformation. For instance, I have worked on the transformation of executable language descriptions. The following list also contains an extended foreword for a special issue on software or program transformation which aimed at integrating otherwise scattered communities at that time. The following lists also illustrates some contributions I made with collaborators on specific problems of software transformation — partial evaluation (i.e., evaluating a program to the extent possible with existing partial input) and API migration (i.e., replacing usage of one API by another).

Related publications

- Bastian Ulke, Friedrich Steimann, and Ralf Lämmel. Partial evaluation of OCL expressions. In *20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2017, Austin, TX, USA, September 17-22, 2017*, pages 63–73. IEEE Computer Society, 2017

- William R. Cook and Ralf Lämmel. Tutorial on online partial evaluation. In Olivier Danvy and Chung-chieh Shan, editors, *Proceedings IFIP Working Conference on Domain-Specific Languages, DSL 2011, Bordeaux, France, 6-8th September 2011*, volume 66 of *EPTCS*, pages 168–180, 2011
- Thiago Tonelli Bartolomei, Krzysztof Czarnecki, and Ralf Lämmel. Swing to SWT and back: Patterns for API migration by wrapping. In *26th IEEE International Conference on Software Maintenance (ICSM 2010), September 12-18, 2010, Timisoara, Romania*, pages 1–10. IEEE Computer Society, 2010
- Thiago Tonelli Bartolomei, Krzysztof Czarnecki, Ralf Lämmel, and Tijs van der Storm. Study of an API migration for two XML apis. In Mark van den Brand, Dragan Gasevic, and Jeff Gray, editors, *Software Language Engineering, Second International Conference, SLE 2009, Denver, CO, USA, October 5-6, 2009, Revised Selected Papers*, volume 5969 of *Lecture Notes in Computer Science*, pages 42–61. Springer, 2009
- Ralf Lämmel. Transformations everywhere. *Sci. Comput. Program.*, 52:1–8, 2004
- Ralf Lämmel. Evolution of rule-based programs. *J. Log. Algebraic Methods Program.*, 60-61:141–193, 2004
- James R. Cordy, Ralf Lämmel, and Andreas Winter. 05161 executive summary - transformation techniques in software engineering. In James R. Cordy, Ralf Lämmel, and Andreas Winter, editors, *Transformation Techniques in Software Engineering, 17.-22. April 2005*, volume 05161 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005
- Ralf Lämmel. Evolution scenarios for rule-based implementations of language-based functionality. *Electron. Notes Theor. Comput. Sci.*, 128(1):61–79, 2005
- A. Steven Klusener, Ralf Lämmel, and Chris Verhoef. Architectural modifications to deployed software. *Sci. Comput. Program.*, 54(2-3):143–211, 2005
- Jan Kort and Ralf Lämmel. Parse-tree annotations meet re-engineering concerns. In *3rd IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2003), 26-27 September 2003, Amsterdam, The Netherlands*, page 161. IEEE Computer Society, 2003
- Ralf Lämmel. Towards generic refactoring. In Bernd Fischer and Eelco Visser, editors, *Proceedings of the 2002 ACM SIGPLAN Workshop on Rule-Based Programming, Pittsburgh, Pennsylvania, USA, 2002*, pages 15–28. ACM, 2002