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. Wiki-data is an open-source knowledge graph that stores structured data for Wikipedia with an ontology defined by Schema.org which structures Wiki-data. 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**

- Martin Leinberger, Philipp Seifer, Tjitze Rienstra, Ralf Lämmel, and Steffen Staab. Deciding SHACL shape containment through description logics


**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 apart 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
- 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,
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**

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-
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 devel-
oper 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 neg-
atively 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 en-
gagement, 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 particu-
larly 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 qual-
ity 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 refac-
torings with positive impact. Refactoring recommendation concerns code
quality in terms of maintainability. None of these approaches address re-
viewer engagement, except for the most clear-cut cases such as ‘apply this
refactoring’ or ‘accept this code fix’. Reviewer engagement recommen-
dation requires code-segment level advice and requires understanding the
intent of a change and also requires an explanation of the potential qual-
ity 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


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**


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, Eelco Visser, and Joost Visser. Strategic programming meets adaptive programming. In William G. Griswold and Mehmet Aksit, editors,


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


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

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


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


