IN PARTNERSHIP WITH:
Centrum Wiskunde & Informatica

Activity Report 2014

Project-Team ATEAMS

Analysis and Transformation based on rEliAble tool coMpositionS

RESEARCH CENTER
Lille - Nord Europe

THEME
Architecture, Languages and Compilation

Table of contents

Project-Team ATEAMS

Keywords: Programming Languages, Formal Methods, Domain-specific Languages, Software Engineering, Meta-modeling

Creation of the Project-Team: 2009 July 01.

1. Members

Research Scientists
Jurgen Vinju [Team leader, Centrum Wiskunde & Informatica, Professor]
Paul Klint [Centrum Wiskunde & Informatica, Professor]
Tijs Van Der Storm [Centrum Wiskunde & Informatica]
Jan Van Eijck [Centrum Wiskunde & Informatica]

Engineers
Maarten Dijkema [Centrum Wiskunde & Informatica]
Bert Lisser [Centrum Wiskunde & Informatica]

Administrative Assistant
Sandrine Meilen [Inria]

Others
Ali Afroozeh [Centrum Wiskunde & Informatica]
Pablo Inostroza Valdera [Centrum Wiskunde & Informatica]
Anastasia Izmaylova [Centrum Wiskunde & Informatica]
Davy Landman [Centrum Wiskunde & Informatica]
Ashim Shahi [Centrum Wiskunde & Informatica]
Michael Steindorfer [Centrum Wiskunde & Informatica]
Jouke Stoel [Centrum Wiskunde & Informatica]
Atze Van Der Ploeg [Centrum Wiskunde & Informatica]
Riemer Van Rozen [Centrum Wiskunde & Informatica]

2. Overall Objectives

2.1. Presentation

Software is very complex, and it seems to become more complex every year. Over the last decades, computer science has delivered various insights how to organize software better. Via structured programming, modules, objects, components and agents, these days software systems are more and more evolving into "systems of systems" that provide services to each other. Each system is large, uses incompatible — new, outdated or non-standard — technology and above all, exhibits failures.

It is becoming more and more urgent to analyze the properties of these complicated, heterogeneous and very large software systems and to refactor and transform them to make them simpler and to keep them up-to-date. With the plethora of different languages and technology platforms it is becoming very difficult and very expensive to construct tools to achieve this.

The main challenge of ATEAMS is to address this combination of a need for all kinds of novel analysis and transformation tools and the existence of the diversity of programming environments. We do this by investing in a virtual laboratory called "Rascal". It is a domain specific programming language for source code analysis, transformation and generation. Rascal is programming language parametric, such that it can be used to analyze, transform or generated source code in any language. By combining concepts from both program analysis and transformation into this language we can efficiently experiment with all kinds of tools and algorithms.

We now focus on three sub-problems. Firstly, we study software analysis: to extract information from existing software systems and to analyze it. The extracted information is vital to construct sound abstract models that can be used in further analysis. We apply these extraction techniques now to analyze (large bodies of) source code: finding bugs and finding the causes of software complexity.

Secondly, we study refactoring: to semi-automatically improve the quality of a software system without changing its behavior. Refactoring tools are a combination of analysis and transformations. Implementations of refactoring tools are complex and often broken. We study better ways of designing refactorings and we study ways to enable new (more advanced and useful) refactorings. We apply these refactorings now to isolate design choices in large software systems and compare systems that are equal except a single design choice.

Finally, we study code generation from domain specific languages (DSLs). Here we also find a combination of analysis and transformation. Designing, implementing and, very importantly, maintaining DSLs is costly. We focus on application areas such as Computational Auditing, Game Economies, and Core Banking to experiment with this subject. In Computational Auditing we are focusing on modeling interactive questionnaires. The Game economies domain involves modeling and verifying the dynamic behaviour of game play. Core banking requires the formal modeling of financial services and products.

3. Research Program

3.1. Research method

We are inspired by formal methods and logic to construct new tools for software analysis, transformation and generation. We try and proof the correctness of new algorithms using any means necessary.

Nevertheless we mainly focus on the study of existing (large) software artifacts to validate the effectiveness of new tools. We apply the scientific method. To (in)validate our hypothesis we often use detailed manual source code analysis, or we use software metrics, and we have started to use more human subjects (programmers).

Note that we maintain ties with the CWI spinoff “Software Improvement Group” which services most of the Dutch software industry and government and many European companies as well. This provides access to software systems and information about software systems that is valuable in our research.

3.2. Software analysis

This research focuses on source code; to analyze it, transform it and generate it. Each analysis or transformation begins with fact extraction. After that we may analyze specific software systems or large bodies of software systems. Our goal is to improve software systems by understanding and resolving the causes of software complexity. The approach is captured in the EASY acronym: Extract Analyze SYnthesize. The first step is to extract facts from source code. These facts are then enriched and refined in an analysis phase. Finally the result is synthesized in the form of transformed or generated source code, a metrics report, a visualization or some other output artifact.

The mother and father of fact extraction techniques are probably Lex, a scanner generator, and AWK, a language intended for fact extraction from textual records and report generation. Lex is intended to read a file character-by-character and produce output when certain regular expressions (for identifiers, floating point constants, keywords) are recognized. AWK reads its input line-by-line and regular expression matches are applied to each line to extract facts. User-defined actions (in particular print statements) can be associated with each successful match. This approach based on regular expressions is in wide use for solving many problems such as data collection, data mining, fact extraction, consistency checking, and system administration. This same approach is used in languages like Perl, Python, and Ruby. Murphy and Notkin have specialized the AWK-approach for the domain of fact extraction from source code. The key idea is to extend the expressivity of regular expressions by adding context information, in such a way that, for instance, the begin and end of a procedure declaration can be recognized. This approach has, for instance, been used for call graph extraction but becomes cumbersome when more complex context information has to be taken into account such as scope information, variable qualification, or nested language constructs. This suggests using grammar-based approaches as will be pursued in the proposed project. Another line of research is the explicit instrumentation of existing compilers with fact extraction capabilities. Examples are: the GNU C compiler GCC, the CPPX C++ compiler, and the Columbus C/C++ analysis framework. The Rigi system provides several fixed fact extractors for a number of languages. The extracted facts are represented as tuples (see below). The CodeSurfer source code analysis tool extracts a standard collection of facts that can be further analyzed with built-in tools or user-defined programs written in Scheme. In all these cases the programming language as well as the set of extracted facts are fixed thus limiting the range of problems that can be solved.

The approach we are exploring is the use of syntax-related program patterns for fact extraction. An early proposal for such a pattern-based approach consisted of extending a fixed base language (either C or PL/1 variant) with pattern matching primitives. In our own previous work on RScript we have already proposed a query algebra to express direct queries on the syntax tree. It also allows the querying of information that is attached to the syntax tree via annotations. A unifying view is to consider the syntax tree itself as “facts” and to represent it as a relation. This idea is already quite old. For instance, Linton proposes to represent all syntactic as well as semantic aspects of a program as relations and to use SQL to query them. Due to the lack of expressiveness of SQL (notably the lack of transitive closure) and the performance problems encountered, this approach has not seen wider use.

Parsing is a fundamental tool for fact extraction for source code. Our group has longstanding contributions in the field of Generalized LR parsing and Scannerless parsing. Such generalized parsing techniques enable generation of parsers for a wide range of existing (legacy) programming languages, which is highly relevant for experimental research and validation.

Extracted facts are often refined, enriched and queried in the analysis phase. We propose to use a relational formalization of the facts. That is, facts are represented as sets of tuples, which can then be queried using relational algebra operators (e.g., domain, transitive closure, projection, composition etc.). This relational representation facilitates dealing with graphs, which are commonly needed during program analysis, for instance when processing control-flow or data-flow graphs. The Rascal language integrates a relational sub-language by providing comprehensions over different kinds of data types, in combination with powerful pattern matching and built-in primitives for computing (transitive/reflexive) closures and fixpoint computations (equation solving).

3.2.1. Goals

The main goal is to replace labour-intensive manual programming of fact extractors by automatic generation based on concise and formal specification. There is a wide open scientific challenge here: to create a uniform and generic framework for fact extraction that is superior to current more ad-hoc approaches, yet flexible enough to be customized to the analysis case at hand. We expect to develop new ideas and techniques for generic (language-parametric) fact extraction from source code and other software artifacts.

Given the advances made in fact extraction we are starting to apply our techniques to observe source code and analyze it in detail.

3.3. Refactoring and Transformation

The second goal, to be able to safely refactor or transform source code can be realized in strong collaboration with extraction and analysis.

Software refactoring is usually understood as changing software with the purpose of increasing its readability and maintainability rather than changing its external behavior. Refactoring is an essential tool in all agile software engineering methodologies. Refactoring is usually supported by an interactive refactoring tool and consists of the following steps:

- Select a code fragment to refactor.
- Select a refactoring to apply to it.
- Optionally, provide extra parameter needed by the refactoring (e.g., a new name in a renaming).

The refactoring tool will now test whether the preconditions for the refactoring are satisfied. Note that this requires fact extraction from the source code. If this fails the user is informed. The refactoring tool shows the effects of the refactoring before effectuating them. This gives the user the opportunity to disable the refactoring in specific cases.The refactoring tool applies the refactoring for all enabled cases. Note that this implies a transformation of the source code. Some refactorings can be applied to any programming language (e.g., rename) and others are language specific (e.g., Pull Up Method). At http://www.refactoring.com an extensive list of refactorings can be found.

There is hardly any general and pragmatic theory for refactoring, since each refactoring requires different static analysis techniques to be able to check the preconditions. Full blown semantic specification of programming languages have turned out to be infeasible, let alone easily adaptable to small changes in language semantics. On the other hand, each refactoring is an instance of the extract, analyze and transform paradigm. Software transformation regards more general changes such as adding functionality and improving non-functional properties like performance and reliability. It also includes transformation from/to the same language (source-to-source translation) and transformation between different languages (conversion, translation). The underlying techniques for refactoring and transformation are mostly the same. We base our source code transformation techniques on the classical concept of term rewriting, or aspects thereof. It offers simple but powerful pattern matching and pattern construction features (list matching, AC Matching), and type-safe heterogenous data-structure traversal methods that are certainly applicable for source code transformation.

3.3.1. Goals

Our goal is to integrate the techniques from program transformation completely with relational queries. Refactoring and transformation form the Achilles Heel of any effort to change and improve software. Our innovation is in the strict language-parametric approach that may yield a library of generic analyses and transformations that can be reused across a wide range of programming and application languages. The challenge is to make this approach scale to large bodies of source code and rapid response times for precondition checking.

3.4. The Rascal Meta-programming language

The Rascal Domain-Specific Language for Source code analysis and Transformation is developed by ATeams. It is a language specifically designed for any kind of meta programming.

Meta programming is a large and diverse area both conceptually and technologically. There are plentiful libraries, tools and languages available but integrated facilities that combine both source code analysis and source code transformation are scarce. Both domains depend on a wide range of concepts such as grammars and parsing, abstract syntax trees, pattern matching, generalized tree traversal, constraint solving, type inference, high fidelity transformations, slicing, abstract interpretation, model checking, and abstract state machines. Examples of tools that implement some of these concepts are ANTLR, ASF+SDF, CodeSurfer, Crocopat, DMS, Grok, Stratego, TOM and TXL. These tools either specialize in analysis or in transformation, but not in both. As a result, combinations of analysis and transformation tools are used to get the job done. For instance, ASF+SDF relies on RScript for querying and TXL interfaces with databases or query tools. In other approaches, analysis and transformation are implemented from scratch, as done in the Eclipse JDT. The TOM tool adds transformation primitives to Java, such that libraries for analysis can be used directly. In either approach, the job of integrating analysis with transformation has to be done over and over again for each application and this requires a significant investment.

We propose a more radical solution by completely merging the set of concepts for analysis and transformation of source code into a single language called Rascal. This language covers the range of applications from pure analyses to pure transformations and everything in between. Our contribution does not consist of new concepts or language features *per se*, but rather the careful collaboration, integration and cross-fertilization of existing concepts and language features.

3.4.1. Goals

The goals of Rascal are: (a) to remove the cognitive and computational overhead of integrating analysis and transformation tools, (b) to provide a safe and interactive environment for constructing and experimenting with large and complicated source code analyses and transformations such as, for instance, needed for refactorings, and (c) to be easily understandable by a large group of computer programming experts. Rascal is not limited to one particular object programming language, but is generically applicable. Reusable, language specific, functionality is realized as libraries. As an end-result we envision Rascal to be a one-stop shop for source code analysis, transformation, generation and visualization.

3.5. Domain-specific Languages

Our final goal is centered around Domain-specific languages (DSLs), which are software languages tailored to a specific problem domain. DSLs can provide orders of magnitude improvement in terms of software quality and productivity. However, the implementation of DSLs is challenging and requires not only thorough knowledge of the problem domain (e.g., finance, digital forensics, insurance, auditing etc.), but also knowledge of language implementation (e.g., parsing, compilation, type checking etc.). Tools for language implementation have been around since the archetypical parser generator YACC. However, many of such tools are characterized by high learning curves, lack of integration of language implementation facets, and lead to implementations that are hard to maintain. This line of research focuses on two topics: improve the practice and experience of DSL implementation, and evaluate the success of DSLs in industrial practice.

Language workbenches [4] are integrated environments to facilitate the development of all aspects of DSLs. This includes IDE support (e.g., syntax coloring, outlining, reference resolving etc.) for the defined languages. Rascal can be seen as a language workbench that focuses on flexibility, programmability and modularity. DSL implementation is, in essence, an instance of source code analysis and transformation. As a result, Rascal's features for fact extraction, analysis, tree traversal and synthesis are an excellent fit for this area. An important aspect in this line of research is bringing the IDE closer to the source code. This will involve investigation of heterogeneous representations of source code, by integrating graphical, tabular or forms-based user interface elements. As a result, we propose Rascal as a feature-rich workbench for model-driven software development.

The second component of this research is concerned with evaluating DSLs in industrial contexts. This means that DSLs constructed using Rascal will be applied in real-life environments so that expected improvements in quality, performance, or productivity can be observed. We already have experience with this in the domain of digital forensics, computational auditing and games.

3.5.1. Goals

The goal of this research topic is to improve the practice of DSL-based software development through language design and tool support. A primary focus is to extend the IDE support provided by Rascal, and to facilitate incremental, and iterative design of DSLs. The latter is supported by new (meta-)language constructs for extending existing language implementations. This will require research into extensible programming and composition of compilers, interpreters and type checkers. Finally, a DSL is never an island: it will have to integrate with (third-party) source code, such as host language, libraries, runtime systems etc. This leads to the vision of multi-lingual programming environments [15].

4. New Software and Platforms

4.1. MicroMachinations

Participant: Riemer Van Rozen [correspondent].

Characterization: A-2-up3, SO-4, SM-2-up3, EM-3, SDL-3-up4, OC-DA-3-CD-3-MS-3-TPM-3.

WWW:

Objective: To create an integrated, live environment for modeling and evolving game economies. This will allow game designers to experiment with different strategies to realize game mechanics. The environment integrates with the SPIN model checker to prove properties (reachability, liveness). A runtime system for executing game economies allows MicroMachinations models to be embedded in actual games.

Users: Game designers

Impact: One of the important problems in game software development is the distance between game design and implementation in software. MicroMachinations has the potential to bridge this gap by providing live design tools that directly modify or create the desired software behaviors.

Competition: None.

Engineering: The front-end of MicroMachinations is built using the Rascal language workbench, including visualization, model checking, debugging and standard IDE features. The runtime library is implemented in C++ and will be evaluated in the context of industrial game design.

Publications: [11]

4.1.1. Novelties

- MMLib was finished to allow the execution of game economies directly within games. This supports "Live programming" of the behavior of games. The library has been used in the development of the real-life game "Johnny Jetstream", designed by IC3DMedia.

4.2. Naked Object Algebras

Participant: Tijs Van Der Storm [correspondent].

Characterization: A5, SO-4, SM-4, EM-4, SDL-4-up5, OC-DA-3-CD-3-MS-3-TPM-3.

WWW: https://github.com/cwi-swat/naked-object-algebras

Objective: Supporting modular and extensible language development.

Users: Programmers, language designers.

Impact: Object Algebras promise a new level of modularity and extensibility in the implementation of recursive data types. The NAO framework lifts this to the implementation of software languages, including the declarative declaration of concrete syntax.

Competition: Language prototyping tools.

Engineering: NAO consists of a few hundred lines of Java code. It has no external dependencies, except ANTLR for parsing.

Publications: [27], [33]

4.2.1. Novelties

- NAO has been used to develop an extensible variant of the QL questionnaire language [33].

4.3. Rascal

Participants: Paul Klint, Jurgen Vinju [correspondent], Tijs Van Der Storm, Pablo Inostroza Valdera, Davy Landman, Bert Lisser, Atze Van Der Ploeg, Vadim Zaytsev, Anastasia Izmaylova, Michael Steindorfer, Jouke Stoel, Ali Afroozeh, Ashim Shahi.

Characterization: A5, SO-4, SM-4, EM-4, SDL-4-up5, OC-DA-3-CD-3-MS-3-TPM-3.

WWW: http://www.rascal-mpl.org

Objective: Provide a completely integrated programming language parametric meta programming language for the construction of any kind of meta program for any kind of programming language: analysis, transformation, generation, visualization.

Users: Researchers in model driven engineering, programming languages, software engineering, software analysis, as well as practitioners that need specialized tools.

Impact: Rascal is making the mechanics of meta programming into a non-issue. We can now focus on the interesting details of the particular fact extraction, model, source analysis, domain analysis as opposed to being distracted by the engineering details. Simple things are easy in Rascal and complex things are manageable, due to the integration, the general type system and high-level programming features.

Competition: There is a plethora of meta programming toolboxes and frameworks available, ranging from plain parser generators to fully integrated environments. Rascal is distinguished because it is a programming language rather than a specification formalism and because it completely integrates different technical domains (syntax definition, term rewriting, relational calculus). For simple tools, Rascal competes with scripting languages and for complex tools it competes context-free general parser generators, with query engines based on relational calculus and with term rewriting and strategic programming languages.

Engineering: Rascal is about 100 kLOC of Java code, designed by a core team of three and with a team of around 8 PhD students and post-docs contributing to its design, implementation and maintenance. The goal is to work towards more bootstrapping and less Java code as the project continues.

Publications: [7], [6], [8], [5], [6]

4.3.1. Novelties

- Improvements of the language-parametric model to represent software projects (M3) [9].
- Performance improvements of the Rascal interpreter throughout.
- Further improvements to the compiler for Rascal, based on new language construct guarded coroutines.
- New language feature: keyword parameters. This will further allow simplificiation of the core language, as well as support better extensibility.
- Significant improvements to the Rascal static type checker.
- Further improvements to the new GLL parser (Iguana).
- Design of a new DSL for describing core banking infrastructure was started (ReBEL). Rascal was also used to develop a state machine DSL for use in embedded devices (Machino).

4.4. IDE Meta-tooling Platform

Participants: Jurgen Vinju [correspondent], Michael Steindorfer.

IMP, the IDE meta tooling platform is an Eclipse plugin developed mainly by the team of Robert M. Fuhrer at IBM TJ Watson Research institute. It is both an abstract layer for Eclipse, allowing rapid development of Eclipse based IDEs for programming languages, and a collection of meta programming tools for generating source code analysis and transformation tools.

Characterization: A5, SO-3, SM4-up5, EM-4, SDL-5, DA-2-CD-2-MS-2-TPM-2

WWW: https://github.com/impulse-org/

Objective: The IDE Meta Tooling Platform (IMP) provides a high-level abstraction over the Eclipse API such that programmers can extend Eclipse with new programming languages or domain specific languages in a few simple steps. IMP also provides a number of standard meta tools such as a parser generator and a domain specific language for formal specifications of configuration parameters.

Users: Designers and implementers of IDEs for programming languages and domain specific languages. Also, designers and implementers of meta programming tools.

Impact: IMP is a popular among meta programmers especially for it provides the right level of abstraction.

Competition: IMP competes with other Eclipse plugins for meta programming (such as Model Driven Engineering tools), but its API is more general and more flexible. IMP is a programmers framework rather than a set of generators.

Engineering: IMP is a long-lived project of many contributors, which is managed as an Eclipse incubation project at `eclipse.org`. Currently we are moving the project to Github to explore more different ways of collaboration.

Publications: [2] [29]

4.4.1. Novelties

- Significant performance improvements to the IMP program database. Performance is now better than equivalent data structure libraries in Scala and Clojure.

4.5. Ensō

Participant: Tijs Van Der Storm [correspondent].

Characterization: A5, SO-4, SM-3-up-4, EM-2-up-4, SDL-4, OC-DA-4-CD-4-MS-4-TPM-4

WWW: http://www.enso-lang.org

Objective: Together with Prof. Dr. William R. Cook of the University of Texas at Austin, and Alex Loh, Tijs van der Storm has been designing and implementing a new programming system, called Ensō. Ensō is theoretically sound and practical reformulation of model-based development. It is based on model-interpretation as opposed to model transformation and code generation. Currently, the system already supports models for schemas (data models), web applications, context-free grammars, diagram editors and security.

Users: All programmers.

Impact: Ensō has the potential to revolutionize the activity of programming. By looking at model driven engineering from a completely fresh perspective, with as key ingredients interpreters and partial evaluation, it may make higher level (domain level) program construction and maintenance as effective as normal programming.

Competition: Ensō competes as a programming paradigm with model driven engineering tools and generic programming and languages that provide syntax macros and language extensions.

Engineering: Ensō is a completely self-hosted system in 7000 lines of code.

Publications: [14], [16], [13]

5. New Results

5.1. Highlights of the Year

- Davy Landman, Jurgen Vinju received a Best paper award nomination, for their paper “Empirical analysis of the relationship between CC and SLOC in a large corpus of Java methods”(ICSM’14).

5.2. Cyclomatic complexity $\neq$ Lines of Code

It has long been believed that cyclomatic complexity of source code correlates linearly with lines of code (SLOC). After extensive study of a large corpus of Java source code, Davy Landman and Jurgen Vinju refuted this belief [34]. This provides a new landmark in how to assess and measure the quality of software. In short: cyclomatic complexity measures something different than lines of code.

5.3. Language-Parametric, Capture-Avoiding Program Transformation

Hygienic transformations are well-studied in the area of programming languages that feature (syntax) macros. For instance, in Scheme, macro expansion is guaranteed to not involuntarily capture existing bindings, or allow new bindings to be captured. Together with Sebastian Erdweg and Yi Dai, Tijs van der Storm designed a technique, “name-fix”, that can be used to ensure hygiene in arbitrary program transformations, even when source and target language are completely different [24].

5.4. Memory Efficient Hash Tries

The hash trie data structure is a common part in standard collection libraries of JVM programming languages such as Clojure and Scala. It enables fast immutable implementations of maps, sets, and vectors, but it requires considerably more memory than an equivalent array-based data structure. Michael Steindorfer designed a product family of hash tries to generate specialized Java source code [29]. A preliminary experiment on the implementation of sets and maps shows that this technique leads to a median decrease of 55% in memory footprint for maps and 78% for sets.

5.5. Reflection without Remorse

A series of list appends or monadic binds for many monads performs algorithmically worse when it is left-associated. Continuation-passing style (CPS) is well-known to cure this severe dependence of performance on the association pattern. The advantage of CPS dwindles or disappears if we have to examine or modify the intermediate result of a series of appends or binds, before continuing the series. Such examination is frequently needed, for example, to control search in non-determinism monads. Atze van der Ploeg (together with Oleg Kiselyov) developed an alternative approach that is just as general as CPS but more robust: it makes series of binds and other such operations efficient regardless of the association pattern [30]. This solution solves previously undocumented, severe performance problems in iteratees, LogicT transformers, free monads and extensible effects.

5.6. General Parser Combinators

Parser combinators are a well-known approach to parsing where grammars are represented using (higher-order) functions. Unfortunately, parser combinators are commonly implemented using recursive descent parsing as the underlying algorithm. As a result, most parser combinators frameworks do not support left-recursive rules, and may exhibit exponential runtime performance due to backtracking. Anastasia Izmaylova and Ali Afroozeh developed “general parser combinators” (GPC) which do not suffer from these problems: all context-free grammars are supported (even ambiguous ones) and performance is worst-case cubic. As result, GPC combines the expressiveness and performance guarantees of general parsing algorithms like GLL and GLR with the flexibility and extensibility of parser combinators.

6. Bilateral Contracts and Grants with Industry

6.1. Bilateral Contracts with Industry

- ING co-financed one PhD position in the context of CWI public-private collaboration program. The goal is to apply domain-specific language technology to revitalize core banking infrastructure.
- AimValley won the CWI research voucher for developing a DSL for state machines in the context of embedded devices. Davy Landman performed the research and development.

7. Partnerships and Cooperations

7.1. National Initiatives

7.1.1. Master Software Engineering

ATEAMS is the core partner in the Master Software Engineering at Universiteit van Amsterdam. This master is a collaboration between SWAT/ATEAMS, Universiteit van Amsterdam, Vrije Universiteit and Hogeschool van Amsterdam.

7.1.2. Early Quality Assurance in Software Production

The EQUA project is a collaboration among Hogeschool van Amsterdam (main partner) Centrum Wiskunde & Informatica (CWI), Technisch Universiteit Delft, Laboratory for Quality of Software (LaQuSo), Info Support, Software Improvement Group (SIG), and Fontys Hogeschool Eindhoven.

7.1.3. Next Generation Auditing: Data-assurance as a service

This is a collaboration between Centrum Wiskunde & Informatic (CWI) PriceWaterhouseCoopers (PWC), Belastingdienst (National Tax Office), and Computational Auditing, is to enable research in the field of computational auditing.

7.2. European Initiatives

7.2.1. FP7 & H2020 Projects

Program: FP7 STREP

Project acronym: OSSMeter

Project title: Automated Measurement and Analysis of Open Source Software

Duration: 30 months (2012-10-01 – 2015-03-31)

Coordinator: Scott Hansen

Other partners: CWI, SOFTEAM (France), Tecnalia Research and Innovation (Spain), The Open Group (Belgium), University of L'Aquila (Italy), UNINOVA (Portugal), National Centre for Text Mining University of Manchester (UK), University of York (UK), Unparallel Innovation (Portugal).

7.3. International Research Visitors

7.3.1. Visits of International Scientists

7.3.1.1. Internships

- Cleverton Hentz, PhD Candidate at the Department of Informatics and Applied Mathematics (Dimap) at Federal University of Rio Grande do Norte (UFRN).

8. Dissemination

8.1. Promoting Scientific Activities

8.1.1. Scientific events organisation

8.1.1.1. General chair, scientific chair

- Jurgen Vinju: General Chair, The 7th International Conference on Software Language Engineering (SLE), Steering committee member SLE.

8.1.1.2. Member of the organizing committee

- Davy Landman: Web chair, The 7th International Conference on Software Language Engineering (SLE).
- Tijs van der Storm: Social Media Chair, The 7th International Conference on Software Language Engineering (SLE); co-organizer Belgium Netherlands Seminar on Software Evolution (BENEVOL'14), co-organizer Software Development Automation'14 (SDA'14).

8.1.2. Scientific events selection

8.1.2.1. Member of the conference program committee

- Paul Klint: WCRE/CSMR ERA 2014.
- Tijs van der Storm: SLE'14, DSLDI'14, SEMS'14, FTJP'14, TTC'14.
- Jurgen Vinju: WCRE/CSMR Tool track.

8.1.2.2. Reviewer

- Tijs van der Storm: PLDI'14.

8.1.3. Journal

8.1.3.1. Member of the editorial board

- Jan van Eijck: Editor of the Journal of Logics and their Applications.
- Paul Klint: Editor Science of Computer Programming, editor Springer LNCS Series on Services Science.
- Jurgen Vinju: co-editor ERCIM news, special issue on Software Quality.

8.1.3.2. Reviewer

- Jan van Eijck: Reviewer for ESSLLI, Journal of Semantics, Journal of Logic and Computation, Fundamenta Informaticae, Synthese, Journal of Philosophical Logic, Journal of Logic, Language and Information, Cambridge University Press.
- Tijs van der Storm: Journal of Software & Systems Modeling, Science of Computer Programming, Computer Languages, Computer Languages, Systems and Structures, Automated Softare Engineering.

8.2. Teaching - Supervision - Juries

8.2.1. Teaching

Master : Jan van Eijck, “Software Testing”, 6 EC, Master Programme Software Engineering, Universiteit van Amsterdam, The Netherlands.

Master: Jan van Eijck, “Functional Specification of Algorithms”, 6 EC, Master Programme Logic, Universiteit van Amsterdam, The Netherlands.

Master : Tijs van der Storm, “Software Construction”, 6 EC, Universiteit van Amsterdam, The Netherlands.

8.2.2. Supervision

PhD : Jeroen van den Bos, “Gathering Evidence: Model-Driven Software Engineering in Automated Digital Forensics”, Universiteit van Amsterdam, January, 9th, 2014, Paul Klint, Tijs van der Storm.

PhD in progress : Ali Afroozeh, “Advances in GLL Parsing” , 2012, Paul Klint, Jurgen Vinju.

PhD in progress : Pablo Inostroza Valdera, “Modularity in DSL development”, 2013, Paul Klint, Tijs van der Storm.

PhD in progress : Anastasia Izmaylova, “Language Parametric Refactoring”, 2011, Paul Klint, Jurgen Vinju.

PhD in progress : Atze van der Ploeg, “Efficient Abstractions for Visualization and Interaction”, 2011, Paul Klint, Tijs van der Storm.

PhD in progress : Riemer van Rozen, “Model-Driven Game Development”, 2013, Paul Klint, Tijs van der Storm.

PhD in progress : Michael Steindorfer, “Efficient Data Structures for Meta Programming”, 2012, Paul Klint, Jurgen Vinju.

PhD in progress : Jouke Stoel, “Model Driven Infrastructure for Core Banking”, 2014, Jurgen Vinju, Tijs van der Storm.

8.2.3. Juries

Jurgen Vinju acted as reading committee member of the PhD of Maartje de Jonge, at TU Delft.

8.3. Popularization

Paul Klint:

- De Softwarerevolutie, Valedictory lecture.
- Nemo, Hoe ontstond de eerste computer?
- Nemo/Klokhuis, Hoe ontstond de eerste computer?
- BYOM: Bring Your Own Metrics (EQUA Symposium).
- The Revenge of the Coroutines (SEN Symposium).

Tijs van der Storm:

- Who’s afraid of Object Algebras?, Joy of Coding 2014.
- Hack your DSL with Rascal, CodeGeneration 2014.
- The Rascal Language Workbench, NSPyre, 2014.
- I am plain text, – resistance is futile, Sioux, 2014.
- Rascal: functional programming for source code analysis and transformation, guest lecture at Hogeschool van Arnhem en Nijmegen (HAN).

Jurgen Vinju:

- De allereerste computerprogrammeur Ada Lovelace (1815 - 1852), Kennis van NU (radio appearance).
- Complexe Software, Eindhovens Dagblad.
- De eerste programmeur, VPRO Gids.

Jan van Eijck is member of the Advisory Board (‘Raad van Advies’) of the Artificial Intelligence Curriculum, University of Groningen (since Summer 2013). Paul Klint acts as treasurer of the EAPLS and was directory of the Master Software Engineering at Universiteit van Amsterdam (UvA) until September 1, 2014. He is also board member of the Instituut voor Programmatuur en Algoritmiek (IPA). Tijs van der Storm is head of the internship committee at CWI, co-organizer of the CWI Scientific Meeting and secretary of the CWI works council.

9. Bibliography

Major publications by the team in recent years

[1] B. BASTEN. *Tracking Down the Origins of Ambiguity in Context-Free Grammars*, in "Seventh International Colloquium on Theoretical Aspects of Computing (ICTAC 2010)", A. CAVALCANTI, D. DEHARBE, M.-C. GAUDEL, J. WOODCOCK (editors), Springer, September 2010, vol. 6255, pp. 76-90

[2] P. CHARLES, R. M. FUHRER, S. M. SUTTON JR, E. DUESTERWALD, J. VINJU. *Accelerating the Creation of Customized, Language-Specific IDEs in Eclipse*, in "Proceedings of the 24th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2009", S. ARORA, G. T. LEAVENS (editors), 2009

[3] JAN VAN. EIJCK, C. UNGER. *Computational Semantics with Functional Programming*, Cambridge University Press, September 2010

[4] S. ERDWEG, T. STORM VAN DER, M. VOELTER, M. BOERSMA, R. BOSMAN, W. R. COOK, A. GERRITSEN, A. HULSHOUT, S. KELLY, A. LOH, G. KONAT, P. J. MOLINA, M. PATATNIK, R. POHJONEN, E. SCHINDLER, K. SCHINDLER, R. SOLMI, V. VERGU, K. B. VAN DER VLIST, G. WACHSMUTH, J. M. VAN DER WONING. *The State Of The Art In Language Workbenches. Conclusions From The Language Workbench Challenge*, in "Proceedings of the International Conference on Software Language Engineering (SLE, 2013)", Indianapolis, USA, 2013, http://hal.inria.fr/hal-00923386

[5] M. HILLS, P. KLINT, J. VINJU. *Meta-Language Support For Type-Safe Access To External Resources*, in "Pre-Proceedings of the 5th International Conference on Software Language Engineering", Dresden, Netherlands, K. CZARNECKI, G. HEDIN (editors), Fakultät Informatik, Technische Universität Dresden, 2012, pp. 370 - 389, http://hal.inria.fr/hal-00756878

[6] M. HILLS, P. KLINT, J. VINJU. *Program Analysis Scenarios In Rascal*, in "Proceedings of the International Workshop on Rewriting Logic and its Applications (WRLA, 2012)", Talinn, Estonia, F. DURÁN (editor), Springer, 2012, vol. 7571, pp. 10 - 30, An invited paper for WRLA 2012, describing our work on program analysis and comparing our approach to approaches based on rewriting logic semantics, http://hal.inria.fr/hal-00756880

[7] M. HILLS, P. KLINT, J. VINJU. *Scripting A Refactoring With Rascal And Eclipse*, in "Proceedings of the 5th Workshop on Refactoring Tools 2012", Rapperswil, Switzerland, P. SOMMERLAD (editor), ACM, 2012, pp. 40 - 49, http://hal.inria.fr/hal-00756879

[8] M. HILLS, P. KLINT, T. VAN DER STORM, J. VINJU. *A One-Stop Shop For Software Evolution Tool Construction*, in "ERCIM News", 2012, n^{o} 88, pp. 11 - 12, http://hal.inria.fr/hal-00756876

[9] A. IZMAYLOVA, P. KLINT, A. SHAHI, J. VINJU. *M3: An Open Model For Measuring Code Artifacts*, 2013, n^{o} arXiv-1312.1188, pp. 1-2, https://hal.inria.fr/hal-00923379

[10] P. KLINT, T. VAN DER STORM, J. VINJU. *EASY Meta-programming with Rascal*, in "Generative and Transformational Techniques in Software Engineering III", J. FERNANDES, R. LÄMMEL, J. VISSER, J. SARAIVA (editors), Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2011, vol. 6491, pp. 222-289, http://dx.doi.org/10.1007/978-3-642-18023-1_6

[11] P. KLINT, R. VAN ROZEN. *Micro-Machinations: A DSL For Game Economies*, in "Proceedings of the International Conference on Software Language Engineering (SLE, 2013)", Unknown, M. ERWIG, R. F. PAIGE, E. VAN WYK (editors), Lecture Notes in Computer Science, Springer, 2013, vol. 8225, pp. 36 - 55, https://hal.inria.fr/hal-00923383

[12] P. KLINT, T. VAN DER STORM, J. VINJU. *RASCAL: A Domain Specific Language for Source Code Analysis and Manipulation*, in "IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'09)", Los Alamitos, CA, USA, 2009, pp. 168-177, http://doi.ieeecomputersociety.org/10.1109/SCAM.2009.28

[13] A. LOH, T. VAN DER STORM, W. R. COOK. *Managed Data: Modular Strategies For Data Abstraction*, in "Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software 2012", Tucson, United States, ACM, 2012, pp. 179 - 194, http://hal.inria.fr/hal-00756886

[14] B. C. D. S. OLIVEIRA, T. VAN DER STORM, A. LOH, W. R. COOK. *Feature-Oriented Programming With Object Algebras*, in "Proceedings of the European Conference on Object-Oriented Programming (ECOOP)", 2013, http://hal.inria.fr/hal-00923387

[15] T. VAN DER STORM, J. VINJU. *Towards Multilingual Programming Environments*, in "Science of Computer Programming", 2013, https://hal.inria.fr/hal-00923385

[16] T. VAN DER STORM, W. R. COOK, A. LOH. *Object Grammars: Compositional & Bidirectional Mapping Between Text and Graphs*, in "Software Language Engineering", Dresden, Germany, K. CZARNECKI, G. HEDIN (editors), September 2012, http://hal.inria.fr/hal-00758627

[17] J. VINJU, M. W. GODFREY. *What does control flow really look like? Eyeballing the Cyclomatic Complexity Metric*, in "Ninth IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM'12)", IEEE Computer Society, 2012

[18] J. VAN DEN BOS, T. VAN DER STORM. *Bringing Domain-Specific Languages to Digital Forensics*, in "Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011", Honolulu, United States, ACM, 2011, pp. 671-680, http://hal.inria.fr/hal-00644687/en

[19] J. VAN DEN BOS, T. VAN DER STORM. *Domain-Specific Languages For Better Forensic Software*, in "ERCIM News", 2012, vol. 2012, n^o 90, http://hal.inria.fr/hal-00756885

[20] J. VAN DEN BOS, T. VAN DER STORM. *Domain-Specific Optimization In Digital Forensics*, in "Proceedings of the International Conference on Model Transformation (ICMT, 2012)", Prague, Czech Republic, Z. HU, J. DE LARA (editors), Springer, 2012, vol. 7307, pp. 121 - 136, http://hal.inria.fr/hal-00756891

Publications of the year

Articles in International Peer-Reviewed Journals

[21] M. BRUNTINK, J. VINJU. *Looking Towards A Future Where Software Is Controlled By The Public and Not The Other Way Round*, in "ERCIM News", 2014, vol. 99, 1 p. , https://hal.inria.fr/hal-01110831

[22] A. CLEVE, J. VINJU. *Software Quality*, in "ERCIM News", 2014, 1 p. , https://hal.inria.fr/hal-01110830

[23] T. VAN DER STORM, W. J. COOK, A. LOH. *The design and implementation of Object Grammars*, in "Science of Computer Programming", 2014, vol. 96, pp. 460 - 487 [*DOI :* 10.1016/J.SCICO.2014.02.023], https://hal.inria.fr/hal-01110829

International Conferences with Proceedings

[24] S. ERDWEG, T. VAN DER STORM, Y. DAI. *Capture-Avoiding and Hygienic Program Transformations*, in "ECOOP 2014 - Proceedings of European Conference on Object-Oriented Programming", Uppsala, Sweden, Springer, July 2014, pp. 489 - 514, https://hal.inria.fr/hal-01110895

[25] M. HILLS, P. KLINT, J. VINJU. *Static, lightweight includes resolution for PHP*, in "ASE 29 Proceedings of International Conference on Automated Software Engineering 2014", Vasteras, Sweden, September 2014, pp. 503 - 514 [*DOI :* 10.1145/2642937.2643017], https://hal.inria.fr/hal-01110903

[26] P. INOSTROZA, T. VAN DER STORM, S. ERDWEG. *Tracing Program Transformations with String Origins*, in "ICMT - Proceedings of International Conference on Model Transformation", York, United Kingdom, 2014, pp. 154 - 169 [*DOI :* 10.1007/978-3-319-08789-4_12], https://hal.inria.fr/hal-01110885

[27] P. INOSTROZA, T. VAN DER STORM. *Evolving Languages with Object Algebras*, in "BENEVOL 2014 - Proceedings of the Belgian-Netherlands Evoluation Workshop", Amsterdam, Netherlands, 2014, 2 p. , https://hal.inria.fr/hal-01110869

[28] P. INOSTROZA, T. VAN DER STORM. *The TTC 2014 Movie Database Case: Rascal Solution* *, in "Transformation Tool Contest", L'Aquila, Italy, Proceedings of Transformation Tool Contest 2014 (TTC'14), CEUR, 2014, pp. 155 - 159, https://hal.inria.fr/hal-01110851

[29] M. J. STEINDORFER, J. VINJU. *Code Specialization for Memory Efficient Hash Tries (Short Paper)*, in "GPCE - Proceedings of ACM International Conference on Generative Programming and Component Engineering 2014", Vasteras, Sweden, ACM, September 2014, 4 p. , https://hal.inria.fr/hal-01111004

[30] A. VAN DER PLOEG, O. KISELYOV. *Reflection without Remorse: Revealing a hidden sequence to speed up monadic reflection*, in "Haskell '14 - Proceedings of the 2014 ACM SIGPLAN symposium on Haskell", Gothenburg, Sweden, ACM, September 2014, pp. 133-144 [*DOI :* 10.1145/2633357.2633360], https://hal.inria.fr/hal-01110936

[31] R. VAN ROZEN, J. DORMANS. *Adapting Game Mechanics with Micro-Machinations*, in "Foundations of Digital Games", Aboard Royal Caribbean Liberty of the Seas, sailing from Ford Lauderdale, Florida, United States, Proceedings of the 9th International Conference on the Foundations of Digital Games, Society for the Advancement of the Science of Digital Games, April 2014, https://hal.inria.fr/hal-01110847

[32] R. VAN ROZEN, T. VAN DER STORM. *Model Differencing for Textual DSLs*, in "BENEVOL 2014 - Proceedings of the Belgian-Netherlands Evoluation Workshop", Amsterdam, Netherlands, 2014, https://hal.inria.fr/hal-01110856

Conferences without Proceedings

[33] M. GOUSETI, C. PETERS, T. VAN DER STORM. *Extensible language implementation with object algebras (short paper)*, in "GPE 2014 - International Conference on Generative Programming: Concepts and Experiences", Västerås, Sweden, Proceedings of the International Conference on Generative Programming: Concepts and Experiences (GPCE, 2014), 2014, pp. 25 - 28 [*DOI :* 10.1145/2658761.2658765], https://hal.inria.fr/hal-01110872

[34] D. LANDMAN, A. SEREBRENIK, J. VINJU. *Empirical analysis of the relationship between CC and SLOC in a large corpus of Java methods*, in "IEEE International Conference on Software Maintenance and Evolution 2014", Victoria, Canada, L. M. F. MOONEN, L. POLLOCK (editors), IEEE Computer Society, September 2014, pp. 221 - 230, https://hal.inria.fr/hal-01110843

Books or Proceedings Editing

[35] B. COMBEMALE, D. J. PEARCE, O. BARAIS, J. J. VINJU (editors). *Software Language Engineering*, SpringerVästerås, Sweden, 2014, n^{o} 8706, 353 p. , https://hal.inria.fr/hal-01110914