Open and Original Problems in Software Language Engineering 2015 Workshop Report

Anya Helene Bagge
Bergen Language Design Laboratory, University of Bergen, Norway
anya@ii.uib.no

Vadim Zaytsev
Institute of Informatics, Universiteit van Amsterdam, The Netherlands
vadim@grammarware.net

ABSTRACT

OOPSLE is a workshop co-located with a re-engineering conference and serving as a venue for software language engineers to meet outside the SLE conference to discuss either long-standing problems that remain unresolved for years or decades, or oft-avoided problems that everyone is so used to work around that they stop noticing them at all. In 2015, it ran for the third time. The list of topics discussed at the workshop, included transformation in the presence of Boolean grammars, natural language software interfaces, formally supported model management, community-aware language design, domain-specific language design choices and uncertainty-aware development. A report such as this one was requested by the participants, but more condensed and general information can be found on our official webpage at http://oopsle.github.io.

1. INTRODUCTION

The third international workshop on Open and Original Problems in Software Language Engineering (OOPSLE'15), held on 6 March 2015 at SANER, followed the first two editions held at WCRE 2013 in Koblenz [4] and at CSMR-WCRE 2014 in Antwerp [6]. The main focus of the workshop remained in identifying and formulating challenges in the software language engineering field — the challenges that could be addressed later at venues like SLE, MoDELS, SANER, ICSME, OOPSLA, ECOOP, PLDI, POPL and others. The workshop was highly interactive, all speakers were allocated generous time slots (45 minutes) and advised to speak for one third of the time and reserve the rest for discussion. In this report, requested by the participants at the end of the workshop, we try to recall the highlights of the day and document brief summaries of sessions.

Quoting the description distributed in the call for papers [4, 6]:

> The field covered by the workshop, revolves around "software languages" — all kinds of artificial languages used in software development: for programming, markup, pretty-printing, modelling, data description, formal specification, evolution, etc. Software language engineering is a relatively new research domain of systematic, disciplined and measurable approaches of development, evolution and maintenance of such languages. Many concerns of software language engineering are acknowledged by reverse and forward software engineers: robust parsing of language cocktails, fact extraction from heterogeneous codebases, tool interfaces and interoperability, renovation of legacy systems, static and dynamic code analysis, language feature usage analysis, mining repositories and chrestomathies, library versioning and wrapping, etc.

There are research fields that have a list of known open problems already, for example:

- 23 Hilbert's problems [21]
- The POPLmark Challenge [33]
- Open problems in Boolean grammars [31]

The workshop was predominantly motivated by the lack of any comparable list for the field of software language engineering. We wanted to expose expertise hidden in the community and elicit such problems in an explicit list. Another side of the workshop was the hope of formulating open challenges related to software language engineering. There are many contests, challenges, competitions and benchmarks around:

- The benchmark collection of the POPLmark Challenge [33]
- LDTA Tool Challenge held at that workshop in 2011 [27]
- Language Workbench Challenge [17] held at CodeGeneration yearly since 2011
- Transformation Tool Contest held seven times since 2007 [36]
- Rewrite Engines Competition held three times in 2006, 2008 and 2010 at WRLA [13]
- PLT Games held monthly in 2013 [29]

The full list of topics advised for workshop participants can be found on the website: http://oopsle.github.io.

2. FORMAT

In 2013, we started this initiative as a small scale endeavour with three sessions: the academic keynote given by Prof. Dr. Ralf Lämmel (Universität Koblenz-Landau); the industrial keynote by Dr. Darius Blasband (RainCode); and a freeform discussion session attended by approximately ten people interested in the research area.

The format evolved in 2014: we advertised the workshop more broadly and received a number of submissions through EasyChair. All of them were accepted for presentation. The authors of the submissions received review-like feedback from the organizers, some more detailed than others, and were asked to present their contributions in a highly interactive fashion. Again, we had two keynotes: the academic keynote was given by Prof. Dr. Sibylle Schupp (Technische Universität Hamburg-Harburg); the industrial one by Dr. Ira Baxter (Semantic Designs). The extended abstracts of all submissions, including the keynotes, were bundled in a pre-proceedings volume distributed among the workshop participants, as well as online through social media [5]. At the end of the workshop we committed to eventually preparing a post-proceedings special issue of the EC-EASST journal, possibly combined with the contributions of OOPSLE 2013.

In 2015 we opted for a different format by strategically encouraging researchers from a broad selection of subdomains of software language engineering to become workshop presenters. We also moved to a one keynote format by inviting Prof. Dr. James R. Cordy (Queen's University), who is known for his work in academia as well as in the industry. This allowed us to save time in the programme and leave more space for discussions. The final list contained the following presenters:

- James R. Cordy[1] (grammarware engineering, program transformation)
- Emily Hill[2] (social software engineering, natural language processing)
- Zinovy Diskin[3] (model-driven engineering, formal methods)
- Alexander Serebrenik[4] (collaborative software engineering, human aspects)
- Mark Hills[5] (rewriting logic, software analytics)
- Naoyasu Ubayashi[6] (aspect-oriented software development, software quality)

3. OOPSLE 2015

On the following pages we recall the presentations and discussions, one subsection per presenting participant.

3.1 Source Transformation on Boolean Grammars: Advantages and Challenges

In his keynote, James Cordy presented open problems in his work with Andrew Stevenson on transformation in the presence of Boolean grammars [37]. Boolean grammars [31] are an extension of context-free grammars that include a conjunction (the "and" operator) and negation (frequently in the form of negative conjunction, the "and not" operator) in addition to commonly used disjunction (the choice operator "or"). For example, a rule `NonNum ::= Expr &! Number` defines a nonterminal `NonNum` which must be an `Expr` and must not be a `Number` at the same time. The class of Boolean grammars is larger than that of context-free grammars, allowing for languages such as

```
ABC ::= AB c* & a* BC
AB ::= a AB b | ε
BC ::= b BC c | ε
```

[1]http://research.cs.queensu.ca/home/cordy
[2]http://www.drew.edu/mcs/faculty
[3]http://gsd.uwaterloo.ca/zdiskin
[4]http://www.win.tue.nl/~aserebre
[5]http://www.cs.ecu.edu/hillsma
[6]http://posl.ait.kyushu-u.ac.jp/~ubayashi

which defines a language $\{a^n b^n c^n\}$ commonly known to not belong to the context-free class.

Boolean grammars are not (yet) wildly popular, but has received some attention in the programming language community recognising their usefulness in specifying processes that were impossible to describe grammatically before them [41]. In the source-to-source transformation framework TXL [8], Boolean grammars are used to create "views" — additional ways of interpreting the input that are particularly suited to a specific transformation. For example, for transformations, it may be easier to see the language above as a sequence of `a`'s, `b`'s and `c`'s, without any restriction:

```
ABC ::= AB c* & a* BC & A B C
A ::= a*
B ::= b*
C ::= c*
```

The advantage lies in the fact that from this point the first two conjuncts are used for recognising the required language, but the last one can be used for expressing transformations (e.g., `aaA bbB ccC => A B C`). This enables context sensitivity without the usual hassle of propagating artificial constructs in different ways, seen in all attempts to break free from CFGs in concrete syntax processing so far. Essentially, the most convenient one of the views is used as the abstract syntax or layout-agnostic concrete syntax [42] for comfortably expressing program transformations.

The challenge, however, lies in formal semantics of transformations of conjunctive clauses: after transforming one view, we can end up with something not well-formed according to the other views, and the resolution is unclear. For instance, the rule shown above can be broken in three rules `aaA => A`, `bbB => B` and `ccC => C`; each subrule is correct, and their composition yields a well-formed tree, but the intermediate results are not well-formed. So far TXL was built on the assumption that every transformation preserves well-formedness, and behaving differently when transforming with Boolean grammars would be unexpected and undesirable for the users.

Neither Alexander Okhotin, the inventor of Boolean grammars, nor James Cordy and his TXL colleagues, have been able to present a general solution so far. In simple cases there is no problem and simple heuristics can help, but in general the offending transformation can happen several steps before it gets detected and a later application of a rule can invalidate a parse of a much bigger tree than it is scoped for. This is to be expected, though — when we go beyond the class of context-free grammars, it is hard to require transformations to be fully context-free either.

The central questions raised in James Cordy's keynote are:

- How to integrate transformation with Boolean grammars in TXL in such a way that TXL's well-formedness properties hold, without being so restrictive that practical applications are hampered?
- How should we treat composite transformations that ultimately have a valid result, but not until all transformations are complete?
- What happens when different rules transform the same parse tree in different views? How should the result be interpreted?

A number of solutions were sketched by the speaker, none of them perfect:

- Abandoning the well-formedness principle, which would be un-TXL-ish.
- Turn conjuncts into disjuncts upon transformation: dropping support for multiple views per program, and adding little to what TXL could already do.
- Require patterns that include all views (proposed by Vadim Zaytsev) — in general this defeats the purpose of having views in TXL, and optimised implementations remain to be investigated.
- Do full reparsing after transformation (with some optimisation of rule sets), fail the rule if reparsing fails, and let the user worry about extending the grammar to accommodate intermediate results.
- Work only on a single normalised representation, and transform all rules to work on this representation. This solution emerged in a discussion with the audience and would be rather complicated to achieve for TXL which works solely on concrete syntax trees, but may work in an abstract syntax tree setting.
- Give up on Boolean grammars.

Although open problems remain in the use of Boolean grammars in TXL, a number of common usage scenarios is already supported, including viewing a complicated expression grammar with precedence as a simplified grammar with one nonterminal. Also, while the grammar formalism is extended with Boolean operators, there is no change in performance or semantics for users who do not use these features. In the end, it remains to be seen how TXL users will make use of Boolean grammars.

3.2 Engineering Iron Man's J.A.R.V.I.S.

Emily Hill's work mostly concerned natural language processing techniques used for software engineering and reverse engineering. This particular talk was inspired by the *Iron Man* [20] movie where the titular hero talks to his computer assistant in casual colloquial phrases using wording like *"I'm thinking icing is the probable factor"* and *"thrill me!"*, potentially ambiguous even to a human speaker. According to Emily Hill, we should strive toward computer interfaces that require similar lack of strain in formulating a known solution into executable steps, as well as a comparable luxury in abstract sketching instead of precise definitions that can be refined automatically whenever needed.

This point of view was in considerable discord with the trend of the last decades of programming in strict languages with fixed unambiguous semantics (or at least attempting to reach that ideal). However, she did identify related endeavours in code search, code completion, method invocation search, naturalistic programming languages [25], informal software representations [2], people-specific languages [34], keyword programming [28] and representing natural language phrasal concepts [22].

Transforming among people-specific or even technology-specific languages was easily identified as an open problem in software language engineering. Transforming high level abstract and ambiguous natural language description of a problem into executable code for a solution was deemed to be the other side of the same challenge, also stopping J.A.R.V.I.S.-like systems from appearing. However, there were several fundamental issues named that need to be addressed to even enable the first steps toward these goals: semantic ambiguity (human languages are inherently ambiguous and designed as such; computer languages are the opposite); productivity (can programmers be as effective while operating such a natural language interface as they can get with a keyboard and a strict software language?); internationalisation (are all natural language programming interfaces equally usable?). Possible solutions included pulling out actionable verb phrases from the input and searching for similarly labelled code examples like RANDOOP; constrained natural languages like Pegasus; training programming languages with people-specific DSLs; and in general investigating correspondence between high level natural languages and programming/modelling languages.

A fundamental obstacle was identified by Zinovy Diskin to be promptly supported by Valerio Cosentino: sometimes formalising, modelling or programming are done in order to understand the problem better, so avoiding this step in such situations will be harmful. For the most part, the concluding question about the feasibility for natural language interfaces and the reasons for the lack of progress in the programming language community has been answered, but the problem remained firmly standing.

3.3 Sociology of Model Management: Why Model Mappings are to Be the First-class Citizens in MDE

Zinovy Diskin is a prominent researcher in the field of model-driven engineering, specialising in the formal side of it. Patterns found in category theory in the form of commuting diagrams (for pushouts, pullbacks and colimits in general), are directly applicable to mathematical modelling of structures appearing in everyday MDE practice: model merging, transformation, synchronisation, well-formedness and consistency management can all be seen as executions of categorical specifications [10]. Then, if an operation can be defined diagrammatically, composition of them amounts to trivial tiling of commuting squares [9].

The main focus of Zinovy Diskin at our workshop was not a definition of a new open problem, but rather a call for attention to the fact that many problems from different domains have similarly-looking solutions if you formalise them in a certain uniform way (i.e., in categorical terms). The realities of MDE, presented based on a recent survey involving industrial users [30], show a lot of problems involving tool adoption and usability. Additionally, the MDE pipeline is not streamlined and has a lot of "turbulence" [11] because the real world knowledge gets tangled with transformations and code generation, without sufficiently strong support for declaring and leveraging bidirectional transformations.

The solution, according to Diskin, lies in developing a sound theory of model management. This can help change the fact that metamodels often do not correspond to intuition and are in general a sloppy way to define the structure of models (often omitting constraints, having pieces of missing structure and allowing illegally structured instances). The case discussed in detail was *model merge*: without mappings the model of merging is insufficient (or wrong), so viewing it categorically is natural; one only needs to define three components for such a merge. These components are: the equality or correspondence relation that shows which nodes can be joined; the copying or mapping of nodes that cannot be glued together; and a coverage claim to ensure that all the elements of the original models were preserved. The great theorem of set merge states that for any sets A and B and correspondence span R, there is only one merge X, up to isomorphism. Diskin concluded by a Church-Turing-like thesis that any intuitive definition of a set merge amounts to the formal operation he introduced. This obviously includes popular tools for merging/weaving of models: UML Package Merge, Kompose, ADORE, TreMer+ and others; and in general has been known to work on richer data structures like graphs, attributed graphs, Petri nets, and models for a given metamodel and equational constraints.

3.4 Software Languages: Designing for Humans

Alexander Serebrenik is active in many fields, one of which concerns investigating diversity and variability of software project participants and consequences on productivity and community engagement. It is evident from his previous research on Gnome Ecosystem [40] and other GitHub projects [39], participant heterogeneity contributes positively to many properties such as project life expectancy, without noticeable negative effects. As we know from other lines of research, diversity can also positively influence job satisfaction [1]. However, there have been projects like The Digital City ("De Digitale Stad" in Dutch, now defunct) and Stack Overflow that could profit from a wide variety of contributions but did not due to various reasons — The Digital City was shut down after the organisers realised that a project started as a way to get all sorts of people online, turned into a community of middle-age white males; and the Stack Overflow community has recently become concerned with the alienation users from non-dominant categories report. Serebrenik confronted the audience with the following three questions:

- Are differences among the software language users recognised by the language designers?
- Should the differences among the users be taken into account by the language designers?
- Have you applied any participatory techniques when designing a software language? How did it go?

Nobody had direct experience, we turned out to be practitioners of so-called "I-design" (*"I will add feature X because I think users need it"*) instead of participatory (*"users, tell me what you want!"*) or observatory (*"I observe the current user behaviour before proposing changes"*) techniques. A number of well-known cases were discussed: Radia Perlman's LOGO-inspired TORTIS language (1974, aimed at preschool children, remembered by Vadim Zaytsev), Grace Hopper's MATH-MATIC, FLOW-MATIC and COBOL (1957–59, a disruptive innovation that initiated processing programs in words, not in bytecode, also V.Z.), Helium[7] and error messages in Haskell (importance of distinguishing novice users from experts, mentioned by Juriaan Hage), UML (as a scaled up version of successful I-design, also J.H.), non-executable software languages (used a lot in the model-driven community for communication purposes, noted by Javier Luis Cánovas Izquierdo) and finally SQL (J.H.) and Excel (A.S. himself).

3.5 Domain-Specific Languages for Program Analysis

Mark Hills has been involved in the development of the Rascal metaprogramming language [24] and its application to program analysis tasks for several years. One of his recent projects is DCFlow [23], an infrastructure for control flow graph construction with a domain-specific language as an exposed interface. In his OOPSLE talk, Hills explained the language and confronted the audience with language design questions that nobody could truly answer, even though we have all at some point faced them in our own work and had to make educated guesses. For example, these were asked:

[7]http://www.cs.uu.nl/wiki/Helium

- In which cases does it make sense to create a new (internal or external) domain-specific language as opposed to directly using a language like Rascal?
- Are internal DSLs better than external or vice versa? To which kinds of problems are they best suited? When should we support both?
- What is the best way to support complex features without tailoring the DSL too closely to a specific implementation of such features?

The discussion involved many specific details of software language design and concluded that it remains an open problem, even in the presence of the overwhelming body of literature from van Wijngaarden, Hoare and Wirth to Fowler, Parr and Völter. Such work is observatory at best and mostly concerned with extracting guidelines from experience and providing toolset-specific cookbooks as opposed to empirically validating the limits of common choices in language design and implementation.

3.6 Uncertainty-Aware Programming

Naoyasu Ubayashi is a well-known authority on aspect-oriented software development. At OOPSLE, he presented a new programming paradigm: uncertainty-aware programming — which he co-invented with his colleagues Takuya Fukamachi, Shintaro Hosoai and Yasutaka Kamei.

In modern software engineering, many issues either have to be refined and concretised in order to be compiled, or can be sufficiently relaxed with parameterising and generics. However, there remains many uncertainties:

- Should a code fragment be replaced by another code fragment by applying refactoring?
- Which algorithm should be chosen to implement a solution to a common problem?
- Which code variant should be deployed at the clients who keep changing their requirements?
- Should a concern be described as an interface or a module?

Naoyasu Ubayashi thoroughly motivated his position both by practical considerations and by theorising about known knowns (the usual programming concerns), unknown knowns (software asbestos), known unknowns [14] and unknown unknowns (impossible to address without a recommender system based on a bigger codebase than the one being built). The prototype implementation is based on his previous work on Archface [38], and relies not only on aspect-orientation, but also on related work on partial modelling [18] and variability calculus.

The proposal met very warm welcome since most participants acknowledged the problem as open as well. Zinovy Diskin shared his own discussion with the bidirectional transformation community, and Vadim Zaytsev presented his yet unpublished classification of composition of nondeterministic mappings. In general, the work was deemed related to the entire body of research on variability in software product line engineering, as well as to the current modelware struggle for developing stable frameworks of uncertainty-aware bidirectional model transformation [15] based on annotations [19], metamodels [16] or deltas [12].

4. CONCLUSION

The first three years of OOPSLE have seen three somewhat different formats for the workshop. Although all three events have set aside ample time for discussion, both in each time slot and as a separate item on the agenda, the programme was different. The first instance was a small-scale event with two invited talks. The second event was larger, featuring two keynotes and multiple contributed talks through an open call. This year's event featured a single keynote and talks on an intentionally broad selection of topics. Overall, we have perhaps had a higher share of invited/encouraged contributions than is usual for workshops. We feel this has worked in our favour, particularly in fostering interesting and open discussions. However, to avoid becoming a closed community, we also intend to make sure to mix in a number of talks from a normal open submission process. The conclusion drawn by the participants at the end of the day was to proceed with a 2016 instance and advertise the event better within the SLE community.

In the future, it is important to continue dissemination of knowledge of software language engineering, its artefacts, techniques, tools and principles, at broadly scoped venues like MoDELS [42], ICMT [3], ECMFA [26], ECOOP [7], OOPSLA [35], PLDI [32] (all the papers referenced in this paragraph are examples of perfectly SLE-compatible content being presented at other venues with broader audience). This will help attracting more people to the research domain and hence will facilitate constructing and polishing the list of open problems and challenges.

5. REFERENCES

[1] S. T. Acuña, M. Gómez, and N. J. Juzgado. How Do Personality, Team Processes and Task Characteristics Relate to Job Satisfaction and Software Quality? *Information & Software Technology*, 51(3):627–639, 2009.

[2] K. C. Arnold and H. Lieberman. Managing Ambiguity in Programming by Finding Unambiguous Examples. In *OOPSLA*, pages 877–884. ACM, 2010.

[3] A. H. Bagge and R. Lämmel. Walk Your Tree Any Way You Want. In *ICMT*, volume 7909 of *LNCS*, pages 33–49. Springer, 2013.

[4] A. H. Bagge and V. Zaytsev. Workshop on Open and Original Problems in Software Language Engineering (OOPSLE 2013). In *WCRE*, pages 493–494. IEEE, 2013.

[5] A. H. Bagge and V. Zaytsev, editors. *Extended Abstracts of the 2th International Workshop on Open and Original Problems in Software Language Engineering, OOPSLE 2014*, Antwerpen, Belgium, 2014.

[6] A. H. Bagge and V. Zaytsev. International Workshop on Open and Original Problems in Software Language Engineering (OOPSLE 2014). In *CSMR-WCRE*, page 478. IEEE, 2014.

[7] G. M. Bierman, M. Abadi, and M. Torgersen. Understanding TypeScript. In *ECOOP*, volume 8586 of *LNCS*, pages 257–281. Springer, 2014.

[8] J. R. Cordy. The TXL Source Transformation Language. *Science of Computer Programming*, 61(3):190–210, 2006.

[9] Z. Diskin. Model Synchronization: Mappings, Tiles, and Categories. In *GTTSE 2009*, volume 6491 of *LNCS*, pages 92–165. Springer, 2011.

[10] Z. Diskin and T. S. E. Maibaum. Category Theory and Model-Driven Engineering: From Formal Semantics to Design Patterns and Beyond. In *ACCAT*, volume 93 of *EPTCS*, pages 1–21, 2012.

[11] Z. Diskin, A. Wider, H. Gholizadeh, and K. Czarnecki. Towards a Rational Taxonomy for Increasingly Symmetric Model Synchronization. In *Theory and Practice of Model Transformations*, volume 8568 of *LNCS*, pages 57–73. Springer, 2014.

[12] Z. Diskin, Y. Xiong, and K. Czarnecki. From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case. *JOT*, 10:1–25, 2011.

[13] F. Durán, M. Roldán, J.-C. Bach, E. Balland, M. van den Brand, J. R. Cordy, S. Eker, L. Engelen, M. de Jonge, and K. T. Kalleberg. The Third Rewrite Engines Competition. In *WRLA*, volume 6381 of *LNCS*, pages 243–261. Springer, 2010.

[14] S. G. Elbaum and D. S. Rosenblum. Known Unknowns: Testing in the Presence of Uncertainty. In S. Cheung, A. Orso, and M. D. Storey, editors, *FSE*, pages 833–836. ACM, 2014.

[15] R. Eramo, A. Pierantonio, and G. Rosa. Uncertainty in Bidirectional Transformations. In J. M. Atlee, V. Kulkarni, T. Clark, R. B. France, and B. Rumpe, editors, *MiSE*, pages 37–42. ACM, 2014.

[16] R. Eramo, A. Pierantonio, and G. Rosa. Representing Uncertainty in Bidirectional Transformations. In D. Di Ruscio and V. Zaytsev, editors, *SATToSE 2014*, CEUR Workshop Proceedings. CEUR-WS.org, 2015.

[17] S. Erdweg, T. van der Storm, M. Völter, M. Boersma, R. Bosman, W. R. Cook, A. Gerritsen, A. Hulshout, S. Kelly, A. Loh, G. Konat, P. J. Molina, M. Palatnik, R. Pohjonen, E. Schindler, K. Schindler, R. Solmi, V. Vergu, E. Visser, K. van der Vlist, G. Wachsmuth, and J. van der Woning. The State of the Art in Language Workbenches. Conclusions from the Language Workbench Challenge. In *SLE*, volume 8225 of *LNCS*. Springer, 2013.

[18] M. Famelis, R. Salay, and M. Chechik. Partial Models: Towards Modeling and Reasoning with Uncertainty. In M. Glinz, G. C. Murphy, and M. Pezzè, editors, *ICSE*, pages 573–583. IEEE, 2012.

[19] M. Famelis, R. Salay, A. D. Sandro, and M. Chechik. Transformation of Models Containing Uncertainty. In *MoDELS*, volume 8107 of *LNCS*, pages 673–689. Springer, 2013.

[20] J. Favreau. Iron Man. Movie by Marvel Studios and Fairview Entertainment, 2008.

[21] D. Hilbert. Mathematical Problems. *Bulletin of the American Mathematical Society*, 33(4):433–479, 1902.

[22] E. Hill, L. L. Pollock, and K. Vijay-Shanker. Improving Source Code Search with Natural Language Phrasal Representations of Method Signatures. In *ASE*, pages 524–527. IEEE, 2011.

[23] M. Hills. Streamlining Control Flow Graph Construction with DCFlow. In *SLE*, volume 8706 of *LNCS*, pages 322–341. Springer, 2014.

[24] P. Klint, T. van der Storm, and J. Vinju. EASY Meta-programming with Rascal. In *GTTSE 2009*, volume 6491 of *LNCS*, pages 222–289. Springer, Jan. 2011.

[25] R. Knöll and M. Mezini. Pegasus: First Steps Toward a Naturalistic Programming Language. In *OOPSLA*, pages 542–559. ACM, 2006.

[26] R. Lämmel and A. Varanovich. Interpretation of Linguistic Architecture. In *ECMFA*, volume 8569 of *LNCS*, pages 67–82. Springer, 2014.

[27] LDTA 2011. 11th International Workshop on Language Descriptions, Tools and Applications. Tool Challenge, 2011.

[28] G. Little and R. C. Miller. Keyword Programming in Java. In *ASE*, pages 84–93. ACM, 2007.

[29] B. McKenna. The Programming Language Theory Games: a monthly programming language competition, 2012.

[30] G. Mussbacher, D. Amyot, R. Breu, J.-M. Bruel, B. H. Cheng, P. Collet, B. Combemale, R. B. France, R. Heldal, J. Hill, J. Kienzle, M. Schöttle, F. Steimann, D. Stikkolorum, and J. Whittle. The Relevance of Model-Driven Engineering Thirty Years from Now. In *MoDELS*, volume 8767 of *LNCS*, pages 183–200. Springer, 2014.

[31] A. Okhotin. Conjunctive and Boolean Grammars: The True General Case of the Context-Free Grammars. *Computer Science Review*, 9:27–59, 2013.

[32] T. Parr and K. Fisher. LL(*): The Foundation of the ANTLR Parser Generator. In *PLDI*, pages 425–436. ACM, 2011.

[33] B. C. Pierce, P. Sewell, S. Weirich, and S. Zdancewic. It Is Time to Mechanize Programming Language Metatheory. In *Verified Software: Theories, Tools, Experiments*, volume 4171 of *LNCS*, pages 26–30. Springer, 2008.

[34] R. Poss. People-Specific Languages: A Case for Automated Programming Language Generation by Reverse-engineering Programmer Minds. In A. H. Bagge and V. Zaytsev, editors, *OOPSLE*, pages 15–18, 2014.

[35] T. Rendel, J. I. Brachthäuser, and K. Ostermann. From Object Algebras to Attribute Grammars. In *OOPSLA*, pages 377–395. ACM, 2014.

[36] A. Rensink and P. Van Gorp. Graph Transformation Tool Contest 2008. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(3–4):171–181, 2010.

[37] A. Stevenson and J. R. Cordy. Parse Views with Boolean Grammars. *Science of Computer Programming*, 97:59–63, 2015.

[38] N. Ubayashi, J. Nomura, and T. Tamai. Archface: a Contract Place where Architectural Design and Code Meet Together. In *ICSE*, pages 75–84. ACM, 2010.

[39] B. Vasilescu, D. Posnett, B. Ray, M. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov. Gender and Tenure Diversity in GitHub Teams. In *CHI*. ACM, 2015.

[40] B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens. On the Variation and Specialisation of Workload — A Case Study of the GNOME Ecosystem Community. *Empirical Software Engineering*, 19(4):955–1008, 2014.

[41] V. Zaytsev. Formal Foundations for Semi-parsing. In *CSMR-WCRE ERA*, pages 313–317. IEEE, Feb. 2014.

[42] V. Zaytsev and A. H. Bagge. Parsing in a Broad Sense. In *MoDELS*, volume 8767 of *LNCS*, pages 50–67. Springer, 2014.