Open and Original Problems in Software Language Engineering 2025 workshop report

Mikhail Barash[1], Vadim Zaytsev[2,**]

[1]*Bergen Language Design Laboratory, University of Bergen, Bergen, Norway*
[2]*Formal Methods & Tools, University of Twente, Enschede, The Netherlands*

Abstract

Software languages are any artificial languages used in software development. Software language engineering (SLE) is a 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 engineers as well as by forward software engineers. The SLE field is relatively new and has not yet produced a list of acknowledged open problems. This workshop is meant to expose hidden expertise in coping with unsolvable or unsolved problems which commonly remain unexposed in academic publications. OOPSLE aims to serve as a think tank in contributing to the SLE Body of Knowledge by selecting candidates for the open problem list, as well as collecting other kinds of unconventional questions and definitions that do not necessarily have clear answers or solutions, thus facilitating the exposure of dark data and surfacing new challenges.

Keywords

Software Language Engineering, domain-specific languages, modelling, generative AI

1. Workshop Introduction

1.1. Motivation

The main focus of the workshop lies in identifying and formulating challenges in the software language engineering field — these challenges could be addressed later at venues of SPLASH, STAF, MoDELS, SANER, ICSME, ICSE and others. It is a discussion platform, not a mini-conference.

The field covered by the workshop revolves around "software languages" — artificial languages used in software development: for programming, markup, pretty-printing, modelling, transformation, data description, formal specification, evolution, requirements, 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 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.

The following open problems and issues have been repeatedly mentioned at the last five editions of OOPSLE as lacking satisfactory solutions:

- Navigating the ocean of available information and promoting software language engineering views, paradigms and techniques at a broad selection of venues.
- Transformations and coupled transformations of language-aware artefacts, including language definitions, instances and updaters.
- Sound theories and mathematical foundations for software language engineering.
- Human-specific and user-aware languages and interfaces.

Joint Proceedings of the STAF 2025 Workshops: OCL, OOPSLE, LLM4SE, ICMM, AgileMDE, AI4DPS, and TTC. Koblenz, Germany, June 10-13, 2025

mikhail.barash@uib.no (M. Barash); vadim@grammarware.net (V. Zaytsev)
https://orcid.org/0000-0002-7067-2588 (M. Barash); 0000-0001-7764-4224 (V. Zaytsev)

- Support for adaptable cyber-physical systems and bringing the language and its elements closer to the intended domain within rich context.
- Software language design guidelines.
- Dealing with incompleteness, uncertainty and user mistakes on the language level.
- Improving language independence of developed tools and advancing language parametric and language agnostic methods.
- Incorporating advanced modern techniques from the fields of artificial intelligence and machine learning, into the software language engineering workflows.

Similar to some mature research fields that have a list of acknowledged open problems that are being slowly addressed by the community (e.g., the Hilbert's problems [1] or the POPLmark Challenge [2]), we aim to expose hidden expertise in coping with unsolvable or unsolved problems which commonly remain unexposed in academic publications. OOPSLE aims to serve as a think tank in selecting candidates for the open problem list, as well as other kinds of unconventional questions and definitions that do not necessarily have clear answers or solutions, facilitating the exposure of dark data.

1.2. Past Events

The sixth international workshop on Open and Original Problems in Software Language Engineering (OOPSLE'25) follows the first five editions held at WCRE 2013 [3], CSMR-WCRE 2014 [4], SANER 2015 [5], SANER 2016 and STAF 2020 [6], as well as the SLEBoK workshops at GTTSE 2009 [7], SoTeSoLa 2012 [8], SLE 2012 [9] and SPLASH 2018 [10] and the initiatives planned at the Dagstuhl seminars 17342 [11] and 20343 [12].

2. OOPSLE 2025

Below we summarise the opening discussion, as well as the workshop's keynote presentations.

2.1. General discussion

We started the workshop with a general discussion on Software Language Engineering, focusing on both technical and socio-organisation challenges.

Participants noted the human-centred nature of the development of domain-specific languages, calling for a collaboration, both internally within organisations, but also externally. The phrase "What if Siemens knew what Siemens knows?"[1] summarised the often present gap between what an organisation knows and what they are able to leverage — thus identifying a critical need for better mechanisms to structure and share knowledge and practices, especially related to modelling and domain-specific languages. It was also suggested for the SLE community to engage with psychologists and anthropologists in order to better understand how people interact with abstractions and modelling languages.

The discussion then turned to formalisation efforts like the SLE Body of Knowledge (SLEBoK) [13]. There was general support for its potential to standardise and structure the field. Some suggested that it should have dictionary-like granularity, including definitions with variants and subclauses in order to accommodate real-world complexity.

Another discussion point was on the role of ontologies and precision of terminology used by the SLE community. Despite their potential and occasional unrelenting endeavours, the SLE community struggles to engage meaningfully with ontologists.

The discussions then focused on a long-standing (but still unresolved!) technical challenges, such as supporting multiple concrete syntaxes for an abstract syntax, as well as the asymmetries between graphical and textual DSLs. It was noted that many of the tools based on the graphical syntax lack the necessary expressiveness and flexibility: for instance, they allow one to create new connections between existing elements, but do not allow for creating of new elements.

[1]*"Wenn Siemens wüsste, was Siemens weiß"*, attributed to Heinrich von Pierer, CEO of Siemens.

The participants then reflected on the concept of software *libraries*. Unlike traditional libraries, where items are borrowed and returned, software libraries are often copied, modified, and rarely reintegrated, thus breaking the original metaphor. This disconnect calls for rethinking how reuse and sharing are conceptualised and supported in software language engineering.

At the end of the discussion, a concrete proposal was made: to circulate a community survey listing candidate open problems in SLE. Respondents would assess both the significance and the extent to which each problem is considered solved. This would help prioritise future work and improve shared understanding of the field's most important problems. We intend to pursue this path in the following months.

2.2. Addressing the "Engineering" in "Software Language Engineering"

In his keynote, Bernhard Rumpe emphasised a central challenge of software language *engineering*: that languages are being treated as engineered artefacts. The talk demonstrated how the MontiCore language workbench [14] supports this vision by enabling design, composition, extension, and reuse of domain-specific languages, to allow developers to build modular and layered languages. For example, a simple automaton language (L_1) can be extended with action semantics (L_2), then embedded within broader modelling contexts (L_3), preserving backward compatibility and enabling reuse. The talk also discussed the role of *libraries* in language definitions: libraries not only supply vocabulary but also extend a language—in a lightweight and modular fashion. This elevates the language from a syntactic and semantic specification to a practical engineering tool that supports evolution and reuse.

In the discussion following the keynote, the following points were raised:

- *Diffing of models vs. programs*: while computing differences between programs is relatively straightforward due to the standardised textual formats and version control, diffing models may present a significant challenge, since models can often have richer, hierarchical structures, thus making structural and semantic comparison more complex.
- *Language composition, embedding, and aggregation*: the discussion reiterated the importance of modular language engineering. *Composition* allows reuse of language components, *embedding* enables one language to integrate another within its syntax, and *aggregation* allows to semantically link independently developed languages. These mechanisms enable flexible language design, but they also require careful management of tooling.
- *Combining instances vs. composing languages*: language composition ideally entails seamless integration on various levels, including syntactic embedding, semantic resolution, natural scoping, transfer of values over the language border, unified IDE support up to and including debugging, etc. Is it enough to have rules for composite instances combining elements of both languages, to claim that language composition has been achieved?

2.3. Who Will Create the Languages of the Future?

In his keynote, Jordi Cabot explored the evolving landscape of domain-specific languages and their future in the context of generative AI. He shared lessons from developing the BESSER low-code platform [15] and advocated for the growing role of AI in language engineering. The talk introduced the concept of "Vibe DSLing", where large language models (LLMs) assist in the automatic creation of DSLs: from abstract syntax to concrete textual and graphical notations, and code generators. The talk presented a case study where LLMs helped designing and implementing a "Funding DSL" to manage sponsorships of open-source projects. Using generating AI lowers the development costs and can strengthen the impact of DSLs, by automating testing, documentation, and simulation. The talk concludes with a pragmatic advice that language engineers should focus on high-level architectural decisions and let AI handle repetitive and/or infrastructure-heavy tasks.

The discussion following the keynote touched on several practical and philosophical aspects of language engineering in the age of AI:

- *Internal vs. external DSLs*: when the long-term growth or adoption of a DSL is uncertain, it is safer to start with an internal DSL, as this directly allows leveraging the host language's infrastructure and ecosystem.
- *Pitfall of "SemiDSLs"* — ad hoc formats with vague guidelines and no formal tooling. These pseudo-languages give an illusion of structure and abstraction, leading to leaky abstractions and maintenance risks.
- *AI makes it cheap to "reinvent the wheel"*: while generative AI helps democratise experimentation and prototyping, it also raises concerns about duplication and lack of consolidation in DSL development, encouraging quantity over quality.
- *AI for poorly designed APIs*: participants discussed AI's potential to aid in refactoring and/or generating better application programming interfaces: generative AI could provide consistent naming, usage patterns, and documentation.
- *Research around model quality*: it was noted that there is a dearth of rigorous studies on the quality of models produced in DSLs or modelling tools. This gap hinders understanding of what makes a model "good".

2.4. Let's Make Abstraction Engineering Fun Again

In his keynote, Antonio Cicchetti made a case for revitalising abstraction engineering by combining cognitive insights with practical tool support. The keynote's message was that abstraction engineering is a *skill* that must be trained and matured. Antonio stressed the importance of tools that encourage exploration, helping users internalise abstractions by interacting with them transparently. The talk discussed JJODEL [16], which is a web-based language workbench designed to implement this vision. Experience with users emphasise the importance of hands-on learning in abstraction engineering: students tend to shift focus toward language engineering tasks (especially around concrete syntax), while researchers explore challenges such as distributed modelling and multi-view consistency. Antonio closed the talk by questioning who should develop modelling tools, calling for a rethinking of incentives and community support to foster tool innovation.

The discussion after the keynote focused on the following points:

- *Tools as "mediators"*: tools are actively shape how users think and interact with the modelling process. A good tool can become an extension of the user's cognition, while a substandard one can impede it.
- *Adoption in the industry*: participants noted that introducing modelling practices can often be easier in companies with no prior modelling experience. Having no legacy biases or tool-related frustrations, such organisations can adopt modern modelling approaches more openly.
- *Collaborative modelling*: while collaboration is appealing in theory, it also introduces complexity in terms of tool support, consistency management, and user roles.

3. Conclusion

We claim that the post-pandemic resurrection of OOPSLE is a success. Recent developments in generative artificial intelligence give us ample material to talk about and seemingly even more open problems than we used to account for. There is a lot of future work desperately waiting to be done.

One of the keynote speakers was able to produce a written version of their story within the time limits of the post-proceedings. The slides of all three keynotes are available at the workshop website: https://oopsle.github.io/2025.

Declaration on Generative AI

The authors have not employed any Generative AI tools to create, change or rephrase the content of this document.

References

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

[2] B. C. Pierce, P. Sewell, S. Weirich, S. Zdancewic, It Is Time to Mechanize Programming Language Metatheory, in: B. Meyer, J. Woodcock (Eds.), Verified Software: Theories, Tools, Experiments, volume 4171 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2008, pp. 26–30. doi:10.1007/978-3-540-69149-5_3.

[3] A. H. Bagge, V. Zaytsev, Open and Original Problems in Software Language Engineering, 2013. URL: http://oopsle.github.io/2013.

[4] A. H. Bagge, V. Zaytsev, International Workshop on Open and Original Problems in Software Language Engineering (OOPSLE 2014), in: S. Demeyer, D. Binkley, F. Ricca (Eds.), Proceedings of the Software Evolution Week (IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering), Workshop Descriptions (CSMR-WCRE 2014), IEEE, 2014, p. 478. doi:10.1109/CSMR-WCRE.2014.6747223.

[5] A. H. Bagge, V. Zaytsev, Open and Original Problems in Software Language Engineering 2015 Workshop Report, SIGSOFT Software Engineering Notes 40 (2015) 32–37. doi:10.1145/2757308.2757313.

[6] V. Zaytsev, A. H. Bagge, OOPSLE 2020: Open and Original Problems in Software Language Engineering, in: L. BurgueÃ±o, L. M. Kristensen (Eds.), STAF Workshop Proceedings (STAF), volume 2707 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 47–51. URL: http://ceur-ws.org/Vol-2707/oopslepaper1.pdf.

[7] J.-M. Favre, D. Avrilionis, Research 2.0 and Software Engineering 2.0: How Community Engineering will Change our Worlds, http://gttse.wikidot.com/2009:research-2-0, 2009.

[8] J.-M. Favre, Research 2.0 Working Group at SoTeSoLa, https://github.com/SATToSE/SoTeSoLa2012/wiki/Research, 2012.

[9] J.-M. Favre, J. Vinju, SL(E)BOK 2.0 @ SLE2012, http://www.sleconf.org/2012/SLEBOK_SLE2012.html, 2012.

[10] E. Van Wyk, V. Zaytsev, Software Language Engineering Body of Knowledge Workshop, https://2018.splashcon.org/track/slebok-2018, 2018.

[11] B. Combemale, R. Lämmel, E. Van Wyk, SLEBOK: The Software Language Engineering Body of Knowledge, https://www.dagstuhl.de/17342, 2017.

[12] F. Steimann, R. Lämmel, V. Zaytsev, Software Language Engineering Body of Knowledge (SLEBoK) Recap, https://www.dagstuhl.de/20343, 2020.

[13] Vadim Zaytsev, Editor in Chief, *SLEBoK: Software Language Engineering Body of Knowledge*, 2013. URL: https://slebok.github.io, accessed: 2025-07-10.

[14] N. Jansen, A. Lüpges, B. Rumpe, Lessons Learned from Developing the MontiCore Language Workbench: Challenges of Modular Language Design, in: G. Hedin, R. Hebig, V. Zaytsev (Eds.), Proceedings of the 18th ACM SIGPLAN International Conference on Software Language Engineering (SLE), ACM, 2025, pp. 112–127. doi:10.1145/3732771.3742717.

[15] I. Alfonso, A. D. Conrardy, A. Sulejmani, A. Nirumand, F. Ul Haq, M. Gomez-Vazquez, J. Sottet, J. Cabot, Building BESSER: An Open-Source Low-Code Platform, in: H. van der Aa, D. Bork, R. Schmidt, A. Sturm (Eds.), Proceedings of the International Conference on Enterprise, Business-Process and Information Systems Modeling (BPMDS/EMMSAD), volume 511 of *Lecture Notes in Business Information Processing*, Springer, 2024, pp. 203–212. doi:10.1007/978-3-031-61007-3_16.

[16] J. Di Rocco, D. Di Ruscio, A. Di Salle, D. Di Vincenzo, A. Pierantonio, G. Tinella, jjodel - A Reflective Cloud-Based Modeling Framework, in: ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2023 Companion, Västerås, Sweden, October 1-6, 2023, IEEE, 2023, pp. 55–59. doi:10.1109/MODELS-C59198.2023.00019.