We Should Stop Claiming Generality in Our Domain-Specific Language Papers (Extended Abstract)*

Daco C. Harkes
Delft University of Technology
The Netherlands
d.c.harkes@tudelft.nl

Abstract

Our community believes that new domain-specific languages should be as general as possible to increase their impact. However, I argue that we should stop claiming generality for new domain-specific languages. Instead, we should document how domain-specific language based software development is beneficial to the overall software development process.

CCS Concepts • **Software and its engineering** → *Domain specific languages*; *Software development methods*; • **General and reference** → *General literature*;

Keywords Domain-specific languages, domain-specific language engineering, scientific rhetoric

ACM Reference Format:
Daco C. Harkes. 2018. We Should Stop Claiming Generality in Our Domain-Specific Language Papers (Extended Abstract). In *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '18), November 7–8, 2018, Boston, MA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3276954.3276967

1 Introduction

In our community, scientific arguments for new domain-specific languages (DSLs) usually include an argument for generality, despite advice to not over-generalize [4]. Moreover, generality is also a criterion for accepting DSL papers. My goal, with the essay published in The Art, Science, and Engineering of Programming [2], is to argue against this criterion, because DSLs can be very useful without generality.

More general DSLs can actually be less useful. Less general DSLs which are useful, might not be generalizable at all. The generality criterion often leads to weak claims of generality in research papers, which are undone by later work. Moreover, these claims leave an impression that work can solve everything. This is harmful for newcomers to our field and dissemination of knowledge to industry, because it makes it hard to assess whether a DSL is applicable to a specific problem.

From a theoretical point of view, there is a trade-off between more specific and more general DSLs. More specific DSLs cannot express as many programs, but the programs contain less boiler-plate code. This is because the more specific DSL makes assumptions which programs cannot work around. The same assumptions need to be made explicit in the more general DSL program, creating boiler-plate code.

In practice this trade-off between scrap-your-boilerplate and generality tends to favor the former. DSLs are not designed to be general because most DSLs are developed in tandem with the programs written in them. This is true for scientific literature as well as for the projects using DSLs in academia and industry. In scientific literature, the running example is usually the one piece of software that the researcher wanted to create. Creating a better DSL was the way to do it. Also in industry DSLs are co-developed with their applications. Often new DSL features are added when they are needed for a client application.

If DSLs are co-developed with applications, we as a research community should support and research that process. Extensible languages are one way to cater for co-development of language and application. Another way is reducing the development effort of DSLs in general by creating language workbenches [1]. In order to further streamline co-development, these language workbenches should provide live programming of DSL and applications [3].

But more importantly, we need a shift in our scientific rhetoric. We should stop claiming generality, and as reviewers we should stop judging based on generality. Instead, we should start claiming co-development is beneficial, and as reviewers we should start judging DSLs on benefit in the software engineering process. We can judge the benefit of DSLs by doing and reporting case studies. Second, if DSL

*The full version of this essay is available in The Art, Science, and Engineering of Programming [2]

Onward! '18, November 7–8, 2018, Boston, MA, USA

ACM ISBN 978-1-4503-6031-9/18/11.
https://doi.org/10.1145/3276954.3276967

authors write papers with an emphasis on language design instead of benefit to the overall software engineering process, then reviewers should reward authors for being explicit about the limits to the applicability of their DSLs.

References

[1] Martin Fowler. 2005. Language Workbenches: The Killer-App for Domain Specific Languages? http://www.martinfowler.com/articles/languageWorkbench.html. Accessed: 2018-08-08.

[2] Daco C. Harkes. 2018. We should Stop Claiming Generality in our Domain-Specific Language Papers. *The Art, Science, and Engineering of Programming* (2018).

[3] Gabriël Konat, Michael J. Steindorfer, Sebastian Erdweg, and Eelco Visser. 2018. PIE: A Domain-Specific Language for Interactive Software Development Pipelines. *The Art, Science, and Engineering of Programming* 2, 3 (2018). https://doi.org/10.22152/programming-journal.org/2018/2/9

[4] Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and how to develop domain-specific languages. *Comput. Surveys* 37, 4 (2005), 316–344. https://doi.org/10.1145/1118890.1118892